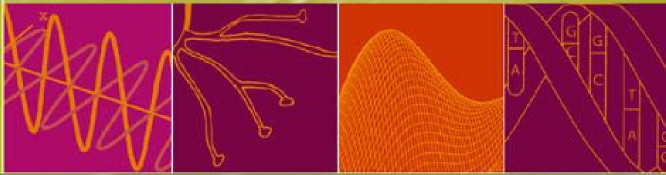


Anne Condon · David Harel · Joost N. Kok
Arto Salomaa · Erik Winfree (Eds.)

Algorithmic Bioprocesses

NATURAL COMPUTING SERIES

Springer

Natural Computing Series

For further volumes:
http://www.springer.com/series/4190

Anne Condon • David Harel • Joost N. Kok •
Arto Salomaa • Erik Winfree
Editors

Algorithmic Bioprocesses

Springer

Editors

Dr. Anne Condon
Department of Computer Science
University of British Columbia
Vancouver, BC V6T1Z4, Canada
condon@cs.ubc.ca

Prof. David Harel
Department of Computer Science
and Applied Mathematics
The Weizmann Institute of Science
Rehovot 76100, Israel
david.harel@weizmann.ac.il

Prof. Dr. Joost N. Kok
Leiden Institute of Advanced Computer
Science (LIACS)
Leiden University
2333 Leiden, The Netherlands
joost@liacs.nl

Prof. Dr. Arto Salomaa
Turku Centre for Computer Science
and Mathematics Department
University of Turku
Turku 20014, Finland
asalomaa@utu.fi

Dr. Erik Winfree
Computer Science, Computation and Neural
Systems and Bioengineering
Caltech
Pasadena, CA 91125, USA
winfree@caltech.edu

ISSN 1619-7127
ISBN 978-3-540-88868-0 e-ISBN 978-3-540-88869-7
DOI 10.1007/978-3-540-88869-7
Springer Dordrecht Heidelberg London New York

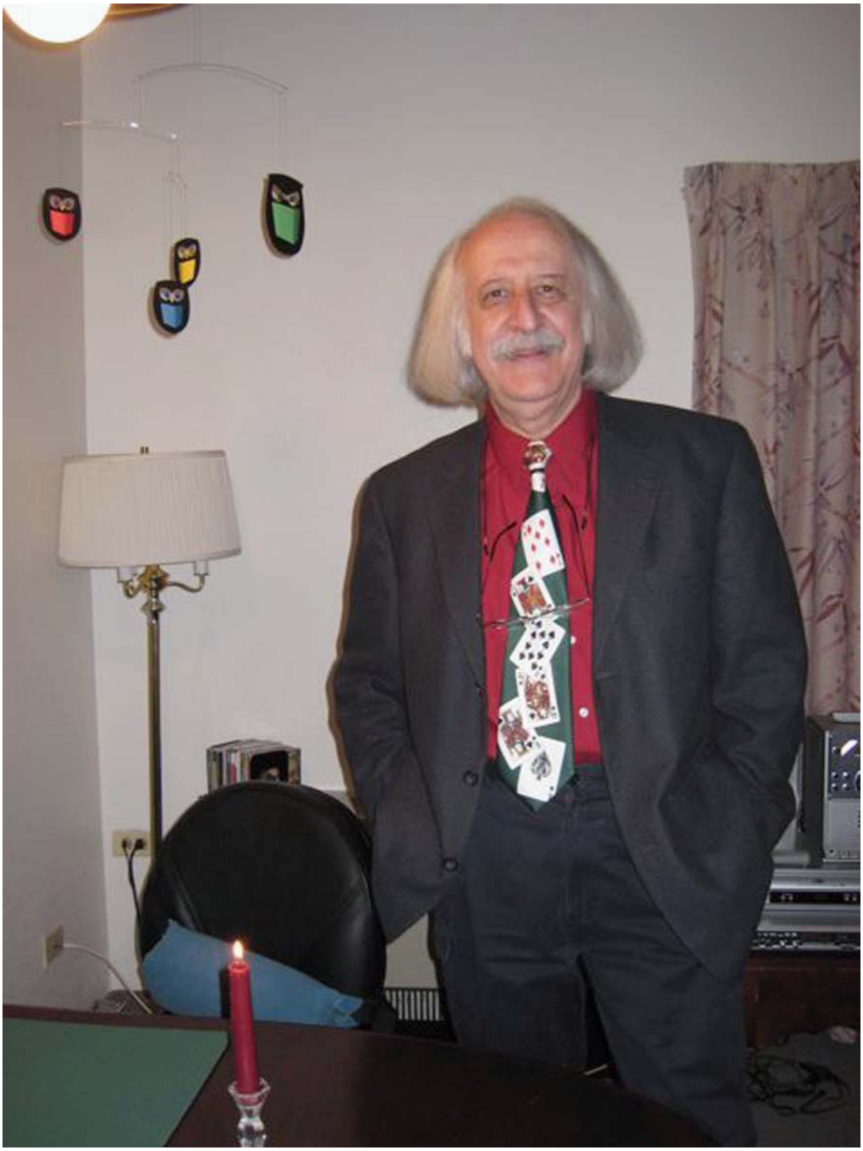
Library of Congress Control Number: 2008944308

ACM Computing Classification (1998): J.3, F.2

Cover design: KünkelLopka GmbH Heidelberg

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)



This book is dedicated to Grzegorz Rozenberg

This image was created by DADARA

Preface

This Festschrift celebrates the 65th birthday of Grzegorz Rozenberg, one of the world leaders in research on theoretical computer science and natural computing. Grzegorz had—and still has—enormous influence on the development of both disciplines. He has published over 500 research papers, 6 books, and coedited about 90 books. His papers shaped research in a whole range of areas, including formal language and automata theory, the theory of concurrent systems, the theory of graph transformations, mathematical structures in computer science, and natural computing.

He is often referred to as a guru of natural computing, as he was promoting the vision of natural computing as a coherent scientific discipline already back in the 1970s.

He is especially fond of—and very successful in—interdisciplinary research. As a matter of fact, his education and research career is interdisciplinary in a quite symbolic way: His first degree was as an electronics engineer; he got his master's degree in computer science and his Ph.D. in mathematics, and for over 30 years he has worked closely with biologists on information processing in biological systems.

Grzegorz is well known for his work for the scientific community. Apparently, this has its roots at the beginning of his career in Poland, as he felt that scientists there were quite isolated and forgotten by their colleagues in the West. He promised himself that if he ever got out of Eastern Europe he would spend a considerable part of his life working for the community.

His current or past functions for the academic community include: president of the European Association for Theoretical Computer Science (EATCS); a cofounder and president of the International Society for Nanoscale Science, Computation, and Engineering (ISNSCE); chair of the steering committee of the DNA Computing Conference; chair of the steering committee of the International Conference on Application and Theory of Petri Nets; chair of the steering committee of the European Educational Forum; a cofounder and chair of the steering committee of the International Conference on Developments in Language Theory; and a cochair of the steering committee of the International Conference on Unconventional Computation.

He is on the editorial boards of many journals and book series. As a matter of fact, he was a cofounder of some well-known journals and book series, most notably the journal Natural Computing, the journal Theoretical Computer Science C (Theory of Natural Computing), the book series Monographs and Texts in Theoretical Computer Science, the book series Natural Computing, and the book series Advances in

Petri Nets. For over 20 years, he was the editor of the Bulletin of the European Association for Theoretical Computer Science, and is a coeditor of four handbooks—on formal languages, on graph grammars and computing by graph transformations, on membrane computing, and on natural computing.

He also made valuable contributions on the national scene in The Netherlands. For example, he was the founder and the first president of the Dutch Association for Theoretical Computer Science. Also, he initiated the Dutch Theory Day, which is today the main meeting forum for researchers working on theoretical computer science in The Netherlands.

His life in science is only one of his several lives. He is a professional magician—often performing at scientific conferences. Also, he spends a considerable part of his intellectual life studying the paintings of Hieronymus Bosch. [Apparently, he is writing a book on the art of Bosch.] But, most of all, he is a family man, and an admirer of the art of his son, DADARA. The phrase "family man" must be seen here in a broader context, as he calls his Ph.D. students and postdocs, and many of the scientists who come to work with him, "his children." For more insight into Grzegorz's life and personality, please see the tribute "Grzegorz Rozenberg: A Magical Scientist and Brother" by his close friend and collaborator Arto Salomaa in Part I of this book.

To celebrate the 65th birthday of Grzegorz, we organized in December 2007 a workshop on Algorithmic Bioprocesses at the Lorentz Center of Leiden University. The theme of the workshop is one of Grzegorz's research areas, and it falls within the general field of natural computing which is certainly his favorite research field now. As a matter of fact, we wanted the workshop to reflect Grzegorz's views of computer science and natural computing, which in a nutshell, can be explained as follows.

The spectacular progress in information and communications technology (ICT) is very much supported by the evolution of computer science which designs and develops the instruments needed for this progress: computers, computer networks, software methodologies, etc. Since ICT has a tremendous impact on our everyday life, so too does computer science. However, there is much more to computer science than ICT: It is the science of information processing, and, as such, it is a fundamental science for other scientific disciplines.

On the one hand, the only common denominator among the many, diverse research areas in computer science is that they require us to think about various aspects of information processing. Therefore, the frequently used term "informatics"—most commonly used in Europe—is much better than "computer science"; the latter stipulates that a specific instrument, namely the computer, is the main research topic of our discipline. On the other hand, one of the important developments of the last century for a number of other scientific disciplines is the adoption of information and information processing as central notions and frameworks of thought—biology and physics are prime examples here. For these scientific disciplines, informatics provides not only instruments but also a way of thinking. One of the grand challenges of informatics is to understand the world around us in terms of information processing.

Informatics (computer science) is now undergoing an important transformation which is stimulated to a great extent by all kinds of interactions with the natural sciences. It is adapting and extending its traditional notions of computation and computational techniques to account for computation taking place in nature around us.

Natural computing is concerned with computing taking place in nature as well as human-designed computing inspired by nature. Using the terminology of natural sciences, specifically biochemistry, one can say that natural computing is an important catalyst for the interactions between informatics and natural sciences, and hence an important catalyst for the ongoing, exciting development of informatics.

The main idea of the "Algorithmic Bioprocesses" workshop was to reflect the above point of view, and most importantly to reflect the power and excitement of interdisciplinary research, which forms the core of natural computing. We have chosen the name "Algorithmic Bioprocesses" because the workshop did not cover the whole spectrum of natural computing research, but rather it was mostly focused on the interactions between computer science on the one hand and biology, chemistry, and DNA-oriented nanoscience on the other.

The workshop turned out to be a great success due—among other reasons – to the fact that many world leaders of research in natural computing participated. It was also a very pleasant social event. Grzegorz was visibly happy with the workshop, obviously because of its high scientific level, but also because so many of the speakers were really his "children."

The idea of a book based on the workshop had already occurred to us before the event, and participants suggested additional contributors to the book, especially scientists whose work is related to Grzegorz's research interests. As a result, this book contains some contributions that were not presented at the workshop. We believe that the resulting book provides a valuable perspective on an important part of current research in natural computing.

The book is divided into parts which, with the exception of Part I, reflect various research areas related to algorithmic bioprocesses. Part I is a tribute to Grzegorz written by his very close friend and collaborator Arto Salomaa. Part II, "Sequence Discovery, Generation, and Analysis," is concerned with many aspects of fundamental studies of biological systems. Part III, "Gene Assembly in Ciliates," covers theoretical, computational, and experimental research on the process of gene assembly, including the hypothesis on the biological realization of this process. Part IV, "Nanoconstructions and Self-assembly," discusses many aspects of nanoconstructions and self-assembly including theoretical and computational aspects as well as pragmatic considerations on implementations of physical chemistry processes. Part V, "Membrane Computing," discusses the membrane computing model inspired by the compartmentalization of cells by biological membranes. It also surveys research on a more recent model of spiking neural P systems which is motivated by the behavior of spiking neural networks. Part VI, "Formal Models and Analysis," surveys general computer science-inspired approaches to modeling and analysis of biological processes, such as the use of Petri nets, and the use of probabilistic model checking. It also presents models for specific biological phenomena such as recombination, the eukaryotic heat shock response, the MAP kinase cascade, and the behavior of the calyx of Held synapse. Part VII, “Process Calculi and Automata,” deals with the relationship between automata and process calculi on the one hand and biochemistry on the other. It considers molecules as automata, biochemical implementation of automata, process calculi as a general framework for studying biological processes, and translations from process algebra models into differential equations—a more traditional framework used by biologists. Part VIII, “Biochemical Reactions,” considers problems of stochastic simulation and modeling in relation to dynamic description of biochemical reactions. Finally, Part IX, “Broader Perspective,” provides a more general setting for the book by adding a number of additional topics from natural computing. They include molecular evolution, regulation of gene expression, light-based computing, cellular automata, realistic modeling of biological systems, and evolutionary computing.

Thus, the book covers a wide spectrum of research topics from natural computing. Since it is a Festschrift, it is fitting to note here that Grzegorz made valuable research contributions to the research themes of all parts of this book.

We hope that this book will contribute to the further development of research in the very exciting and important area of natural computing, by providing valuable perspective and knowledge to researchers in natural computing, and by motivating and encouraging others to join this research field. Most of all, we hope that the book is really a worthy tribute to Grzegorz. We want to thank all the contributors for helping us to achieve these goals.

Vancouver, Canada	Anne Condon
Rehovot, Israel	David Harel
Leiden, The Netherlands	Joost N. Kok
Turku, Finland	Arto Salomaa
Pasadena, USA	Erik Winfree

Contents

Part I Tribute

Part II Sequence Discovery, Generation, and Analysis

Part III Gene Assembly in Ciliates

Part IV Nanoconstructions and Self-assembly

Part V Membrane Computing

Part VI Formal Models and Analysis

Part VII Process Calculi and Automata

Part VIII Biochemical Reactions

Part IX Broader Perspective

Contributors

Angela Angeleska University of South Florida, Florida, FL, USA, aangeles@mail.usf.edu

Alberto Apostolico Dipartimento di Ingegneria dell' Informazione, Università di Padova, Padova, Italy, axa@dei.unipd.it and College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30318, USA

Thomas Bäck Natural Computing Group, Leiden University, Leiden, The Netherlands, baeck@liacs.nl and NuTech Solutions, Dortmund, Germany

Ralph-Johan Back Department of Information Technologies, Åbo Akademi University, Turku 20520, Finland, backrj@abo.fi

Daniela Besozzi Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano, Via Comelico 39, 20135 Milano, Italy, besozzi@dico.unimi.it

Paola Bonizzoni Dipartimento di Informatica Sistemistica e Comunicazione, Università degli Studi di Milano—Bicocca, Viale Sarca 336, 20126 Milano, Italy, bonizzoni@disco.unimib.it

Andrea Bracciali Dipartimento di Informatica, Università di Pisa, Pisa, Italy, braccia@di.unipi.it

Robert Brijder Leiden Institute of Advanced Computer Science, Universiteit Leiden, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands, rbrijder@liacs.nl

Jehoshua Bruck California Institute of Technology, Pasadena, CA, USA, bruck@caltech.edu

Marcello Brunelli Dipartimento di Biologia, Università di Pisa, Pisa, Italy, mbrunelli@biologia.unipi.it

Alessandra Carbone Génomique Analytique, Université Pierre et Marie Curie, INSERM UMRS511, 91, Bd de l'Hôpital, 75013 Paris, France, alessandra.carbone@lip6.fr

Luca Cardelli Microsoft Research, Cambridge, UK, luca@microsoft.com

Enrico Cataldo Dipartimento di Biologia, Università di Pisa, Pisa, Italy, ecataldo@biologia.unipi.it

Paolo Cazzaniga Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano—Bicocca, Viale Sarca 336, 20126 Milano, Italy, cazzaniga@disco.unimib.it

Irun R. Cohen Weizmann Institute of Science, Rehovot 76100, Israel, irun.cohen@weizmann.ac.il

Matthew Cook Institute of Neuroinformatics, UZH, ETH Zürich, Zürich, Switzerland, cook@ini.phys.ethz.ch

Mark Daley Departments of Biology and Computer Science, University of Western Ontario, London, ON, N6A 5B7, Canada, daley@csd.uwo.ca

Pierpaolo Degano Dipartimento di Informatica, Università di Pisa, Pisa, Italy, degano@di.unipi.it

Michael Domaratzki Department of Computer Science, University of Manitoba, Winnipeg, MB, R3T 2N2, Canada, mdomarat@cs.umanitoba.ca

Adam Duguid School of Informatics, The University of Edinburgh, Edinburgh, Scotland, UK, a.j.duguid@sms.ed.ac.uk

Stefan Engelen Génomique Analytique, Université Pierre et Marie Curie, INSERM UMRS511, 91, Bd de l'Hôpital, 75013 Paris, France, stefengelen@gmail.com

John E. Eriksson Turku Centre for Biotechnology, Turku, Finland and Department of Biochemistry, Åbo Akademi University, Turku 20520, Finland, john.eriksson@btk.fi

Maria Luisa Guerriero Laboratory for Foundations of Computer Science, The University of Edinburgh, Informatics Forum, 10 Crichton Street, EH8 9AB, Edinburgh, UK, mguerrie@inf.ed.ac.uk

Jurriaan Hage Department of Information and Computing Sciences, Universiteit Utrecht, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands, jur@cs.uu.nl

Masami Hagiya Graduate School of Information Science and Technology, University of Tokyo, Tokyo, Japan, hagiya@is.s.u-tokyo.ac.jp

David Harel Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel, david.harel@weizmann.ac.il

Tero Harju Department of Mathematics, University of Turku, 20014 Turku, Finland, harju@utu.fi

Tom Head Mathematical Sciences, Binghamton University, Binghamton, NY 13902-6000, USA, tom@math.binghamton.edu

Monika Heiner Department of Computer Science, Brandenburg University of Technology, Postbox 101344, 03013 Cottbus, Germany, monika.heiner@tu-cottbus.de

Jane Hillston School of Informatics, The University of Edinburgh, Edinburgh, Scotland, UK, jane.hillston@ed.ac.uk

Hendrik Jan Hoogeboom Leiden Institute of Advanced Computer Science, Universiteit Leiden, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands, hoogeboom@liacs.nl

Peter T. Hraber Theoretical Biology & Biophysics Group, Los Alamos National Laboratory, Los Alamos, NM 87545, USA, phraber@lanl.gov

Claire L. Hyder Turku Centre for Biotechnology, Turku, Finland and Department of Biochemistry, Åbo Akademi University, Turku 20520, Finland, chyder@btk.fi

Oscar H. Ibarra Department of Computer Science, University of California, Santa Barbara, CA 93106, USA, ibarra@cs.ucsb.edu

Sorin Istrail Department of Computer Science and Center for Computational Molecular Biology, Brown University, 115 Waterman Street, Box 1910, Providence, RI 02912, USA, sorin@cs.brown.edu

Masami Ito Department of Mathematics, Faculty of Science, Kyoto Sangyo University, Kyoto 603-8555, Japan, ito@ksuvx0.kyoto-su.ac.jp

Nataša Jonoska Department of Mathematics, University of South Florida, Tampa, FL 33620, USA, jonoska@math.usf.edu

Jarkko Kari Department of Mathematics, University of Turku, 20014 Turku, Finland, jkari@utu.fi and Turku Centre for Computer Science, 20520 Turku, Finland

Lila Kari Department of Computer Science, University of Western Ontario, London, Ontario, N6A 5B7, Canada, lila@csd.uwo.ca

Ehud Keinan Department of Chemistry, Technion—Israel Institute of Technology, Technion City, Haifa 32000, Israel, keinan@tx.technion.ac.il and Department of Molecular Biology and The Skaggs Institute for Chemical Biology, The Scripps Research Institute, 10550 North Torrey Pines Road, La Jolla, CA 92037, USA

Zachary Kincaid Department of Computer Science, University of Western Ontario, London, Ontario, N6A 5B7, Canada, zkincaid@uwo.ca and Department of Mathematics, University of Western Ontario, London, Ontario, N6A 5B7, Canada

Satoshi Kobayashi Department of Computer Science, University of Electro-Communications, 1-5-1, Chofugaoka, Chofu, Tokyo 182-8585, Japan, satoshi@cs.uec.ac.jp

Marta Kwiatkowska Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, UK, marta.kwiatkowska@comlab.ox.ac.uk

Thomas H. LaBean Department of Computer Science, Duke University, Durham, NC 27708, USA, thomas.labean@duke.edu

Laura F. Landweber Ecology and Evolutionary Biology, Princeton University, Princeton, NJ, USA, lfl@princeton.edu

Remco Loos EMBL-EBI, European Bioinformatics Institute, Wellcome Trust Genome Campus, Cambridge CB10 1SD, UK, remco.loos@ebi.ac.uk

Ville Lukkarila Department of Mathematics, University of Turku, 20014 Turku, Finland and Turku Centre for Computer Science, 20520 Turku, Finland, vinilu@utu.fi

Vincenzo Manca Department of Computer Science, Strada Le Grazie, 15-37134 Verona, Italy, vincenzo.manca@univr.it

Giancarlo Mauri Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano—Bicocca, Viale Sarca 336, 20126 Milano, Italy, mauri@disco.unimib.it

Avi E. Mayo Weizmann Institute of Science, Rehovot 76100, Israel, avi.mayo@weizmann.ac.il

Andrey Mikhailov Turku Centre for Biotechnology, Turku, Finland and Department of Biochemistry, Åbo Akademi University, Turku 20520, Finland, andrey.mikhailov@btk.fi

Andrzej Mizera Department of Information Technologies, Åbo Akademi University, Turku 20520, Finland, amizera@abo.fi

Kazufumi Mizunuma IBM, Tokyo, Japan, e34594@jp.ibm.com

Gethin Norman Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK, gethin.norman@comlab.ox.ac.uk

Mario J. Pérez-Jiménez Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, University of Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain, marper@us.es

Gheorghe Păun Institute of Mathematics of the Romanian Academy, PO Box 1-764, 014700 Bucureşti, Romania, george.paun@imar.ro and Department of Computer Science and Artificial Intelligence, University of Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain, gpaun@us.es

David Parker Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK, david.parker@comlab.ox.ac.uk

Dario Pescini Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano—Bicocca, Viale Sarca 336, 20126 Milano, Italy, pescini@disco.unimib.it

Ion Petre Department of Information Technologies, Åbo Akademi University, Turku 20520, Finland, ipetre@abo.fi

Davide Prandi Dipartimento di Medicina Sperimentale e Clinica, Università "Magna Graecia" di Catanzaro, Catanzaro, Italy, prandi@unicz.it

Corrado Priami Dipartimento di Ingegneria e Scienza dell'Informazione, Università di Trento, Trento, Italy and The Microsoft Research, University of Trento Centre for Computational and Systems Biology, Trento, Italy, priami@cosbi.eu

Paola Quaglia Dipartimento di Ingegneria e Scienza dell'Informazione, Università di Trento, Trento, Italy, quaglia@disi.unitn.it

Tamar Ratner Department of Chemistry, Technion—Israel Institute of Technology, Technion City, Haifa 32000, Israel, tamar@tx.technion.ac.il

John H. Reif Department of Computer Science, Duke University, Box 90129, Durham, NC 27708, USA, reif@cs.duke.edu

Sudheer Sahu Department of Computer Science, Duke University, Box 90129, Durham, NC 27708, USA, sudheer@cs.duke.edu

Masahico Saito Department of Mathematics, University of South Florida, Florida, FL, USA, saito@math.usf.edu

Yasubumi Sakakibara Department of Biosciences and Informatics, Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama 223–8522, Japan, yasu@bio.keio.ac.jp

Arto Salomaa Turku Centre for Computer Science, Joukahaisenkatu 3-5 B, 20520 Turku, Finland, asalomaa@utu.fi

Kengo Sato Japan Biological Informatics Consortium 2-45 Aomi, Koto-ku, Tokyo 135-8073, Japan, sato-kengo@aist.go.jp

Erik A. Schultes Department of Computer Science, Duke University, Durham, NC 27708, USA, schultes@hedgehogresearch.info

Nadrian C. Seeman Department of Chemistry, New York University, New York, NY 10003, USA, ned.seeman@nyu.edu

Shinnosuke Seki Department of Computer Science, University of Western Ontario, London, Ontario, N6A 5B7, Canada, sseki@csd.uwo.ca

Yaki Setty Computational Biology Group, Microsoft Research, Cambridge CB3 0FB, UK, yaki.setty@microsoft.com

Ofer M. Shir Natural Computing Group, Leiden University, Leiden, The Netherlands, oshir@liacs.nl

Lea Sistonen Turku Centre for Biotechnology, Turku, Finland and Department of Biochemistry, Åbo Akademi University, Turku 20520, Finland, lea.sistonen@btk.fi

David Soloveichik California Institute of Technology, Pasadena, CA, USA, dsolov@caltech.edu

Anne Taormina Department of Mathematical Sciences, University of Durham, Durham DH1 3LE, UK, anne.taormina@durham.ac.uk

Ryan Tarpine Department of Computer Science and Center for Computational Molecular Biology, Brown University, 115 Waterman Street, Box 1910, Providence, RI 02912, USA, ryan@cs.brown.edu

Andrew J. Turberfield Department of Physics, University of Oxford, Parks Road, Oxford OX1 3PU, UK, a.turberfield1@physics.ox.ac.uk

Reidun Twarock Department of Mathematics and Department of Biology, University of York, York YO10 5DD, UK, rt507@york.ac.uk

Erik Winfree California Institute of Technology, Pasadena, CA, USA, winfree@caltech.edu

Sara Woodworth Department of Computer Science, University of California, Santa Barbara, CA 93106, USA, snwoodworth@gmail.com, swood@cs.ucsb.edu

Peng Yin Department of Computer Science, Department of Bioengineering, Center for Biological Circuit Design, Caltech, Pasadena, CA 91125, USA, py@caltech.edu

Takashi Yokomori Department of Mathematics, Faculty of Education and Integrated Arts and Sciences, Waseda University, 1-6-1 Nishiwaseda, Shinjuku-ku, Tokyo 169-8050, Japan, yokomori@waseda.jp

Part I Tribute

Grzegorz Rozenberg: A Magical Scientist and Brother

Arto Salomaa

Abstract This is a personal description of Grzegorz Rozenberg. There is something magical in the fact that one man, Grzegorz, has been able to obtain so many and such good results in so numerous and diverse areas of science. This is why I have called him a "magical scientist." He is also a very interdisciplinary scientist. In some sense this is due to his educational background. His first degree was in electronics engineering, the second a master's in computer science, and the third a Ph.D. in mathematics. However, in the case of Grzegorz, the main drive for new disciplines comes from his tireless search for new challenges in basic science, rather than following known tracks. Starting with fundamental automata and language theory, he soon extended his realm to biologically motivated developmental languages, and further to concurrency, Petri nets, and graph grammars. During the past decade, his main focus has been on *natural computing*, a term coined by Grzegorz himself to mean either computing taking place in nature or human-designed computing inspired by nature.

1 General

Everyone who has worked with or otherwise followed closely Grzegorz Rozenberg has been profoundly impressed by his overwhelming enthusiasm in doing research. *One can do great science and have fun doing it* is one of his thoughts often expressed. Grzegorz has contemplated thoroughly the qualities of good science, and is ready to discuss and give advice about questions such as the choice of your research area or topic, what is important and what is not, or what is applicable and what is not. Sometimes a widely investigated research area begins to stagnate. Often an invigorating new idea, perhaps from a different field of science, changes the situation. These and related thoughts usually come up in discussions with Grzegorz. While some science writers talk about the "end of science," meaning that one already has walked through all significant roads, Grzegorz has quite an opposite view: we are in many respects only at the beginning of the road. In particular, this is true of computer science and natural computing. I will next outline some of Grzegorz's thoughts about these areas.

A. Salomaa (✉)
Turku Centre for Computer Science, Joukahaisenkatu 3-5 B, 20520 Turku, Finland
e-mail: asalomaa@utu.fi

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_1, © Springer-Verlag Berlin Heidelberg 2009

Computer science develops tools needed for ICT, information, and communications technology. Such tools include computer networks and software methodologies, the theoretical basis being essential. Computer science is more than ICT, it is the general science of information and information processing. The European term *informatics* is much better than *computer science*. The former term does not refer to any device or instrument for information processing. Such instruments always presuppose a paradigm of thinking about information processing, whereas if one speaks of "informatics", no devices or paradigms are stipulated.

In particular, informatics should study computing and information processing taking place in nature, and develop paradigms based on it. Research in natural computing, as Grzegorz understands it, is genuinely interdisciplinary and bridges informatics to natural sciences. Evolutionary computing and neural computing are older examples of such bridges. Human-designed computing inspired by nature includes such paradigms as molecular computing and quantum computing.

This is no place to overview Grzegorz's huge scientific production (hundreds of papers, some eighty books or edited books), or to discuss his contributions to the scientific community. (For instance, he was EATCS President much longer than anyone else, not to mention his more than twenty years as the editor of the EATCS Bulletin.) I will still return to our scientific cooperation in terms of three glimpses of it, in Sect. 3. Next, I will discuss other matters.

2 Personal Recollections

2.1 Grzegorz and Bolgani

You get a good picture about Grzegorz as a human being by reading [7]: what is important for him, what he likes to do, where he has been, and whom he has been with. He has surely had a many-faceted, even adventurous life, centered around family (most importantly Maja, Daniel, and Mundo), science, and also magic. Before I add some personal comments to the matters in [7], I mention a couple of things, covered very little or not at all in [7].

Life changed for Grzegorz when his grandson *Mundo* came into the world in August 2005. After that, much of his conversation has consisted of telling people about recent happenings with Mundo. Our email has piles of pictures of Mundo. I have a big picture of Grzegorz and myself, standing in snow during a sauna intermission in 1980. It has the text *Happiness is being a Grandfather*. I am glad that this is now true also for Grzegorz.

Owls are briefly mentioned in [7]. Grzegorz has a huge collection of owl figurines, real stuffed owls, and material dealing with owls. This exhibition is a real wonder to a visitor in Bilthoven. Grzegorz calls owls *magicians of nature*. His interest in owls was initiated by some pictures taken by Juhani Karhumäki in the 1970s.

Also, *Hieronymus Bosch* is only briefly mentioned in [7]. During the past decade, Grzegorz has become a real expert on Bosch. He has a comprehensive library of books on Bosch, in numerous languages. I was sorry to tell him that I have found no such book in Finnish. Grzegorz likes to go to "Boschian" lectures and meetings. My favorite Bosch site is

www.abcgallery.com/B/bosch/bosch.html

I do not know whether Grzegorz has seen it or likes it.

I will tell at the beginning of the next subsection about my first meeting with Grzegorz. Some years after that Grzegorz started to visit Turku. In 1976, one purpose of the visit was the Ph.D. defense of Juhani Karhumäki. Grzegorz has had many excellent Ph.D. students of his own, and has been an external examiner or opponent in many other Ph.D. defenses. The reaction of the candidates has been overwhelmingly positive. How helpful, constructive, and friendly he has been, can be seen from many contributions to [1].

Grzegorz stayed with my family many times in the 1970s and became famous, among many other things, for eating huge amounts of blueberry pie. I was called "Tarzan," and Grzegorz wanted to become a member of the Tarzan family. We decided that he will be *Bolgani*. Later on, "Bolgani" became his official name as a magician, and it also appears on his visiting card. In the following, I will use the two names interchangeably.

Although most of the time a scientist, Bolgani has also periods when he lives with *magic*. He has developed the performing art of close-up magic to a high professional level. I have watched him in numerous shows, both in private homes and at conference gatherings. I have long given up trying to explain or understand the illusions. I just enjoy the show. When Bolgani gave a performance in my home, one of the guests came up later with the "explanation" that he was wearing contact lenses of a special type. Illusions have no explanation.

Undoubtedly, Bolgani is a born magician. The maiden name of his mother is Zauberman. The name of his wife Maja means "illusion" in Hindi. His hands are very sensitive; I have not seen anyone handle a deck of cards as gently as Bolgani.

A magician likes to give a performance. Grzegorz is a great lecturer. The community knows this: he always has numerous invitations. It is difficult to match his sense of humor, and probably impossible to match his capability to include magical illusions in his lectures. Many people make easy things difficult in their lectures. Grzegorz has the talent of making difficult things easy. As Maurice Nivat has said, Grzegorz is *always enthusiastic*, *always optimistic*, *always amusing*, *and never boring*.

Bolgani often presents a sequence of illusions within the framework of a story, such as the following. Bolgani explains that in Chinese the names of the suits *hung tao* (heart) and *hei tao* (spade) are similar and, therefore, some confusion may arise. But now we try to be careful. He then shows the audience some hearts and puts them on the table, as well as some spades and puts them also on the table, far from the hearts. "So this pile is *hung tao*?" The audience agrees. "And this is *hei tao*?" Again consensus. Bolgani then entertains the audience and talks about various matters; about owls, and about the sauna. That laughter and for him, nowadays Mundo is the best medicine. That one should never assume anything. That the only place where success comes before work is in the dictionary. Then Bolgani goes back to the cards. “So this is *hung tao* and this *hei tao*?” General agreement. But when he shows the cards, it is the other way round. “Too bad, let us see what happens if we put everybody in the same pile.” He first puts the pile of hearts on the table and the spades on top. Again, there is some entertainment. “Now let us see if they interchanged again.” Bolgani picks up the cards and shows them. “This is absolutely crazy. They are all *zhao hua* (club, grass flower)!”

2.2 Brother

Grzegorz had been[1] in Holland about one and a half years, and was running a countrywide seminar in Utrecht, where foreign speakers were also invited. I got an invitation (by ordinary mail, not by phone) from G. Rozenberg in May 1971. I was not familiar with the name previously. He waited for me at the airport and arranged our meeting through loudspeakers. He looked much younger than I had anticipated. He drove a small Volkswagen. We got immediately into a very hectic discussion about parallel rewriting and L systems. This was contrary to all principles about driving, expressed by Grzegorz later. After about one hour, I started to wonder why we had not yet arrived at my hotel in Utrecht, well known for the model railroad upstairs. It turned out that we were still on some side streets of Amsterdam.

The same day Grzegorz told me more about L systems, at the university and during dinner at an Indonesian restaurant. This turned out to be an area where our scientific cooperation has been most vivid. The most important outcome is the book [8], but L systems play an important role also in [10, 11].

I had four lectures during the seminar, and Grzegorz also arranged a big party in their Marco Pololaan apartment. Although the twenty people present were quite noisy, the baby Daniel slept all the time. There I got to know many of my friends for the first time, notably *Aristid Lindenmayer*. L systems seemed to be a really fresh idea. I got carried away, and already decided to include a chapter about them in my forthcoming book on formal languages. During the next few weeks, I exchanged numerous letters with Grzegorz about the topic.

At this stage, I would like to tell two stories about our joint books. In some way, they illustrate very well Grzegorz’s friendliness, efficiency, and professionalism.

When working on the book [8], we had many specific deadlines, depending on mutual trips, etc. Once, I got very nervous that Grzegorz was too involved in other research topics, and could not possibly make the deadline. I wrote a very angry letter to him. His answer was very brief. “Tarzan has never before written such an angry letter. The reason is simple. In your letter you mention that you did not go to sauna in two days.” That was it, and he made the deadline.

[1] To justify the word *brother* appearing in the title of this paper, I quote Grzegorz from [7] where he describes his “wonderful family.” *Then I have two brothers of choice, Andrzej and Arto. It is difficult to imagine better brothers* (*an interesting aspect of this brotherhood is that Andrzej and Arto have never met*).

In March 1994, Grzegorz suggested to me: “Let us make a handbook on formal languages.” We started planning it immediately. In less than three years, a copy of [11] (three volumes, more than 2000 pages, 3.6 kilos altogether, more than 50 authors) was in the library of Turku University. Are there similar examples of such speedy editing of such an extensive handbook-type of scientific work?

Very soon Bolgani became a close family member. My wife tells him everything about our cats and often wears blouses given by him as presents. He is a coauthor in my son’s first scientific publication. My daughter always liked to travel to Utrecht because “it is the town with the Orange Julius.” My grandchildren call him *Äijä-Bolgani* (grandpa-Bolgani). My grandson was very impressed because Bolgani drew many coins from his ears, and my granddaughter because of his expanding command of Finnish. On the other hand, Bolgani’s late mother treated me as her own son. I myself have, more and more during the years, learned to ask his advice in difficult decisions and situations.

Bolgani always brings carefully selected gifts, not only to me but also to everybody in my “clan.” In 1975, he gave me a very special memory device. It was with me everywhere for thirty years, until it finally wore out. No substitute was available, and I had to be satisfied with miserable alternatives. But in 2007, Bolgani gave me a substitute, even better than the original!

My brother has an immense supply of jokes and anecdotes, always suitable to the occasion. Often a fatiguing situation entirely changes with his comments. When I came, exhausted in the airport bus, together with some other conference participants to Waterloo in 1977, Bolgani was there to meet us. Instead of greeting me, he went to the driver, pointing toward me, “Did that fellow behave well in the bus?”

Sometimes, I have experienced real surprise. I had forgotten my slippers in Bilthoven. The next time I was there they were not to be found, and Bolgani just remarked that maybe the cleaning lady had somehow misplaced them. Sometime afterwards, I was staying in Hermann Maurer’s (a close friend of both of us and a marvelous scientific collaborator) house in Graz. When I entered my room, my slippers were there!

Bolgani claims that I have a good memory, whereas he has a bad one. I am not sure of this; it could be the other way round. For instance, in Bolgani’s writings to me, the expressions “Rabbit Ph.D. Story” and “Dog Paper Story” appear many times. I do not remember what they refer to.

Some of our best times together have been in the sauna. Bolgani has a special sauna certificate and many sauna records, for instance, the shortest time between the plane landing and him sitting in the sauna, or seeing special animals from the sauna window. I let Bolgani himself speak (“löyly” is the Finnish word for sauna heat, and “supikoira” for raccoon dog):

When you come to Tarzan nest
You get sauna at its best
Where you can admire
Löyly and birch wood on fire
A lot of flora and fauna
Can be seen from Salosauna

But with Bolgani and nice weather
You can see two supikoiras together

3 Case Studies of Scientific Cooperation

3.1 Quasi-uniform Events and Mostowski

One of the early papers on automata by Grzegorz [6] deals with *quasi-uniform events*. “Languages” were often in the early days called “events,” but I will now speak of *languages*. A language L over an alphabet Σ is *quasi-uniform* if, for some $m \geq 0$,

$$L = B_0^* b_1 B_2^* b_3 \dots b_{2m-1} B_{2m}^*,$$

where each b_i is a letter of the alphabet Σ, and each B_i is a subset (possibly empty) of Σ. Observe that B_i^* reduces to the empty word when B_i is empty. Thus, there may be several consecutive letters in the regular expression, but the subsets are always separated by at least one letter. Grzegorz shows, for instance, that it is decidable whether or not a given finite automaton accepts a quasi-uniform language. The reader might want to prove this, especially because the reference [6] is not so easily available.

Much later, when studying subword occurrences and subword histories, I noticed that the language defined by a subword history of a very natural form always is a finite union of quasi-uniform languages (and hence, star-free). This is an example where we have worked separately on similar questions. Indeed, because Grzegorz and I never have time to cover all topics, I have never had a chance to explain to him the connection between quasi-uniform events and subword histories.

The paper [6] was communicated for publication by the famous logician *Andrzej Mostowski* on August 7, 1967. Grzegorz tells about him [7] as follows.

> Mostowski was a very kind man of very good manners. He always had time whenever I wanted to talk to him. I still remember the interview when he offered me a job. Looking through the papers he said at some point: “I see that you will be the youngest member of the Institute,” he paused and then he continued “but this problem will resolve itself with time.”

I also met Mostowski a few times, long before I met Grzegorz. He wore very elegant suits, quite unlike the casual attire now common. He was polite and considerate to me. For instance, he was chairing a session in a big logic conference in Helsinki in 1962, where I gave a talk about infinite-valued truth functions. I was somewhat nervous, and this was apparently noticed by Mostowski. There were some simple questions after my talk, but Mostowski did not want to embarrass me with his involved question. Instead, he came to me later asking, “How do you actually obtain the decidability?” Then it was no problem for me to explain to him the details.

3.2 Developing Cells and Lindenmayer

My cooperation with Grzegorz was centered around developmental languages and L systems for about two decades. I already told about my first meeting with Aristid Lindenmayer. Grzegorz had gotten to know him somewhat earlier [7], after studying a paper by Dirk van Daalen. Aristid became a close friend of ours. Apart from biological matters, we used to consult him about many other things, for instance, the English language.

We edited a book [9], for Aristid's 60th birthday. It also contains information about the early stages of L systems and the people involved. Aristid did not know about our plan. One evening in November 1985, Grzegorz phoned him, inviting him for a cup of coffee. Aristid was surprised to see me also in Bilthoven. Grzegorz told him that we knew about his birthday and bought him a book as a present, giving Aristid a parcel containing [9]. After that, the representatives of Springer-Verlag and other friends who had been hiding upstairs came to congratulate Aristid.

After Aristid's untimely passing away, we wanted to organize a conference dedicated to his memory. It was originally planned to take place in Lapland. But there were many practical difficulties. The conference, the first DLT, took place in Turku. All participants signed a special greeting to Jane Lindenmayer. The letter "L" in "DLT" was originally intended to refer also to L systems.

One of our early joint papers on L systems was the widely referenced article [3, 4], where Mogens Nielsen and Sven Skyum were also coauthors. Much of the work was done during the Oberwolfach conference in 1973 in my family apartment. Sometimes, Grzegorz was shouting so enthusiastically that my family thought we were fighting. The paper has a strange editorial history: part II appeared several months before part I.

One of the technical contributions of [3, 4] and related papers was to show how to get rid of the erasing rules, that is cell death. I mention one example. A *D0L system* consists of a word and a morphism (rules for the development of each letter). We get the language of the system by iterating the morphism on the word. For instance, starting with the single-letter word a and the morphism $a \to aba,\ b \to b^3$, we get the words

$$a,\ aba,\ abab^3aba,\ abab^3abab^9abab^3aba,\ \ldots.$$

(A knowledgeable reader will notice the connection with *Cantor's dust*.) A general D0L system may have arbitrarily many letters, and some of the rules may be erasing. How to get rid of the erasing rules? It was shown in [3, 4] (a very detailed proof appears in [2]) that this is always possible if one allows finitely many starting words and applies a letter-to-letter morphism to the end result.

3.3 Twin-Shuffle and Unisono Duet

Simultaneously with the Handbook [11], Grzegorz and I worked with Gheorghe Păun on a book on DNA-based computing [5]. This was in accordance with Grzegorz's enthusiasm about natural computing. We had already noticed that a property shared by most of the models of DNA computing is that they produce all recursively enumerable sets, that is, are universal in the sense of Turing machines. This property seems to be completely independent, for instance, of a model being grammatical or a machine model. One morning in the summer 1996 Grzegorz called me. We talked for about one hour. He spoke very fast, telling me that there is something *déjà vu* in *Watson–Crick complementarity* which leads to Turing universality. Roughly, the matter can be described as follows.

Consider the binary alphabet $\{0, 1\}$, as well as its "complement" $\{\bar{0}, \bar{1}\}$. For a word x over $\{0, 1\}$, we denote by $\bar{x}$ the word over $\{\bar{0}, \bar{1}\}$, where every letter of x is replaced by its "barred version." For instance, if $x = 001100$, then $\bar{x} = \bar{0}\bar{0}\bar{1}\bar{1}\bar{0}\bar{0}$. The *shuffle* of two words x and y is the set of words obtained by "shuffling" x and y, without changing the order of letters in x or y. For instance, each of the words

$$0\bar{0}0\bar{0}1\bar{1}1\bar{1}0\bar{0}0\bar{0},\ \ \bar{0}\bar{0}\bar{1}\bar{1}\bar{0}\bar{0}001100,\ \ \bar{0}001\bar{0}10\bar{1}\bar{1}\bar{0}0\bar{0}$$

is obtained by shuffling x and $\bar{x}$ with $x = 001100$. The *twin-shuffle language* TS consists of all words obtained by taking an arbitrary word over $\{0, 1\}$ and shuffling it with its complement. It was known before that TS is universal: every recursively enumerable language is obtained from it by a generalized sequential machine mapping. This means that TS stays invariant for each specific "task," only the input–output format has to be specified differently, according to the particular needs. By viewing the four bases of DNA as the letters 0, 1 and their barred versions, the interconnection between Watson–Crick complementarity and the twin-shuffle language becomes clear.

These matters were discussed in [12], and later in [5] and in many other publications. They were also discussed during ICALP'99 in Prague [13]. This was something special. Grzegorz and I were both invited speakers. But for the first time in ICALP history, the invitation was presented to us *jointly*. We took the matter very seriously. Both of us were speaking separately; Grzegorz about the wonders of his favorite ciliate Tom. But there was also a joint part where we both spoke *unisono*. This duet had required much practice. Fortunately, the hotel was luxurious, and I had a big apartment. So, there were no complaints from the neighbors.

4 Conclusion

Grzegorz writes in [7]:

> I have a wonderful family. I have written many papers. I have shuffled many decks of cards. Life has been good to me.

Let us hope that it will continue to be so in the years to come.

References

1. Brauer W et al (1992) Grzegorz: many happy returns. EATCS Bull 46:391–413

2. Kari L, Rozenberg G, Salomaa A (1997) L systems. In: Rozenberg G, Salomaa A (eds) Handbook of formal languages, vol 1. Springer, Berlin
3. Nielsen M, Rozenberg G, Salomaa A, Skyum S (1974) Nonterminals, homomorphisms and codings in different variations of OL-systems, part I. Acta Inform 3:357–364
4. Nielsen M, Rozenberg G, Salomaa A, Skyum S (1974) Nonterminals, homomorphisms and codings in different variations of OL-systems, part II. Acta Inform 4:87–106
5. Păun G, Rozenberg G, Salomaa A (1998) DNA computing. New computing paradigms. Springer, Berlin
6. Rozenberg G (1967) Decision problems for quasi-uniform events. Bull Acad Pol Sci XV:745–752
7. Rozenberg G (1999) The magic of theory and the theory of magic. In: Calude C (ed) People and ideas in theoretical computer science. Springer, Singapore, pp 227–252
8. Rozenberg G, Salomaa A (1980) The mathematical theory of L systems. Academic Press, New York
9. Rozenberg G, Salomaa A (1986) The book of L. Springer, Berlin
10. Rozenberg G, Salomaa A (1994) Cornerstones of undecidability. Prentice Hall, New York
11. Rozenberg G, Salomaa A (eds) (1997) Handbook of formal languages, vols 1–3. Springer, Berlin
12. Rozenberg G, Salomaa A (eds) (1996) Watson–Crick complementarity, universal computations and genetic engineering. Leiden University, Department of Computer Science Technical Report 96–28
13. Rozenberg G, Salomaa A (1999) DNA computing: new ideas and paradigms. In: Wiedermann J, van Emde P (eds) Automata, languages and programming, ICALP’99 proceedings. Springer lecture notes in computer science, vol 1644. Springer, Berlin, pp 106–118

Part II Sequence Discovery, Generation, and Analysis

Monotony and Surprise

Pattern Discovery Under Saturation Constraints

Alberto Apostolico

Abstract This paper reviews models and tools emerged in recent years in the author's work in connection with the discovery of interesting or anomalous patterns in sequences. Whereas customary approaches to pattern discovery proceed from either a statistical or a syntactic characterization alone, the approaches described here present the unifying feature of combining these two descriptors in a solidly intertwined, composite paradigm, whereby both syntactic structure and occurrence lists concur to define and identify a pattern in a subject. In turn, this supports a natural notion of pattern *saturation*, which enables one to partition patterns into equivalence classes over intervals of *monotonicity* of commonly adopted scores, in such a way that the subset of class representatives, consisting solely of saturated patterns, suffices to account for all patterns in the subject. The benefits at the outset consist not only of an increased descriptive power, but especially of a mitigation of the often unmanageable roster of candidates unearthed in a discovery attempt, and of the daunting computational burden that goes with it.

The applications of this paradigm as highlighted here are believed to point to a largely unexpressed potential. The specific pattern structures and configurations described include solid character strings, strings with errors, consensus sequences consisting of intermixed solid and wild characters, co- and multiple occurrences, and association rules thereof, etc. It is also outlined how, from a dual perspective, these constructs support novel paradigms of data compression, which leads to succinct descriptors, kernels, classification, and clustering methods of possible broader interest. Although largely inspired by biological sequence analysis, the ideas presented here apply to sequences of general origin, and mostly generalize to higher aggregates such as arrays, trees, and special types of graphs.

Work supported in part by the Italian Ministry of University and Research under the Bi-National Project FIRB RBIN04BYZ7, and by the Research Program of Georgia Tech.

A. Apostolico (✉)
Dipartimento di Ingegneria dell' Informazione, Università di Padova, Padova, Italy
e-mail: axa@dei.unipd.it

A. Apostolico
College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30318, USA

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_2, © Springer-Verlag Berlin Heidelberg 2009

1 Introduction

The problem of characterizing and detecting surprisingly recurrent patterns and related associations or rules arises ubiquitously in applications and is variously pursued order to compress data, unveil structure, infer succinct descriptions, extract and classify features, etc. In molecular biology, some sequence patterns are variously implicated in important facets of biological structure and function. In monitoring, surveyance and intrusion detection, the emergence of sequences of events or behaviors that depart significantly from routine is cause for alert and scrutiny. In the analysis of network traffic, unusually dense sequences of messages bearing high mutual similarity may indicate fraudulent activity or prelude to malicious attacks. Surprisingly high correlations in test answer sheets have helped economists in exposing cheating by schoolteachers [40]. The application of different measures of text similarity to a corpus of Japanese poetry has revealed connections in "honkadori" or poetic allusions previously unsuspected in the literary circles [49].

To such a broad variety of contexts, there corresponds a multiplicity of models and analytical tools. Roughly, the characterizations offered for the notion of a sequential pattern could be partitioned into statistical and syntactic. In a typical statistical characterization, a pattern is a sequence of m positions such that at each position each character from (some subset of) the alphabet may occur with a given probability or weight. This may be described by a suitable matrix or *profile*, where columns correspond to positions and rows to alphabet characters (see, e.g., [34]). The lineage of syntactic characterizations could be ascribed to the theory of error correcting codes: a pattern is a string w of length m and an occurrence of it is any string at a distance of d, the distance being measured in terms of errors of a certain type. For example, we can have only substitutions like in the *Hamming* variant [33], substitutions and indels like in the *Levenshtein* variant [39], substitutions and swaps and so on. Syntactic characterizations enable us to describe the model of a pattern, or a realization of it, or both, as a string or simple regular expression over the input alphabet perhaps enriched by special a wildcard or "do not care" character.

A somewhat related distinction may be based on whether the patterns being sought belong in a family of *a priori* defined, abstract *models* or templates, or the search is to be limited to substrings in the sample or to some more or less controlled neighborhood of those substrings. The approaches in the first class tend to be more rigorous, but also often pose unbearable computational burdens. Those in the second class tend to be computationally viable, but rest on more shaky methodological grounds. The discovery of interesting patterns in sequences appears thus to be torn between the rigidity of the model on the one hand and the abundance of candidates on the other. In particular, the variety of patterns described by strings affected by errors such as, e.g., mismatches or do not care characters escalates exponentially with their length and introduces a number of candidates disproportionate to the input size. This tends to generate daunting computational burdens, and often gives rise to tables that are impossible to visualize and digest. Faced with such "theories larger than life," one is exposed to telling Horatio that there are more things in his philosophy than are dreamed of in heaven and earth.[1]

Irrespective of the particular model or representation chosen, the tenet of pattern discovery equates overrepresentation with surprise, and hence with interest. Thus, any pattern discovery algorithm must ultimately weigh patterns against some threshold, based on a score that compares empirical and expected frequency, perhaps with some normalization. The departure of a pattern w from expectation is commonly measured by so-called z-scores, which have the form

$$z(w) = \frac{f(w) - E(w)}{N(w)}$$

where $f(w) > 0$ represents a frequency, $E(w) > 0$ an expectation and $N(w) > 0$ is the expected value of some function of w. For given z-score function, set of patterns $\mathcal{W}$, and real positive *threshold* T, patterns such that $z(w) > T$ or $z(w) < -T$ are respectively dubbed *over* or *underrepresented*, or simply *surprising*.

The escalation of the number of candidate patterns may be in part endemic, but another part seems rooted in the traditional characterizations of the notion of a pattern that are based either on syntactic or on statistical properties alone. We study here a systematic application of alternatives that result from a prudent combination of these two aspects in the model. The first ingredient of this perspective is offered by certain attributes of "saturation" that combine in a unique way the syntactic structure and the list of occurrences or frequency for a pattern. In informal terms, a pattern is *saturated* relative to its subject text, if it cannot be made more specific without losing some of its occurrences. The second ingredient is the existence of groups of syntactically related patterns within which z-scores behave monotonically, so that one may confine attention to the most surprising among saturated patterns and neglect the others.

Some notational conventions follow. Given a finite *alphabet* Σ of *characters* or *symbols*, we use Σ^+ to denote the free semigroup generated by Σ, and set $\Sigma^* = \Sigma^+ \cup \{\lambda\}$, where λ is the empty word. An element of Σ^* is called a *string* or *sequence* or *word*, and is denoted by one of the letters s, u, v, w, x, y, and z. We write $x = x_1 x_2 \ldots x_n$ when giving the symbols of x explicitly. The number n of symbols that form x is called the *length* of x and denoted by $|x|$. If $x = vwy$, then w is a *substring* or *factor* of x and the integer $1 + |v|$ is its *(starting) position* in x. Sometimes a special wildcard character "$\bullet$" is introduced, whereby we can write strings on $\Sigma \cup \{\bullet\}$. We build various kinds of *patterns* by concatenation of characters and possibly introducing errors and do not care in the resulting strings. The discussion privileges some special classes of pattern structures and configurations, including solid character strings, strings with errors, consensus sequences consisting of intermixed solid and wild characters, co- and multiple occurrences and association rules thereof, etc. The central issue is the discovery of patterns recurring with

[1] "There are more things in heaven and earth, Horatio, Than art dreamt of in your philosophy"—W. Shakespeare, Hamlet, I, v [76].

unexpectedly high frequency. Such patterns are considered informative in computational biology as well as in a multiplicity of other contexts. From a dual perspective, the overrepresentation exposed in this way may be used for purposes of data compression, and for obtaining succinct measures of information contents and structure, amenable in turn to tasks of classification and clustering.

Because of the primitive structure of strings and of the universal role they play in modeling and computation, the discussion concentrates here on this class of inputs. However, most of the basic principles and of the techniques developed in these contexts carry on to higher discrete structures such as trees, graphs, etc., to within varying degrees of adjustment.

2 Solid Patterns

The search for overrepresented patterns of unspecified length in a text requires the preliminary allocation of the occurrence counts of all substrings of a string [2, 23]. It is known that a suitably expanded suffix tree can be easily adapted to store, for a given string x and for each substring w of x, the number of distinct occurrences of w in x. The storage needed is linear in $|x|$, and the whole substring statistics for x can be computed in linear time (counting the maximum number of occurrences of each word without overlap is only slightly more expensive [19, 20]). This counterintuitive fact rests on an important combinatorial property, which may be described as follows. Given two words x and y, let the *start-set* of y in x be the set of *occurrences* of y in x, *i.e.*, $pos_x(y) = \{i : y = x_i \dots x_j\}$ for some i and j, $1 \le i \le j \le n$. Two strings y and z are equivalent on x if $pos_x(y) = pos_x(z)$. The equivalence relation instituted in this way is denoted by $\equiv_x$ and partitions the set of all strings over Σ into equivalence classes. Recall that the *index* of an equivalence relation is the number of equivalence classes in it.

Lemma 1 *The index k of the equivalence relation $\equiv_x$ obeys $k < 2n$.*

Lemma 1 is established in analogy to its right-context counterpart as seen in connection with DAWGs [23]. In the example of the string $abaabab\ aabaababaababa$, for instance, $\{ab, aba\}$ forms one such class and so does $\{abaa, abaab, abaaba\}$. This says that, on a suffix tree, it is enough to count occurrences of the $O(n)$ words terminating on a branching node, since any of the remaining $\Theta(n^2)$ words will occur always only as a prefix of some such word. This property supports the construction of compact global detectors of unusual words, as is described next.

Once a statistical index is built and empirical probabilities are computed, the next step is to annotate it with the expected values and variances and measures of discrepancy thereof, under some adopted probabilistic model. This may be still rather bulky in practice. For a given probabilistic model and measure of departure from expected frequency, it is possible to come up with an “observed” string such that all of its $\Theta(n^2)$ substrings are surprisingly overrepresented. This means that a table of the “surprising” substrings of a string can contain in principle a number of entries that grows with the square of the length of that string. As it turns out, it is possible to show that under several accepted measures of frequency deviation, also the candidate overrepresented words are restricted to the $O(n)$ words that end at internal nodes of a compact suffix tree.

To convey the intuition behind these facts, consider the naivest possible measure of "surprise" given by the difference: $\delta_w = f_w - (n - |w| + 1)\hat{p}$, where $\hat{p}$ is the product of symbol probabilities for w and f_w is the corresponding observed frequency. Let us say that an overrepresented word w belongs to some class C and let w' be the longest extension of w in C. Then $f_w = f_{w'}$, whence w' constitutes the saturation of w, that is, the most detailed version of w consistent with its set of occurrences. On the other hand, $\delta_w \leq \delta_{w'}$, since the probability of w' cannot exceed the probability of w! Thus, the score δ is monotonically nondecreasing in the transition from δ_w to $\delta_{w'}$. In other words, it suffices to weigh and filter out only the branching nodes of the suffix tree: words that end in the middle of an arc cannot be more surprising. Work performed in connection with biosequence analysis [6, 7] has shown that this approach can be extended to a broad variety of probabilistic models.

The discussion of solid words exposes in a nutshell the synergism between pattern saturation and monotonicity of scores. The interested reader will find that similar properties support the efficient inference and compact representation of variable-length Markov chain sources [5, 46], a structure originally emerged in the context of natural language processing [47] and then improved and applied successfully to the task of learning and classifying proteins [5].

3 Patterns with Mismatches

Patterns of approximate nature are frequent in all walks of information processing and ubiquitous in computational biology. In fact, in most practical cases, "similarity" among objects is more informative than sheer identity. For strings, these problems are expressed using a few basic notions of *distance*, e.g., Hamming [33] and Levenshtein [39] distance, and variations thereof. We treat first the problem of extracting from a given source x strings that occur unusually often in x within a prescribed maximum number of mismatches. In this context, we look thus for pairs (w, k) where w is a string of characters from an alphabet Σ and k is the number of errors or mismatches allowed on w. To quantify "unusually often" for a substring w of x, this is measured by comparing, e.g., the observed frequency and the expected number of occurrences for w with (either exactly, or up to) k mismatches. This problem enters the identification of promoter regions and other important aspects of molecular sequence analysis (see, e.g., [26, 35]). Unfortunately, the computation may become quite imposing, since the number of candidates escalates exponentially with the number of errors permitted. The reason for this reminisces of Plato's myth of the Cave:[2] the textstring x may only witness a few corrupted patterns, mundane

[2] Plato: *The Republic*, Book VII.

reflections of the "ideal" paradigms. The problem is, in our case, there are only a linear number of reflections and exponentially many candidates for the ideal paradigm.

The problem can be mitigated at the expense of a rather harsh approximation, namely by making the assumption that *the pattern must be a substring* of x [17]. Under this assumption, the computation of observed frequency is limited to strings that are, e.g., within k errors of substrings appearing in the text. For the corresponding term in the z-score, let $F_k(w)$ denote the number of observed subwords of x at a distance k from w, and consider the computation of $F_k(w)$ for all words of x of size $|w| = m \pm \delta$, where δ is a fixed interval. This is done by established techniques in $O(nk)$ or even expected sublinear time (see, e.g., [14, 27, 31]). There are $O(n)$ subwords of length m in x, whence the total computation is $O(n^2k)$ or expected $O(n^2)$. This information can be organized in an $n \times (2\delta + 1)$ table $\mathcal{W}(x)$ at the outset, such that the frequencies of substrings of length $[m - \delta \dots m + \delta]$ beginning at position i form the ith column of the table.

Consider now the efficient computation of expected frequencies. This resorts to the notion of correction factor, which in informal terms is the quantity by which the probability of a string has to be multiplied in order to account for a single mutation. Thus, if a character s is allowed to mutate into any of the characters in $\Sigma_s \subseteq \Sigma$, then the *$\Sigma_s$-correction factor* for s is

$$\frac{\sum_{s' \in \Sigma_s} p_{s'}}{p_s}.$$

Given a text x of length n, and a fixed number of errors k, it is possible to build a $[k \times n]$ matrix A of which the generic entry $A[i][j]$ is the correction factor to be applied to the probability of string $x[1 \dots j]$ with exactly i errors. Matrix A is readily computed in time $O(kn)$ by dynamic programming, due to the following lemma.

Lemma 2 *With $f_{x[j]}$ the correction factor of $x[j]$, the following holds*:

$$A[i][j] = \begin{cases} 1 & \textit{if } i = 0 \textit{ and } \forall j, \\ 0 & \textit{if } i \neq 0 \textit{ and } j < i, \\ f_{x[1]} & \textit{if } i = j = 1, \\ A[i][j-1] + A[i-1][j-1] f_{x[j]} & \textit{if } i > 0 \textit{ and } j > i. \end{cases}$$

Once the table A has been built, the computation of the correction factor for any substring $\bar{x}$ with $1, 2, \dots, k$ errors takes only $O(k^2)$ steps. To see this, let $C_k(b, e)$ be the global correction factor to be applied to substring $\bar{x} = x[b \dots e]$ in order to obtain the probability $\hat{p}$ of $\bar{x}$ when exactly k errors are imposed. (That is, $\hat{p}C_k(b, e) = P(\bar{x}_k)$ and this value depends on string x and on the indices (b, e).) Then

Lemma 3 *$C_0(b, e) = 1$. For $k > 0$, $C_k(b, e) = A[k][e] - \sum_{i=0}^{k-1} A[k-i][b-1] \cdot C_i(b, e)$.*

Further elaboration of this scheme leads to compute probabilities for all substrings of x having length in that range, in overall $O(nk)$ time [17].

At this point, given a textstring x and a length range $m \pm \delta$ with constant δ, we can carry out the efficient construction of a table $\mathcal{W}(x)$ containing all patterns w of length between $m - \delta$ and $m + \delta$ and such that w is a substring of x, together with their individual probabilities, and expected number of occurrences. $\mathcal{W}(x)$ can be quite bulky, but once more its size can be reduced by limiting entries to those representing local maxima w.r.t. the score.

4 Patterns Within Bounded Levenshtein Distance

A classical *string editing* problem based on Levenshtein distance consists of transforming a string x into another string y by performing a series of weighted edit operations on x of overall minimum cost. An edit operation on x can be the deletion of a symbol from x, the insertion of a symbol in x, or the substitution of a symbol of x by another symbol. This problem has a well-known $O(|x||y|)$ time sequential solution. Particular choices for the costs or initial conditions yield special instances of the problem such as the Longest Common Subsequence (LCS) and approximate string searching (see, e.g., [4]). The interest and range of applicability of string editing and its derivatives is such that entire volumes have been dedicated exclusively to it since the early 80s. Outside the realm of strings, algorithms for multidimensional arrays and for trees have been also developed (refer, e.g., to [14]).

We focus here on discovery problems that are modeled in terms of the detection of special kinds of subsequences. A pattern $y = y_1 \dots y_m$ occurs as a *subsequence* of a text $x = x_1 \dots x_n$ iff there exist indices $1 \le i_1 < i_2 < \cdots < i_m \le n$ such that $x_{i_1} = y_1, x_{i_2} = y_2, \dots, x_{i_m} = y_m$; in this case, we also say that the substring $w = x_{i_1} x_{i_1+1} \dots x_{i_m}$ of x is a *realization* of y beginning at position i_1 and ending at position i_m in x.

Observe that the problem of testing whether a given string $y = y_1 \dots y_m$ occurs as a subsequence in a text $x = x_1 \dots x_n$ is trivially solved in linear time. Actually, a simple $O(n \log |\Sigma|)$ time preprocessing of x makes it easy to decide subsequently for any x and in at most $|y| \log |\Sigma|$ character comparisons, whether y is a subsequence of x [4]. However, this problem is somewhat vacuous, in far as most any short string can be expected to occur as a subsequence of a long string. Thus, the quest for interesting subsequences becomes meaningful only if we add some constraints. A natural constraint consists of forcing y to pick its consecutive characters within a fixed maximum number of positions of x. One may then impose saturation conditions on the subsequences obeying such constraints and study their behavior [12]. In a nutshell, a *subsequence pattern* in text x is first defined as a string *together* with its set of starting occurrences in x. Then an equivalence class is any set of strings that share their starting occurrences in x as subsequence patterns. One then looks for extremal elements in each class, that is, representatives that embody all information about the elements in the class. This setup exposes some peculiarities compared to the case of solid words. To begin with, the strings in an equivalence class do not admit of a single representative. Instead, distinctly unrelated strings may share the same set of occurrences as subsequences. This makes the problem of finding compact representations harder.

For a more formal treatment, let an *occurrence* of v in x be specified by a list of positions of x matching the characters of v consecutively. An *ω-occurrence* of v in x is an occurrence such that less than ω positions of x elapse between any two consecutive characters of v. The positions of x that correspond to the first (respectively, last) character of v form the *left* (respectively, *right*) *list* of v, denoted by $\mathcal{L}_v = \{l_1, l_2, \ldots, l_L\}$ (respectively, $\mathcal{R}_v = \{l_1, l_2, \ldots, l_R\}$). For any given entry of the left list, each substring of x containing an ω-occurrence of v is an *ω-realization* of v, and the ω-occurrence corresponding to the sequence of lexicographically smallest positions is called *greedy*. The number of ω-occurrences of subsequences of length k occurring at the same starting position in x is $\mathcal{O}(\omega^{k-1})$. The maximum number of ω-occurrences of a specific subsequence v in x is $\mathcal{O}(\omega^{|v|-1} \cdot |x|)$. This upper bound is tight, being attained by the pair $x = \mathtt{A}^{|x|}$, $v = \mathtt{A}^{|v|}$. The maximum number of distinct subsequences of length k that ω-occur in x is $\mathcal{O}(\min(|\Sigma|^k, |x| \cdot \omega^{k-1}))$; this bound is attained on $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ by $x = (\mathtt{ACGT})^n$, $\omega = 4$, $n \gg k$. The number of greedy ω-occurrences of a specific subsequence v in x is $\mathcal{O}(|x|)$, and the maximum number of greedy ω-occurrences of subsequences of length k that start at the same position in x is $\mathcal{O}(\min(|\Sigma|, \omega)^{k-1})$.

It becomes natural to introduce equivalence classes of the following kind. Two subsequences v and w are *left equivalent*, denoted $v \equiv_l w$, if $\mathcal{L}_v = \mathcal{L}_w$. This leads to introduce saturated classes based on the notion of a special subsequence, which extends a nomenclature applied in [28] to solid words as follows. A string $v \in \Sigma^*$ occurring in x starting at positions in $\mathcal{L}_v \neq \emptyset$ is a *special* subsequence if for every symbol $a \in \Sigma \cup \{\$\}$, such that va has some ω-occurrence we have $\mathcal{L}_{va} \subset \mathcal{L}_v$. String v is a *nonspecial* subsequence if under the same conditions there is a symbol $a \in \Sigma \cup \{\$\}$ such that $\mathcal{L}_{va} = \mathcal{L}_v$.

Unfortunately, bounding the number of special subsequences does not seem so easy as in the case of solid words. But a string of 200 characters produced by a pseudo-random source emitting symbols from a decimal alphabet with uniform probabilities has about 4×10^4 4-subsequences, out of which only 10% are special sequences. This ratio is similar for more structured strings such as π and the golden ratio ϕ, as well as for certain proteins.

5 Patterns with Don't Cares

Conceptually, the subsequence patterns of the previous section are first detected one by one and then bundled into equivalence classes based on their respective occurrence lists. A complementary angle of approach proceeds by considering those and other varieties of *gapped* patterns as generated by some consensus or alignment. The patterns may be *rigid*, in the sense that a controlled number of do not care or wild characters are admitted at predefined fixed positions, but also *extensible*, in which case a sequence of gaps can now be stretched within prescribed bounds. In loose terms, these patterns consist of sequences of intermittently solid characters interspersed with wild characters, and appearing more or less frequently in an input sequence. In computational molecular biology and genomic studies, these patterns, sometimes also called *motifs*, play a prominent role. When looking for the surprisingly frequent ones among such motifs, it seems natural to exclude from consideration those that could be enriched by specifying additional solid characters without sacrifice of the corresponding frequency. Motifs that meet such a saturation condition have been also called *maximal*.

Rigid motifs are essentially strings over $\Sigma \cup \{\bullet\}$. Interestingly enough, an algebraic-flavored notion stronger than maximality, called *irredundancy*, was introduced for these motifs by L. Parida (see, e.g., [45]) and subsequently studied also by others. The idea is that, from the roster of all saturated rigid motifs, it is possible to extract a *base* of *irredundant* motifs with the property that any other motif can be inferred both in terms of its pattern structure and list of occurrence by a suitable subset of irredundant motifs on the base. Unfortunately, the size of the base of motifs in a sequence can be exponential in the size of the input [44]. However, when the minimum acceptable number of occurrences for a motif is just 2, then it is seen that the irredundant motifs come from the consensus patterns generated by the autocorrelations of the input. Furthermore, the size of the base is itself linear in the input [16] and for binary alphabets it can be actually built in $O(n^2)$ time for an input string of n characters and $O(n^2 \log n)$ time incrementally for the entire set of suffixes of that string [22].

Allowing for variable spacers in a motif makes it extensible. Such spacers are indicated by annotating the dot characters. Specifically, an annotated "$\bullet$" character is written as $\bullet^\alpha$ where α is a set of positive integers $\{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ or an interval $\alpha = [\alpha_l, \alpha_u]$, representing all integers between α_l and α_u including α_l and α_u. Thus, a motif m is *extensible* if it contains at least one annotated dot. Unlike ω-sequences, the spacers in an extensible motif may be specified by giving each a different maximum value. However, the real distinction between these two notions is in that motifs are meant to be generated not *a priori*, but rather by the consensus of the input string. Like subsequences, an extensible motif m may have multiple occurrences starting at a position of a sequence x. This complicates the probabilistic analysis of extensible motifs even for basic probabilistic models. However, some easy facts still support the synergism between saturation and score monotonicity. To be more specific, let v be a *condensation* of u if v is obtained by inserting one or more extra solid character in u while every starting position of an occurrence of u remains also a starting position of an occurrence of v. With reference to the general form of z-scores given in the Introduction, if $f(w)$ is interpreted now as deriving from the count of *starting positions* rather than from the *occurrences* of w, then the following facts hold [10].

Theorem 1 *Let v and u be extensible motifs under the iid model and let v be a condensation of u. Then there is a value $\hat{p} \leq 1$ such that $p_v = p_u \hat{p}$.*

Theorem 2 *If* $f(u) = f(v) > 0$, $N(v) < N(u)$, *and* $E(v)/N(v) \leq E(u)/N(u)$, *then*

$$\frac{f(v) - E(v)}{N(v)} > \frac{f(u) - E(u)}{N(u)}.$$

In the particular case of the iid model, this becomes the following theorem.

Theorem 3 *Let* u *and* v *be motifs generated with respective probabilities* p_u *and* $p_v = p_u \hat{p}$ *according to an iid process. If* $f(u) = f(v)$ *and* $p_u < 1/2$, *then*

$$\frac{f(v) - E(v)}{\sqrt{E(v)(1 - p_v)}} > \frac{f(u) - E(u)}{\sqrt{E(u)(1 - p_u)}}.$$

These facts identify intervals of monotonicity within which z-score computation may be limited to class representatives. Like with ω-sequences, no polynomial bound on the number of these classes seem to hold in general. In practice, however, a prudent combination of saturation conditions and monotonicity of scores is seen to afford significant parsimony in the generation and testing of candidate overrepresented rigid and extensible motifs.

6 Saturated Associations and Rules

Pattern saturation may be regarded as an indicator of some *implication* or *rule*. For instance, considering an arbitrary word w ending in the middle of an arc in the suffix tree of x, Lemma 1 can be rephrased by saying that any occurrence of that word in x *implies* an occurrence also of its extension to the nearest node. From this perspective, the construction of the tree may be regarded as a method to discover this rule. A germane notion is that of an *association rule*, which is an expression of the form $S_1 \to S_2$ where S_1 and S_2 are sets of data attributes that present themselves in tandem more often than expected. We refer to [1] and [43] for a broader discussion of these concepts.

It is often of interest to find pairs of patterns that have unusually frequent co-occurrences. Sometimes these pairs are also called *dyads*, and they arise in sequences of biological and more general nature, most notably, in the contexts of natural language processing and information extraction. With solid words, for instance, one problem of association discovery consists of finding, for a given textstring x of n symbols and an integer constant d, and for any pair (y, z) of subwords of x, the number of times (called *tandem index*) that y and z occur in tandem (*i.e.*, with no intermediate occurrence of either one in between) within a distance of d positions of x. Although in principle there might be n^4 distinct subword pairs in x, Lemma 1 tells us that it suffices to consider a family of only n^2 such pairs. Clearly, for any neglected pair (w', z'), there is a corresponding pair (y, z) contained in our family and such that: (i) w' is a prefix of w and z' is a prefix of z, and (ii) the tandem index of (w', z') equals that of (w, z). It is shown in [18] that this problem has a fast $O(n^2)$

solution, actually the table of all tandem indices of pairs of maximal words can be built in time proportional to its size [21].

7 Compressing and Classifying

The same kind of structural repetitiveness that is pursued in pattern discovery as a source of information may be treated instead, from a dual perspective, as redundancy to be disposed of in order to save storage space and transmission time. This is the task of data compression methods based on textual substitution [48], in which multiple replicas of a substring can be fruitfully replaced by a small pointer to a single reference copy. With the exception of the elegant family of Lempel–Ziv parses [38], the optimal implementation of the majority of such schemes translates into NP-complete problems. Still, some greedy paradigms of *steepest descent* have been shown to be susceptible of achieving guaranteed approximation ratio [37]. The basic paradigm consists of the iterated selection of the pattern giving the highest compression at that iteration. For solid words, the scheme was discussed in [19, 20] in connection with its algorithmic implications, and it was subsequently implemented in various forms (refer to [15] and references therein), particularly in an effort to cope with the known resilience that biological sequences exert toward compression (see, e.g., [42]).

Saturated string patterns are the obvious candidates for such offline compression schemes. In fact, in the pattern selected that at each iteration, there is no reason to forfeit characters that would not detract from its number of occurrences, nor to encode fewer (nonoverlapping) occurrences than the pattern has in the text. The basic steepest descent approach has a number of variations and extensions. In particular, schemes were developed where a controlled number of errors is traded in exchange for higher compression, or utilizing both rigid and flexible motifs, all the way to controlled lossy variations of online Lempel–Ziv–Welch parses [3, 8, 9, 11]. Such schemes show particularly good performance on signals and images where the gaps can be filled at the receiver by straightforward interpolation, leaving negligible residue distortion. They also demonstrate the use of compression as a possible basis for measuring similarity and classifying large sequences [8], much in the footsteps of [36, 41]. Perhaps one domain that best exposes the subtle interplay between pattern discovery, saturation, and classification is that of table compression, which is highlighted next.

By their nature, the records that appear in tables exhibit substantial interdependency and repetitiveness in field contents. Therefore, data organization in a table might expose certain features and correlations and perhaps at the same time blur others. By the duality inherent to compression and feature extraction, table compression might unveil interesting correlations and lead to interesting clustering and classification of records. Table compression presents both similarities and differences in comparison to data base compression (see, e.g., [29, 30]). Massive table compression has been addressed recently (see, e.g., [24, 25, 32, 50]) from the perspective of rearranging the order of columns and rows to bring together those affected by stronger dependency. However, the most interesting correlations might reside with clusters of records, that is, rows in the table that present commonalities in some of the columns [13].

For a formal discussion, let T be a table of *n records* denoted $t_1, t_2, \ldots, t_n$, where each record is the concatenation of ℓ *fields*, and assume for simplicity that every record is a fixed length string of ℓ *solid* characters from some unified alphabet Σ. Next, define a *mask* as any string of ℓ characters from $\Sigma \cup \{\bullet\}$. Let σ be a singleton character of Σ. For characters σ_1 and σ_2, we write $\sigma_1 \preceq \sigma_2$ if and only if σ_1 is a dot or $\sigma_1 = \sigma_2$.

It is natural to generalize to tables gapped motifs such as discussed earlier. Given a mask m, a record t_i is a *realization* or an *occurrence* of m if $m[j] \preceq t_i[j]$ holds for $1 \leq j \leq \ell$. We use $\mathcal{L}_m = (l_1, l_2, \ldots, l_p)$ to denote the list of all and only the occurrences of m in T. We are then only interested in masks having some realization in T. For a given mask m, let $m[j_1]$, $m[j_2]$, … $m[j_{\ell'}]$ be the sorted list of solid characters in m. Then a *submask* of m is any mask m' obtained by changing some $m[j_k]$ into a do not care. In other words, m is a condensation for any of its submasks. Clearly, for any condensation $\hat{m}$ of m, we have that $\mathcal{L}_{\hat{m}} \subseteq \mathcal{L}_m$. The containment relation defines a partial order on the masks of T. Clearly, we are interested in saturated masks, for which any condensation would disrupt the list of occurrences.

Given two records t_i and t_j of T, let m be the mask that is obtained by taking the "consensus" of t_i and t_j, denoted $t_i \oplus t_j$, such that $m[k] = t_i[k]$ if $t_i[j] = t_j[j]$ and $m[k] = \bullet$ otherwise. Clearly, m is saturated, since we have $i, j \in \mathcal{L}_m$, and changing any one of the do not cares in m by a solid character would expel one or both of i and j from $\mathcal{L}_m$. A mask obtained in this way is a *record intersection* or *intrecord*. In analogy with sequences, it is possible to extract from the set $\mathcal{J}$ of intrecords of T a base, as follows. A intrecord m is *redundant* if there are intrecords $m_1, m_2, \ldots, m_k$ such that m is a submask of m_f, $f = 1, 2, \ldots, k$ and $\mathcal{L}_m \subseteq \cup \mathcal{L}_{m_f}$. Then the collection of intrecords that are not redundant represents a base for our set, in the sense that they can produce any redundant mask with its occurrence list without inspection of T.

Insofar as intrecords capture regularities in the collection of specimens, one of their main interests is in their use in discovering similarities and correlations. For instance, assume that we want to analyze the relationships between the distinct values recorded in the jth field of each record. A value can be described by a point in a multidimensional space, in which dimensions are associated with intrecords, and the coordinate of a point along a dimension is the number of records in which the corresponding value and the specific intrecord cooccur in the table, normalized by the total number of records containing the value. Then a natural measure of correlation between two values may be obtained by taking the Euclidean distance between the corresponding points: computing all pairwise distances between distinct values and performing a neighbor-joining clustering, yields dendrogram structures that portray different classifications of the records.

8 Concluding Remarks

Pattern discovery represents one of the most challenging tasks of modeling and algorithmic design and yet one that is ubiquitous to crucial application domains. Often, we do not know enough of the patterns being sought to formulate crisp, adherent mathematical models. But even when a model is synthesized, the sheer number of candidates seems unmanageable to the point where it appears to defy the very purpose of embarking on the discovery process. The approaches reviewed in this paper proceed by solidly intertwining the syntactic structure of a pattern with its set of occurrences. This leads naturally to a notion of saturation and to the corresponding quest for generator patterns, which are often found to be much smaller subsets of the original pattern sets, but come virtually at no loss of information. It is seen that when looking for surprising patterns it may be enough to restrict attention to saturated ones, insofar as the latter embody the richest syntactic specification while achieving also maximum values within domains of monotonicity for z-scores and other similar measures. The result is to mitigate candidate explosion thereby alleviating the discovery process to various degrees, depending on the particular problem formulation at hand.

References

1. Agrawal R, Imielinski T, Swami A (1999) Mining association rules between sets of items in large databases. In: Proceedinngs of the ACM SIGMOD, Washington, DC, May 1993, pp 207–216
2. Apostolico A (1985) The myriad virtues of subword trees. In: Apostolico A, Galil Z (eds) Combinatorial algorithms on words. NATO ASI series F, vol 12. Springer, Berlin, pp 85–96
3. Apostolico A (2005) Of Lempel–Ziv–Welch parses with refillable gaps. In: Proceedings of IEEE DCC data compression conference, pp 338–347
4. Apostolico A (1996) String editing and longest common subsequences. In: Rozenberg G, Salomaa A (eds) Handbook of formal languages, vol II. Springer, Berlin, pp 361–398
5. Apostolico A, Bejerano G (2000) Optimal amnesic probabilistic automata or how to learn and classify proteins in linear time and space. J Comput Biol 7(3/4):381–393
6. Apostolico A, Bock ME, Lonardi S (2003) Monotony of surprise and large scale quest for unusual words. J Comput Biol 10(3–4):283–311
7. Apostolico A, Bock ME, Lonardi S, Xu X (2000) Efficient detection of unusual words. J Comput Biol 7(1–2):71–94.
8. Apostolico A, Comin M, Parida L (2006) Mining, compressing and classifying with extensible motifs. BMC Algorithms Mol Biol 1(4):1–7
9. Apostolico A, Comin M, Parida L (2004) Motifs in Ziv–Lempel–Welch clef. In: Proceedings of IEEE DCC data compression conference, pp 72–81
10. Apostolico A, Comin M, Parida L (2005) Conservative extraction of overrepresented extensible motifs. In: Proceedings of ISMB 05, intelligent systems for molecular biology, Detroit, MI, pp 9–18
11. Apostolico A, Comin M, Parida L (2006) Bridging lossy and lossless data compression by motif pattern discovery. In: Ahlswede R, Bäumer L, Cai N (eds) General theory of information transfer and combinatorics, vol II of Research report ZIF (Center of interdisciplinary studies) project, Bielefeld, October 1, 2002–August 31, 2003. Lecture notes in computer science, vol 4123. Springer, Berlin, pp 787–799

12. Apostolico A, Cunial F (2009) The subsequence composition of a string. Theor. Comp. Sci. (in press)
13. Apostolico A, Cunial F, Kaul V (2008) Table compression by record intersection. In: Proceedings of IEEE DCC data compression conference, pp 11–22
14. Apostolico A, Galil Z (eds) (1997) Pattern matching algorithms. Oxford University Press, Oxford
15. Apostolico A, Lonardi S (2000) Off-line compression by greedy textual substitution. Proc IEEE 88(11):1733–1744
16. Apostolico A, Parida L (2004) Incremental paradigms for motif discovery. J Comput Biol 11(1):15–25
17. Apostolico A, Pizzi C (2004) Monotone scoring of patterns with mismatches. In: Proceedings of WABI. Lecture notes in computer science, vol 3240. Springer, Berlin, pp 87–98
18. Apostolico A, Pizzi C, Satta G (2004) Optimal discovery of subword associations in strings. In: Proceedings of the 7th discovery science conference. Lecture notes in artificial intelligence, vol 3245. Springer, Berlin, pp 270–277
19. Apostolico A, Preparata FP (1985) Structural properties of the string statistics problem. J Comput Syst Sci 31(3):394–411
20. Apostolico A, Preparata FP (1996) Data structures and algorithms for the string statistics problem. Algorithmica 15:481–494
21. Apostolico A, Satta G (2009) Discovering subword associations in strings in time linear in the output size. J Discrete Algorithms 7(2):227–238
22. Apostolico A, Tagliacollo C (2008) Incremental discovery of irredundant motif bases for all suffixes of a string in $O(|\Sigma|n2\log n)$ time. Theor Comput Sci. doi:10.1016/j.tcs.2008.08.002
23. Blumer A, Blumer J, Ehrenfeucht A, Haussler D, Chen MT, Seiferas J (1985) The smallest automaton recognizing the subwords of a text. Theor Comput Sci 40:31–55
24. Buchsbaum AL, Caldwell DF, Church KW, Fowler GS, Muthukrishnan S (2000) Engineering the compression of massive tables: an experimental approach. In: Proceedings of 11th ACM–SIAM symposium on discrete algorithms, San Francisco, CA, pp 175–184
25. Buchsbaum AL, Fowler GS, Giancarlo R (2003) Improving table compression with combinatorial optimization. J ACM 50(6):825–851
26. Buhler J, Tompa M (2002) Finding motifs using random projections. J Comput Biol 9(2):225–242
27. Cole R, Gottlieb LA, Lewenstein M (2004) Dictionary matching and indexing with errors and don't cares. Typescript
28. Colosimo A, De Luca A (2000) Special factors in biological strings. J Theor Biol 204:29–46
29. Cormack G (1985) Data compression in a data base system. Commun ACM 28(12):1336
30. Goldstein J, Ramakrishnan R, Shaft U (1998) Compressing relations and indexes. In: Proceedings of the 14th international conference on data engineering, pp 370–379
31. Gusfield D (1997) Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, Cambridge
32. Johnson DS, Krishnan S, Chhugani J, Kumar S, Venkatasubramanian S (2004) Compressing large Boolean matrices using reordering techniques. In: Proceedings of the 30th international conference on very large databases (VLDB), pp 13–23
33. Hamming RW (1950) Error detecting and error correcting codes. Bell Syst Tech J 29:147–160
34. Hertz GZ, Stormo GD (1999) Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. Bioinformatics 15:563–577
35. Keich H, Pevzner P (2002) Finding motifs in the twilight zone. In: Annual international conference on computational molecular biology, Washington, DC, April 2002, pp 195–204
36. Kolmogorov AN (1965) Three approaches to the quantitative definition of information. Probl Pederachi Inf 1
37. Lehman E, Shelat A (2002) Approximation algorithms for grammar based compression. In: Proceedings of the eleventh ACM–SIAM symposium on discrete algorithms (SODA 2002), pp 205–212
38. Lempel A, Ziv J (1976) On the complexity of finite sequences. IEEE Trans Inf Theory 22:75–81

39. Levenshtein VI (1966) Binary codes capable of correcting deletions, insertions and reversals. Sov Phys Dokl 6:707–710
40. Levitt SD, Dubner William SJ (2005) Freakonomics: a rogue economist explores the hidden side of everything. Morrow
41. Martin-Lof P (1966) The definition of random sequences. Inf Control 9(6):602–619
42. Nevill-Manning CG, Witten IH (1999) Protein is incompressible. In: Proceedings of the IEEE data compression conference, pp 257–266
43. Piatesky-Shapiro G, Frawley WJ (eds) (1991) Knowledge discovery in databases. AAAI Press/MIT Press, Menlo Park
44. Pisanti N, Crochemore M, Grossi R, Sagot M-F (2005) Bases of motifs for generating repeated patterns with wild cards. IEEE/ACM Trans Comput Biol Bioinform 2(1):40–50
45. Rigoutsos I, Floratos A, Parida L, Gao Y, Platt D (2000) The emergence of pattern discovery techniques in computational biology. J Metab Eng 2:159–177
46. Rissanen J (1986) Complexity of strings in the class of Markov sources. IEEE Trans Inf Theory 32(4):526–532
47. Ron D, Singer Y, Tishby N (1996) The power of amnesia: learning probabilistic automata with variable memory length. Mach Learn 25:117–150
48. Storer JA (1988) Data compression: methods and theory. Computer Science Press, New York
49. Takeda M, Fukuda T, Nanri I, Yamasaki ăM, Tamari ăK (2003) Discovering instances of poetic allusion from anthologies of classical Japanese poems. Theor Comput Sci 292(2):497–524
50. Vo BD, Vo KP (2004) Using column dependency to compress tables. In: Proceedings of DCC 2004. IEEE Computer Society, Los Alamitos, pp 92–101

Information Content of Sets of Biological Sequences Revisited

Alessandra Carbone and Stefan Engelen

Abstract To analyze the information included in a pool of amino acid sequences, a first approach is to align the sequences, to estimate the probability of each amino acid to occur within columns of the aligned sequences and to combine these values through an "entropy" function whose minimum corresponds to absence of information, that is, to the case where each amino acid has the same probability to occur. Another alternative is to construct a distance tree between sequences (issued by the alignment) based on sequence similarity and to properly interpret the tree topology so to model the evolutionary property of residue conservation. We introduce the concept of "evolutionary content" of a tree of sequences, and demonstrate at what extent the more classical notion of "information content" on sequences approximates the new measure and in what manner tree topology contributes sharper information for the detection of protein binding sites.

1 Introduction

Comparison of multiple amino acid sequences resulting from years of evolution demonstrated to provide insightful information on the relationships between sequence, structure, function, and evolution of protein families [6, 8]. Multiple sequence alignments were originally used to explore phylogenetic relationships between organisms [14], and more recently, to detect more and more distant homologues, conserved functional features of proteins and major evolutionary events in genomes [2, 12, 13, 18, 19]. Also, significant improvements in predictions of both 3D fold [11] and function [20] are also achieved through multiple sequence comparison.

Pools of aligned amino acid sequences are usually constituted by very few sequence instances which are available around us today, and their grouping in sequence space highlights potential similar sequences which might exist but that we did not "see" (yet). We argue that a definition of the information content of a tree issued from a sequence alignment has to be based on the information coming from these potential viable sequences also. In this study, we check the hypothesis that the topology of a distance tree of sequences codes for interesting "biological" information of evolutionary origin, which can be extracted from a combinatorial analysis of

A. Carbone (✉)
Génomique Analytique, Université Pierre et Marie Curie, INSERM UMRS511,
91, Bd de l'Hôpital, 75013 Paris, France
e-mail: alessandra.carbone@lip6.fr

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_3, © Springer-Verlag Berlin Heidelberg 2009

the tree and a suitable interpretation of its nodes. To establish the conditions under which a pool of sequences organized in a tree is informative or not is a primary question addressed by our model.

The information content of a biological molecule corresponds to the number of sequence constraints that have to be conserved to maintain its function under random mutations [1, 5]. Expressed in amino acid units, the maximum information content that can be encoded on a N-amino acid long protein sequence is precisely N, which defines a unique sequence among the 20^N different protein sequences with N amino acids. The calculation of the maximum information content encoded on a tree of N leaves, is precisely N when we admit the leaves to be labeled by the same sequence. If the more realistic hypothesis of considering leaves labeled by different sequences is adopted, then the computation is more subtle since it depends on the topology of the tree. In this paper, we explain how to make this computation.

In what follows, sequences are made out of amino acids a_k, with $k = 1, \dots, 20$. The approach to multiple sequence analysis that we propose is general and it might be applied to arbitrary finite languages. An important general insight to retain from this analysis is that what we "observe" (that is, the actual data) is just a small amount of what we actually represent through trees constructed with clustering algorithms that group together biological objects by similarity.

2 The Model

We suppose to have N sequences which have been aligned, L be the length of the alignment, and T be the associated distance tree, whose N leaves are labeled by sequences. We think of the root of T as the set of all possible sequences "represented" by the N sequences of the original pool as follows. For each position i in the alignment, we define a characteristic function

$$\chi_i(a_k) = \begin{cases} 0 & \text{no residue } a_k \text{ appears at position } i, \\ 1 & \text{otherwise} \end{cases}$$

where $k = 1, \dots, 20$. Since an alignment contains gaps, we encode when needed, a gap as a 21st residue and we name it a_{21}. We let $P'_0 = \prod_{i=1}^{L} \sum_{k=1}^{21} \chi_i(a_k)$ be the number of potential sequences which are *coherent* with the original pool of sequences, that is those sequences which are composed by residues which appear at least once at a given position. Note that having considered a gap as a residue, we count here also aligned sequences which are formed by gaps at almost all positions. These sequences might be considered undesired and if so, it is reasonable to subtract from the pool P'_0 all sequences containing more than $\frac{L}{2} - 1$ gaps. This way, sequences cover at least the 50% of a sequence alignment and for any two sequences in the alignment the overlap is guaranteed. We call P_0 the cardinality of the resulting set of potential sequences.

As done for the root, a value P can be associated to any internal node of the tree. It corresponds to all potential sequences represented by the sequences labeling the leaves of the associated subtree.

We are interested to evaluate the information content of the pool of aligned sequences at a position i which is induced by the tree structure, where $i = 1, \dots, L$. For each i, we consider the S^i maximal subtrees of T where position i appears to be conserved (that is, all sequences labeling a maximal subtree contain the same amino acid at position i). For each i, it is easy to see that such decomposition of T into maximal subtrees is unique. For each such subtree T^i_j, we evaluate the associated P^i_j. The computation of P^i_j is done as for P_0 above. Based on the P^i_js, for $j = 1, \dots, S^i$, we compute the *evolution content at a position* i, denoted EC^i, with the entropy function

$$EC^i = \gamma\left(n_i^{\alpha} \log_2 \beta n_i - \log_2 \beta\right)$$

where α, β, γ are parameters depending on the specific tree T we are working with. We define them and comment their significance below. The value n_i is computed from $n_i^* = \frac{\sum_{j=1}^{S^i} P^i_j}{P_0}$, where $\sum_{j=1}^{S^i} P^i_j \geq N$, by considering $\log_{10} n_i^*$ (this operation gives a value in the interval $[-x, 0]$, for some x) and by rescaling the result to the interval $[0, 1]$. The rationale is that the ratio $\frac{P^i_j}{P_0}$ represents the evolutionary distance between the root of T^i_j and the root of T. The larger the ratio is, the closer the evolutionary content of the subtree is to the root. Note that for the leaves of the tree $P = 1$, since only one sequence is associated to a leaf.

The *evolution content of a tree of aligned sequences* is defined as

$$EC = \sum_{i=1}^{L} EC^i.$$

To estimate the values of the parameters α, β, γ for a given protein, we randomly select disjoint subtrees W_j in T in such a way that all leaves in T belong to some subtree W_j. Let m be the number of selected subtrees. After selection, we compute an expected value $n^{*\exp} = \frac{\sum_{j=1}^{m} P_j}{P_0}$ and rescale it (by applying first $\log_{10}$) to $n^{\exp} \in [0, 1]$. In practice, the random generation has to be repeated a sufficiently large number of times (about 100 times, for instance) and the effective $n^{\exp}$ (to be used in the analysis) can be defined to be the average of the expected $n^{\exp}$'s issued by each random selection. When a set of proteins is considered instead of a single protein, we compute the average of the $n^{\exp}$'s estimated for each protein. For different topologies of T, the value $n^{\exp}$ may vary, since it is directly associated to a distribution of subtrees in T.

The parameters α, β allow us to model information on residue conservation for a specific set of sequences and its associated tree. Namely, α, β are set so to preserve the convexity of the entropy function within $[0, 1]$ and in such a way that $n^{\exp}$ becomes the x-value where the entropy function takes its minimum. The parameter γ guarantees the y-values of the entropic function to fall into the same interval, in case several sequences are considered. The constant $\log_2 \beta$ guarantees the maximum of the entropic function to be at 0. The computation of the parameters is described in Materials and Methods.

3 Comparison of *EC* and *IC* on Residue Positions

The advantage of using the measure of information *EC* instead of the more classical *IC* is shown, as a proof of principle, for a homodimeric D-amino acid aminotransferase protein. We test the hypothesis that the most "informative" residues of the protein structure (pdb:1daa) [17] are those residues lying at the protein homodimeric interface. For this, we evaluate the prediction of the protein interaction site based on the two notions. For each position i of the sequence alignment, we compute the corresponding EC^i and IC^i and we rank accordingly all residue positions from the most to the less informative. We then evaluate, by taking gradually larger sets of top ranked positions (coverage), whether these positions lie into the known interface site of the protein or not. We found that the EC^i notion ranks with much more precision the interface site, which intuitively defines the region where (functional) information resides. Particularly, EC^i detects especially well that signals of conservation are missing in the complementary region and makes prediction of the protein interface more sharp (Fig. 1). The numerical evaluation is reported for different coverages of the protein in (Table 1). See also Fig. 2 (left).

The same analysis has being performed on a large database of 62 protein complexes, the Huang database [4], and for each protein complex the *EC* measure behaved better than the *IC* measure (with respect to all comparative scores). The analysis shows that homodimeric and heterodimeric protein interfaces gain in the

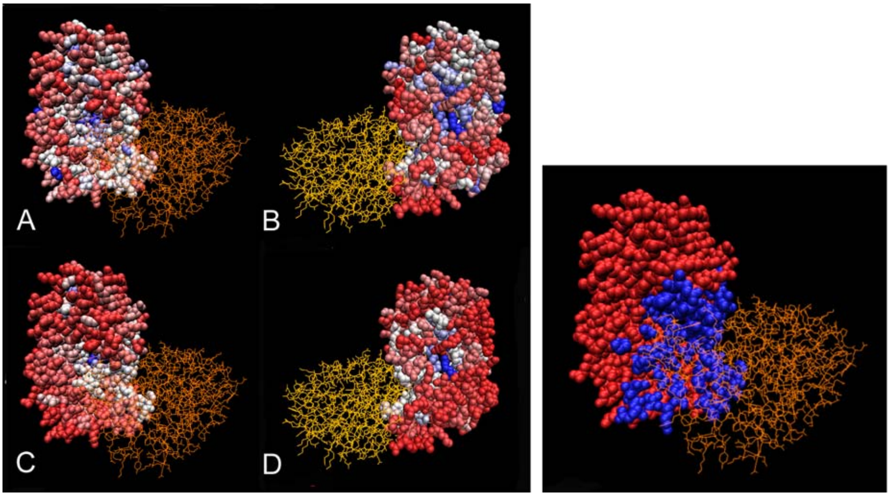


Fig. 1 *Left*: **A–B** (*top*): two views of the protein where residue positions are colored with respect to their IC^i value. Red colors are associated to residues ranking low and blue colors to residues ranking high. The color scale starts at red, passes through rose and white to reach clear blue and blue. **C–D** (*bottom*): the same two views of the protein, as in **A–B**, where residue positions are colored with respect to their EC^i value. The color scale is the same as above. Note the rather sharply identifiable interaction site of the protein which is mainly colored white in **C–D**. In contrast, the scattering of white residues does not allow an easy identification of the interaction site in **A–B**. *Right*: residues belonging to the real interface are colored blue. All others are left *red*

Table 1 Evaluation of the EC^i-ranking (top) and IC^i-ranking (bottom) on the D-amino acid aminotransferase protein structure pdb:1daa. Lines correspond to increasing coverage of the protein and describe prediction of the interaction site from 5% to 100% coverage

cover	coverSurf	sens	ScoreSens	PPV	ScorePPV	spec	ScoreSpec	acc	ScoreAcc
Residues detected using EC^i in protein structure 1daa									
0.0505	0.0424	0.1212	0.0787	0.8888	2.8551	0.9931	0.0355	0.7216	0.049
0.101	0.0943	0.2575	0.1632	0.85	2.7303	0.9794	0.0737	0.7547	0.1016
0.1516	0.132	0.3484	0.2164	0.8214	2.6385	0.9657	0.0978	0.7735	0.1347
0.2021	0.1792	0.4545	0.2753	0.7894	2.5358	0.9452	0.1244	0.7924	0.1714
0.2527	0.2264	0.5	0.2735	0.6875	2.2083	0.8972	0.1236	0.7735	0.1703
0.3032	0.2688	0.5303	0.2614	0.614	1.9723	0.8493	0.1181	0.75	0.1627
0.3501	0.3254	0.606	0.2805	0.5797	1.862	0.8013	0.1268	0.7405	0.1746
0.4007	0.3584	0.6515	0.293	0.5657	1.8173	0.7739	0.1324	0.7358	0.1824
0.4512	0.4056	0.6818	0.2761	0.5232	1.6807	0.7191	0.1248	0.7075	0.1719
0.5018	0.4481	0.7272	0.2791	0.5052	1.6229	0.678	0.1261	0.6933	0.1738
0.5523	0.4952	0.7575	0.2622	0.4761	1.5295	0.6232	0.1185	0.665	0.1633
0.6028	0.533	0.7878	0.2548	0.4601	1.4781	0.5821	0.1151	0.6462	0.1586
0.6534	0.5943	0.8484	0.2541	0.4444	1.4275	0.5205	0.1148	0.6226	0.1582
0.7003	0.6462	0.8636	0.2174	0.416	1.3364	0.452	0.0982	0.5801	0.1353
0.7509	0.6981	0.8636	0.1655	0.3851	1.237	0.3767	0.0748	0.5283	0.103
0.8014	0.7547	0.8939	0.1392	0.3687	1.1844	0.3082	0.0629	0.4905	0.0866
0.8519	0.8113	0.9393	0.128	0.3604	1.1578	0.2465	0.0578	0.4622	0.0797
0.9025	0.8726	0.9393	0.0667	0.3351	1.0764	0.1575	0.0301	0.4009	0.0415
0.953	0.9386	0.9696	0.031	0.3216	1.033	0.0753	1.40E–02	0.3537	0.0193
1	1	1	0	0.3113	0.9999	0	0	0.3113	0
Residues detected using IC^i in protein structure 1daa									
0.0505	0.0471	0.0757	0.0285	0.5	1.606	0.9657	0.0129	0.6886	0.0177
0.101	0.0849	0.1515	0.0666	0.5555	1.7844	0.9452	0.0301	0.6981	0.0414
0.1516	0.1132	0.1666	0.0534	0.4583	1.4722	0.9109	0.0241	0.6792	0.0332
0.2021	0.1603	0.2575	0.0972	0.5	1.606	0.8835	0.0439	0.6886	0.0605
0.2527	0.1981	0.3181	0.12	0.5	1.606	0.8561	0.0542	0.6886	0.0747
0.3032	0.2169	0.3333	0.1163	0.4782	1.5362	0.8356	0.0525	0.6792	0.0724
0.3501	0.2641	0.3939	0.1297	0.4642	1.4913	0.7945	0.0586	0.6698	0.0808
0.4007	0.3018	0.4393	0.1375	0.4531	1.4554	0.7602	0.0621	0.6603	0.0856
0.4512	0.349	0.5	0.1509	0.4459	1.4324	0.7191	0.0682	0.6509	0.0939
0.5018	0.3915	0.5454	0.1539	0.4337	1.3931	0.678	0.0695	0.6367	0.0958
0.5523	0.4481	0.5757	0.1276	0.4	1.2848	0.6095	0.0576	0.599	0.0794
0.6028	0.5047	0.6212	0.1165	0.3831	1.2307	0.5479	0.0526	0.5707	0.0725
0.6534	0.566	0.6515	0.0854	0.3583	1.1509	0.4726	0.0386	0.5283	0.0532
0.7003	0.6273	0.6969	0.0696	0.3458	1.1109	0.4041	0.0314	0.4952	0.0433
0.7509	0.6886	0.7272	0.0385	0.3287	1.056	0.3287	0.0174	0.4528	0.024

Table 1 (Continued)

cover	coverSurf	sens	ScoreSens	PPV	ScorePPV	spec	ScoreSpec	acc	ScoreAcc
0.8014	0.7452	0.7424	−0.0028	0.3101	0.9961	0.2534	−0.0012	0.4056	−0.0017
0.8519	0.8113	0.7575	−0.0537	0.2906	0.9337	0.1643	−0.0242	0.349	−0.0334
0.9025	0.8773	0.8636	−0.0137	0.3064	0.9843	0.1164	−0.0062	0.349	−0.0085
0.953	0.9433	0.9393	−0.004	0.31	0.9957	0.0547	−0.0018	0.3301	−0.0025
1	1	1	0	0.3113	0.9999	0	0	0.3113	0

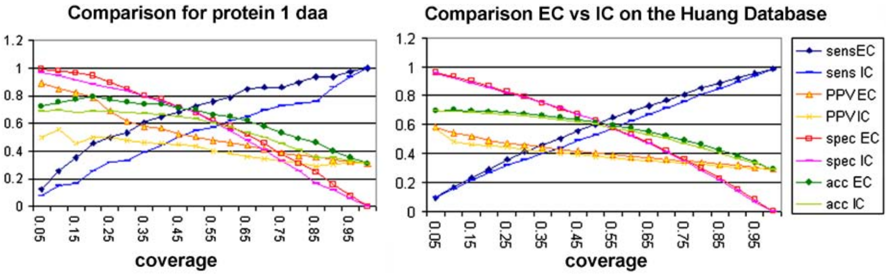


Fig. 2 *Left*: Comparative evaluation of the predictions of the protein interaction site for the D-amino acid aminotransferase protein pdb:1daa based on the notions of *EC* and *IC*. *Right*: Comparative evaluation of the (average of the) predictions of the protein interaction site for all proteins in Huang Database based on the notions of *EC* and *IC*. The evaluation is realized on different coverage levels (x-axis)

EC evaluation, while transient protein interfaces are detected with sensitivity and PPV scores which are very low, that is close to random. See Fig. 2 (right) for average evaluation scores computed on all protein complexes of the dataset, Table 2 for a numerical evaluation and Fig. 3 for average evaluation scores computed on the three classes of protein interfaces.

4 Materials and Methods

Protein Complexes Dataset for Testing

The Huang database [4] of 62 protein complexes constituted by 41 homodimers (82 chains), 11 heterodimers (23 chains) and 8 transient complexes (17 chains) has been used to test the *EC* notion versus *IC*.

Evaluation

To properly compare the IC^i, EC^i notions on specific proteins, we rely on the following quantities: the number of residues correctly predicted as interacting (true

Table 2 Evaluation of the EC^i-ranking (top) and IC^i-ranking (bottom) on the Huang database. Lines correspond to increasing coverage of the protein and describe average predictions of the interaction site from 5% to 100% coverage computed for all 62 protein complexes of the database

cover	coverSurf	sens	ScoreSens	PPV	ScorePPV	spec	ScoreSpec	acc	ScoreAcc
Residues detected using EC^i in Huang database									
0.0521	0.0432	0.0926	0.0494	0.5857	2.1094	0.9602	0.0034	0.699	0.0258
0.1021	0.0861	0.1677	0.0814	0.5437	1.922	0.9315	0.0177	0.7001	0.044
0.1518	0.127	0.2323	0.1052	0.5139	1.796	0.9008	0.0279	0.6975	0.0575
0.202	0.1697	0.2938	0.124	0.4921	1.7111	0.8668	0.0366	0.6918	0.0683
0.2516	0.2129	0.3547	0.1417	0.4736	1.6501	0.8302	0.0432	0.6841	0.0774
0.3019	0.2547	0.41	0.1552	0.4602	1.5962	0.7951	0.0499	0.6758	0.0854
0.352	0.2983	0.4585	0.1601	0.441	1.5207	0.7515	0.0499	0.6617	0.0884
0.4019	0.3421	0.5053	0.1631	0.4261	1.4617	0.7103	0.0526	0.6475	0.091
0.4519	0.3857	0.5543	0.1685	0.4156	1.4207	0.67	0.0558	0.634	0.0944
0.5009	0.4305	0.6037	0.1731	0.4055	1.3845	0.6276	0.0581	0.6184	0.0967
0.5521	0.4779	0.6465	0.1685	0.3922	1.3355	0.5792	0.0572	0.5972	0.0944
0.6017	0.5256	0.6873	0.1616	0.3807	1.2901	0.5295	0.0552	0.5745	0.0914
0.652	0.5753	0.7301	0.1547	0.3702	1.2521	0.4745	0.0499	0.5505	0.0877
0.7021	0.6268	0.7701	0.1432	0.359	1.212	0.4179	0.0447	0.5232	0.0812
0.7513	0.6814	0.8102	0.1288	0.3481	1.1726	0.3585	0.04	0.4933	0.0733
0.8017	0.7378	0.8534	0.1155	0.3389	1.14	0.2972	0.0352	0.4628	0.0659
0.8522	0.7979	0.8905	0.0925	0.3278	1.0997	0.2277	0.0256	0.4257	0.0532
0.9019	0.8589	0.9232	0.0642	0.3162	1.059	0.1539	0.0129	0.3843	0.0371
0.9519	0.9222	0.9567	0.0343	0.3054	1.0218	0.0775	−1.00E–04	0.3409	0.0198
1	0.9851	0.9851	0	0.2948	0.985	0	−0.0148	0.2948	0
Residues detected using IC^i in Huang database									
0.0521	0.045	0.0911	0.046	0.5684	2.0399	0.9576	0.0027	0.6968	0.0243
0.102	0.0874	0.1529	0.0655	0.4862	1.7151	0.9232	0.0107	0.6907	0.035
0.1518	0.1283	0.2126	0.0842	0.4662	1.6318	0.89	0.0184	0.6849	0.0453
0.202	0.1672	0.2671	0.0998	0.4516	1.5824	0.8549	0.0222	0.6776	0.0533
0.2516	0.2046	0.3177	0.113	0.4413	1.5407	0.8233	0.028	0.6704	0.0606
0.3019	0.2405	0.3575	0.1169	0.4256	1.4772	0.79	0.0306	0.6597	0.0634
0.3521	0.2788	0.4011	0.1223	0.4152	1.425	0.7556	0.0345	0.6494	0.0678
0.4019	0.3187	0.44	0.1212	0.3995	1.3658	0.7163	0.0351	0.6339	0.0674
0.4519	0.3629	0.489	0.126	0.3892	1.3324	0.6709	0.0339	0.6181	0.0691
0.5009	0.4066	0.5319	0.1253	0.3788	1.2927	0.627	0.0337	0.6007	0.0689
0.5521	0.4569	0.5783	0.1213	0.3689	1.25	0.5771	0.0341	0.5798	0.0682
0.6017	0.5074	0.6245	0.117	0.36	1.2147	0.5267	0.0342	0.558	0.0667
0.652	0.5605	0.6714	0.1108	0.3503	1.1814	0.4716	0.0322	0.5321	0.0628
0.7021	0.6167	0.7182	0.1015	0.3422	1.1482	0.4134	0.0302	0.5055	0.059
0.7513	0.6751	0.7641	0.0889	0.3327	1.1157	0.3496	0.0248	0.4739	0.0515

Table 2 (Continued)

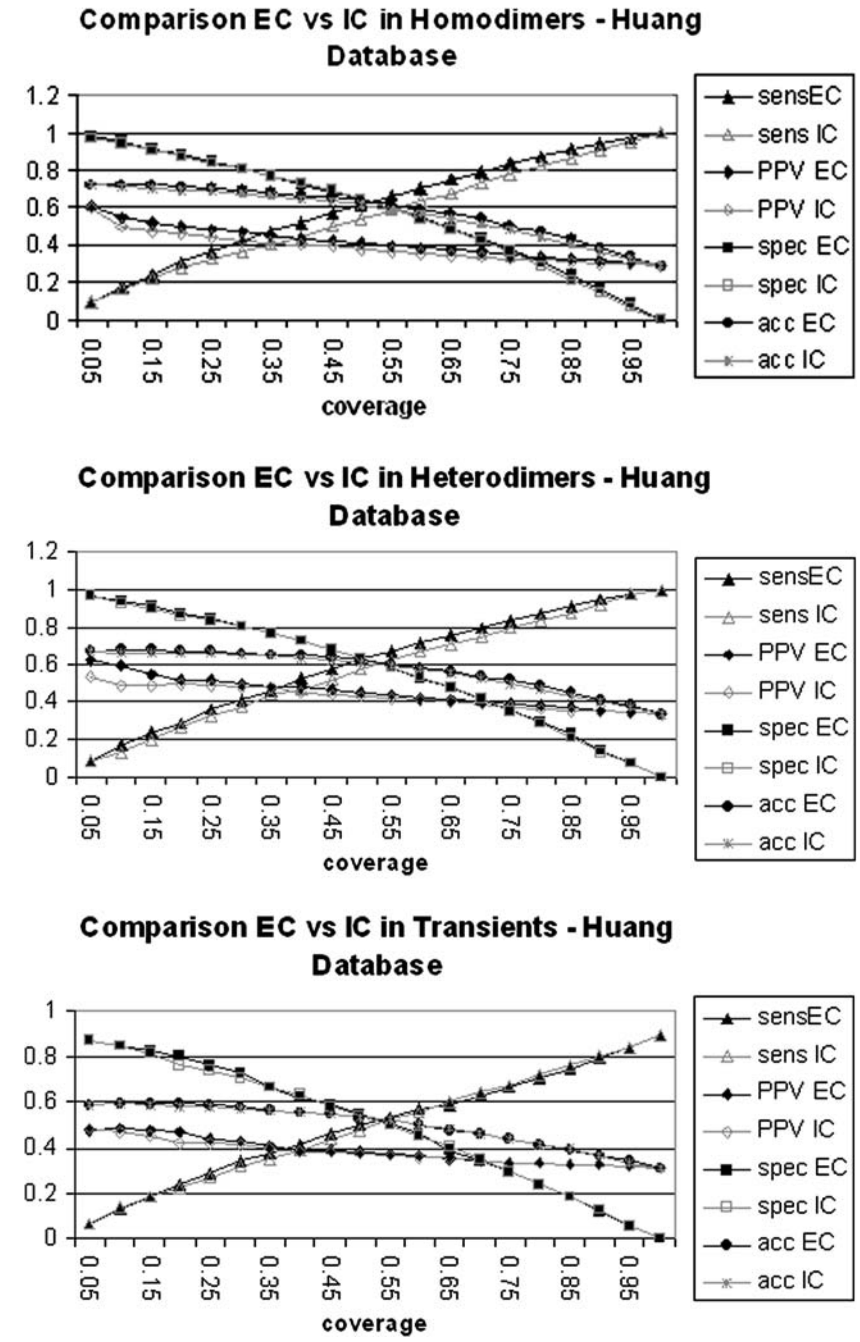
cover	coverSurf	sens	ScoreSens	PPV	ScorePPV	spec	ScoreSpec	acc	ScoreAcc
0.8017	0.7355	0.8075	0.0719	0.3226	1.0819	0.2809	0.0166	0.4389	0.0411
0.8522	0.798	0.848	0.0499	0.3131	1.0469	0.2095	0.0077	0.4014	0.0293
0.9019	0.8605	0.8929	0.0323	0.3057	1.0221	0.1385	−8.00E–04	0.3651	0.0188
0.9519	0.9236	0.9386	0.0149	0.2998	1.001	0.0681	−0.0081	0.3296	0.009
1	0.9851	0.9851	0	0.2949	0.985	0	−0.0148	0.2949	0

positives, *TP*), the number of residues correctly predicted as noninteracting (true negatives, *TN*), the number of noninteracting residues incorrectly predicted as interacting (false positives, *FP*) and the number of interacting residues incorrectly predicted as noninteracting (false negatives, *FN*). We use four standard measures of performance: sensitivity $Sen = TP/(TP + FN)$, specificity $Spe = TN/(TN + FP)$, accuracy $Acc = (TP + TN)/(TP + FN + TN + FP)$ and positive predictive value $PPV = TP/(TP + FP)$. We also consider scores to evaluate the pertinence of the measures above with respect to expected values. Expected values are calculated on $TP^{\mathrm{exp}} = C \cdot S$, $TN^{\mathrm{exp}} = (1-C)(N-S)$, $FP^{\mathrm{exp}} = C \cdot (N-S)$, $FN^{\mathrm{exp}} = (1-C) \cdot S$, where $C = P/N$ is the coverage of the protein, where P is the number of surface residues predicted, N is the total number of surface residues and S is the number of residues in the real interaction site. Notice that the calculation of expected values assumes that $C \cdot N$ residues have been selected at random as being positives on the structure of the protein under study. This means that expected values are different for different proteins. Then we can compute sensitivity Sen^{exp}, specificity Spe^{exp}, accuracy Acc^{exp} and positive predictive value PPV^{exp} for the random case: C, $1 - C$, $((1 - C) \cdot (1 - S/N)) + C \cdot S/N$, S/N, respectively. Pertinence scores are computed as follows: sensitivity score $ScSen = Sen - Sen^{\mathrm{exp}}$, specificity score $ScSpe = Spe - Spe^{\mathrm{exp}}$, accuracy score $ScAcc = Acc - Acc^{\mathrm{exp}}$ and PPV score $ScPPV = PPV/PPV^{\mathrm{exp}}$.

α, β, γ Parameterization for the Entropy Function Applied to a Single Protein or to a Dataset

The parameters α, β are set so to preserve the convexity of the entropy function within $[0, 1]$ and in such a way that the expected value n^{exp} for a given protein structure or a given database of proteins (defined above), becomes the minimum of the entropy function. Intuitively, while $\alpha \leq 1$ moves the minimum of the entropic function toward $n = 0$, the parameter β allows to start (at $\alpha = 1$) from a minimum which is close enough to n^{exp}.

The parameter β is the same for all sequences of a database. To explain its role, let us consider its intrinsic relation with the parameter α. The equation $\log_2 \beta \cdot \alpha^2 + (2 - \log_2 \beta) \cdot \alpha - 1 = 0$ expresses α in terms of β. There are two solutions α_1, α_2 for the equation and if α_1 falls into the interval $[0, 1]$, then the convexity of the function is guaranteed for $\alpha \in [\alpha_1, 1]$, otherwise it is guaranteed for



Comparison EC vs IC in Homodimers - Huang Database

Comparison EC vs IC in Heterodimers - Huang Database

Comparison EC vs IC in Transients - Huang Database

Fig. 3 Comparative evaluation of the (average of the) predictions of homodimer (*top*), heterodimer (*center*) and transient (*bottom*) interfaces based on the notions of *EC* and *IC*. The evaluation is realized on different coverage levels

$(0, 1]$. By varying α within the interval $[\alpha_1, 1]$, the minimum of the entropic function falls into an interval I of the form $[\frac{e^{-1/\alpha_1}}{\beta}, \frac{e^{-1}}{\beta}]$. If α varies within $(0, 1]$, then the values of the entropic function fall within $I = (0, \frac{e^{-1}}{\beta}]$. By parameterizing β, we want all values n^{exp} associated to the full dataset of sequences (possibly one sequence) to belong to I. To do this, we fix β in such a way that $n \log_2 \beta n$ has the minimum at $\frac{e^{-1}}{\beta}$ (notice that here $\alpha = 1$).

For the Huang database, β takes value 2.1836, the interval of variation for α is $[0.5941, 1]$ and the minima of the entropic functions vary within $I = [0.0851, 0.1684]$. The interval I includes all values n^{exp} computed for the sequences of the Huang database.

To compare entropy values associated to trees of several different sequences, we use the parameter γ to guarantee the y-values of the entropic functions to fall into the same interval. Given α, β, we define γ to be $\frac{\min(n \log_2 \beta n - \log_2 \beta)}{\min(n^\alpha \log_2 \beta n - \log_2 \beta)}$. This way, all y-values of the entropic function fall into the common interval $[\min(n \log_2 \beta n - \log_2 \beta), 0]$.

5 Discussion

A numerical value coding for the information content of a structure is a very valuable quantity, and to correctly interpret this value is key for understanding how to use it. The classical definition of *IC* for a set of aligned amino acid sequences is known to be representing the conservation level of the amino acids in a protein. We show that it is only an “approximation” of this idea and that the conservation level can be described more properly by revisiting the *IC* notion with a new and very simple interpretation of the distance tree associated to multiple sequence alignment. The new notion *EC* provides a better estimation of the conserved interaction sites in a protein. Particularly, it detects especially well whether the complementary region of an interaction site is missing signals of conservation. This property is of particular importance when one wants to couple interaction site detection with docking algorithms. On a large database of protein complexes, we consistently observe that approximating *IC* with *EC* is always profitable.

One can envisage a definition of information content of sets of sequences that includes not only residue position conservation (coded in tree topology) but coevolved residue positions, also coded in tree topology. This aim is far from being a trivial one. Notice that a similar attempt lead to the definition of information content for RNAs [21], where RNA secondary structures provide a way to quantify feasible structures presenting coevolving sites. For amino acid sequences, this task appears much more complicated due to the intrinsic physical-chemical nature of proteins. We should expect the new ranking induced by coevolved residues to be correlated to sparse networks of amino acids associated to functional and mechanical properties of the proteins in the sense of [3, 9, 16].

Some work related to our approach is the study of spaces of sequences evolved for protein folding [15, 22]. Also, an attempt to mix the notion of information content on sequences and the information coming from the tree topology has been proposed in [10]. We demonstrated somewhere else that the definition reported there can be simplified by a better reading of the combinatorial structure of the tree [7]. In contrast to [10], notice that our contribution in this paper is to introduce a new explicit reading of distance trees from which to derive the information content of the pool of sequences.

Acknowledgements Part of this work has been done with the financial support of the AFM/IBM/CNRS Decrypthon project.

References

1. Adami C, Cerf NJ (2000) Physical complexity of symbolic sequences. Physica D 137:62–69
2. Altschul SF, Madden TL, Schaffer AA, Zhang J, Zhang Z, Miller W, Lipman DJ (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res 25(3):389–3402
3. Baussand J (2008) Évolution des séquences protéiques: signatures structurales hydrophobes et réseaux d'acides aminés co-évolués. Thèse de Doctorat de l'Université Pierre et Marie Curie-Paris 6
4. Caffrey DR, Somaroo S, Hughes JH, Mintseris J, Huang ES (2004) Are protein–protein interfaces more conserved in sequence than the rest of the protein surface? Protein Sci 13:190–189
5. Carothers JM, Oestreich SC, Davis JH, Szostak JW (2004) Informational complexity and functional activity of RNA structures. J Am Chem Soc 126:5130–5137
6. Duret L, Abdeddaim S (2000) Multiple alignment for structural functional or phylogenetic analyses of homologous sequences. In: Higgins D, Taylor W (eds) Bioinformatics sequence structure and databanks. Oxford University Press, Oxford
7. Engelen S, Trojan LA, Sacquin-Mora S, Lavery R, Carbone A (2009) JET: detection and analysis of protein interfaces based on evolution. PLOS Comput Biol 5(1):e1000267, 1–17
8. Lecompte O, Thompson JD, Plewniak F, Thierry J, Poch O (2001) Multiple alignment of complete sequences (MACS) in the post-genomic era. Gene 270:17–30
9. Lockless S, Ranganathan R (1999) Evolutionary conserved pathways of energetic connectivity in protein families. Science 286:295–299
10. Mihalek I, Reš I, Lichtarge O (2004) A family of evolution-entropy hybrid methods for ranking protein residues by importance. J Mol Biol 336:1265–1282
11. Moult J (2005) A decade of CASP: progress, bottlenecks and prognosis in protein structure prediction. Curr Opin Struct Biol 15:285–289
12. Notredame C (2002) Recent progresses in multiple sequence alignment: a survey. Pharmacogenomics 31:131–144
13. Notredame C (2007) Recent evolutions of multiple sequence alignment algorithms. PLOS Comput Biol 8:e123
14. Phillips A, Janies D, Wheeler W (2000) Multiple sequence alignment in phylogenetic analysis. Mol Phylogenet Evol 16:317–330
15. Schmidt Am Busch M, Lopes A, Mignon D, Simonson T (2007) Computational protein design: software implementation, parameter optimization, and performance of a simple model. J Comput Chem 29(7):1092–1102
16. Suel G, Lockless S, Wall M, Ranganthan R (2003) Evolutionary conserved networks of residues mediate allosteric communication in proteins. Nat Struct Biol 23:59–69
17. Sugio S, Petsko GA, Manning JM, Soda K, Ringe D (1995) Crystal structure of a D-amino acid aminotransferase: how the protein controls stereoselectivity. Biochemistry 34:9661–9669
18. Thompson JD, Plewniak F, Poch O (1999) A comprehensive comparison of multiple sequence alignment programs. Nucleic Acids Res 27:12682–12690

19. Wallace IM, Blackshields G, Higgins DG (2005) Multiple sequence alignments. Curr Opin Struct Biol 15:261–266
20. Watson JD, Laskowski RA, Thornton JM (2005) Predicting protein function from sequence and structural data. Curr Opin Struct Biol 15:275–284
21. Xayaphoummine A, Viasnoff V, Harlepp S, Isambert H (2007) Encoding folding paths of RNA switches. Nucleic Acids Res 35:614–622
22. Xia Y, Levitt M (2004) Simulating protein evolution in sequence and structure space. Curr Opin Struct Biol 14:202–207

Duplication in DNA Sequences

Masami Ito, Lila Kari, Zachary Kincaid, and Shinnosuke Seki

Abstract The duplication and repeat-deletion operations are the basis of a formal language theoretic model of errors that can occur during DNA replication. During DNA replication, subsequences of a strand of DNA may be copied several times (resulting in duplications) or skipped (resulting in repeat-deletions). As formal language operations, iterated duplication and repeat-deletion of words and languages have been well studied in the literature. However, little is known about single-step duplications and repeat-deletions. In this paper, we investigate several properties of these operations, including closure properties of language families in the Chomsky hierarchy and equations involving these operations. We also make progress toward a characterization of regular languages that are generated by duplicating a regular language.

1 Introduction

Duplication grammars and duplication languages have recently received a great deal of attention in the formal language theory community. Duplication grammars, defined in [16], model duplication using string rewriting systems. Several properties of languages generated by duplication grammars were investigated in [16] and [17]. Another prevalent model for duplication is a unary operation on words [2, 3, 9, 11–13]. The research on duplication is motivated by errors that occur during DNA[1] replication. Duplication and repeat-deletion (also called repeat expansion and repeat contraction, i.e., insertions and deletions of tandem repeating sequence) are biologically significant because they are among the most common errors that occur during DNA replication. In general, insertions and deletions have been linked to cancer

[1] A DNA single strand is a string over the DNA alphabet of bases $\{A, C, G, T\}$. Due to the Watson–Crick complementarity property of bases, wherein A is complement to T and C is complement to G, two DNA single strands of opposite orientation and exact complementary sequences can bind to each other to form a double DNA strand. This process is called base-pairing.

This research was supported by Grant-in-Aid for Scientific Research No. 19-07810 by the Japan Society for the Promotion of Sciences and Research Grant No. 015 by Kyoto Sangyo University to M.I., and The Natural Sciences and Engineering Council of Canada Discovery Grant and Canada Research Chair Award to L.K.

S. Seki (✉)
Department of Computer Science, University of Western Ontario, London, Ontario, N6A 5B7, Canada
e-mail: sseki@csd.uwo.ca

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_4, © Springer-Verlag Berlin Heidelberg 2009

and more than 15 hereditary diseases [1]. They can also have positive consequences such as a contribution to the genetic functional compensation [5]. Interestingly, the mechanisms that cause insertions and deletions are not all well understood by geneticists [4]. For example, the strand slippages at tandem repeats and interspersed repeats are well understood but the repeat expansion and contraction in trinucleotide repeat diseases remain unexplained.

Strand slippage is a prevalent explanation for the occurrence of repeat expansions and repeat contractions during DNA replication. DNA replication is the process by which the DNA polymerase enzyme creates a new "nascent DNA strand" that is the complement of a given single strand of DNA referred to as the "template strand." The replication process begins by mixing together the template DNA strand, the DNA polymerase enzyme, a special short DNA single strand called a "primer," and sufficient individual bases that will be used as building blocks. The primer is specially designed to base-pair with the template, and thus make it double-stranded for the length of the primer. The DNA polymerase will use the primer-template double-strand subsequence as a toe-hold, and will start adding complementary bases to the template strand, one by one, in one direction only, until the entire template strand becomes double-stranded. It has been observed that errors can happen during this process; the most common of them being insertions and deletions of bases. The current explanation is that these repeat expansions and repeat contractions are caused by misalignments between the template and nascent strand during replication [4]. DNA polymerase is not known to have any "memory" to remember which base on the template has been just copied onto the nascent strand, and hence the template and nascent strands can *slip*. As such, the DNA polymerase may copy a part of the template twice (resulting in an insertion) or forget to copy it (deletion). Repeat expansions and contractions occur most frequently on repeated sequences, so they are appropriately modeled by the rewriting rules $u \to uu$ and $uu \to u$, respectively.

The rule $u \to uu$ is a natural model for duplication, and the rule $uu \to u$ models the dual of duplication, which we call *repeat-deletion*. Since strand slippage is responsible for both these operations, it is natural to study both duplication and repeat-deletion. Repeat-deletion has already been extensively studied, e.g., in [10]. However, the existing literature addresses mainly the iterated application of both repeat-deletion and duplication. This paper investigates the effects of a *single* duplication or repeat-deletion. This restriction introduces subtle new complexities into languages that can be obtained as a duplication or repeat-deletion of a language.

This paper is organized as follows: in Sect. 2, we define terminology and notations to be used throughout the paper. Section 3 is dedicated to the closure properties of the language families of the Chomsky hierarchy under duplication and repeat-deletion. In Sect. 4, we present and solve language equations based on these operations, and give constructive solutions of the equation in the case involving duplication operation and regular languages. In Sect. 5, we introduce a generalization of duplication, namely controlled duplication. Section 6 investigates a characterization of the regular languages that can be obtained as a duplication of a regular language. When complete, such a characterization would constructively solve the language equation involving repeat-deletion and regular languages, for a certain class of languages. Lastly, in Sect. 7, we present some results on the relationship between duplication, repeat-deletion, and primitive words.

The conference version of this paper was published in [8].

2 Preliminaries

We now provide definitions for terms and notations to be used throughout the paper. For basic concepts in formal language theory, we refer the reader to [6, 7, 20, 22]. For a relation R, we denote by R^* the reflexive, transitive closure of R. Σ denotes a finite alphabet, Σ^* denotes the set of words over Σ, and Σ^+ denotes the set of words over Σ excluding the empty word λ. For a nonnegative integer $n \geq 0$, Σ^n denotes the set of words of length n over Σ, and let $\Sigma^{\leq n} = \bigcup_{i=0}^{n} \Sigma^i$. The length of a word $w \in \Sigma^*$ is denoted by $|w|$. A language over Σ is a subset of Σ^*. For a language $L \subseteq \Sigma^*$, the set of all (internal) factors (resp. prefixes, suffixes) of L are denoted by $\mathrm{F}(L)$ (resp. $\mathrm{Pref}(L)$, $\mathrm{Suff}(L)$). The complement of a language $L \subseteq \Sigma^*$, denoted by L^c, is defined as $L^c = \Sigma^* \setminus L$. We denote by FIN the family of all finite languages, by REG the family of all regular languages, by CFL the family of all context-free languages, and by CSL the family of all context-sensitive languages. We note that FIN $\subsetneq$ REG $\subsetneq$ CFL $\subsetneq$ CSL.

For a finite automaton $A = (Q, \Sigma, \delta, s, F)$ (where Q is a state set, Σ is an alphabet, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $s \in Q$ is the start state, and $F \subseteq Q$ is a set of final states), let $\mathcal{L}(A)$ denote the language accepted by A. We extend δ to $\hat{\delta} : Q \times \Sigma^* \to 2^Q$ as follows: (1) $\hat{\delta}(q, \lambda) = \{q\}$ for $q \in Q$ and (2) $\hat{\delta}(q, wa) = \bigcup_{p \in \hat{\delta}(q,w)} \delta(p, a)$ for $q \in Q$, $w \in \Sigma^*$, and $a \in \Sigma$. For $P_1, P_2 \subseteq Q$, we define an automaton $A_{(P_1,P_2)} = (Q \cup s_0, \Sigma, \delta', s_0, P_2)$, where $s_0 \notin Q$ is a new start state and $\delta' = \delta \cup (s_0, \lambda, P_1)$. Thus,

$$\mathcal{L}(A_{(P_1,P_2)}) = \left\{w \mid \hat{\delta}(p_1, w) \cap P_2 \neq \emptyset \text{ for some } p_1 \in P_1\right\}.$$

If P_i is the singleton set $\{p_i\}$, then we may simply write p_i for $i \in \{1, 2\}$.

In this paper, we investigate two operations that are defined on words and extended to languages: *duplication* and *repeat-deletion*. We employ the duplication operation $\heartsuit$ described in [2], which is defined as follows:

$$u^{\heartsuit} = \left\{v \mid u = xyz,\ v = xyyz \text{ for some } x, z \in \Sigma^*, y \in \Sigma^+\right\}.$$

In the canonical way, the duplication operation is extended to a language $L \subseteq \Sigma^*$:

$$L^{\heartsuit} = \bigcup_{u \in L} u^{\heartsuit}.$$

We also define another unary operation based on the dual of the $\heartsuit$ operation. We term this operation *repeat-deletion* and denote it by $\spadesuit$. Note that while biologists refer to this process simply as deletion, in formal language theory, the term deletion typically refers to removing arbitrary (rather than repeated) factors of word.

Definition 1 For a word $v \in \Sigma^*$, the language generated by repeat-deletion of v is defined

$$v^{\spadesuit} = \{u \mid v = xyyz,\ u = xyz \text{ for some } x, z \in \Sigma^*, y \in \Sigma^+\}.$$

Again, the repeat-deletion operation is extended to languages: for a given language $L \subseteq \Sigma^*$,

$$L^{\spadesuit} = \bigcup_{v \in L} v^{\spadesuit}.$$

We avoid inverse notation because $\heartsuit$ and $\spadesuit$ are not inverses when considered as operations on languages. That is, for a language $L \subseteq \Sigma^*$, $L \subseteq (L^{\heartsuit})^{\spadesuit}$ but it is not always the case that $L = (L^{\heartsuit})^{\spadesuit}$.

Example 1 Let $L = a^*bb$. Then $abb \in L \Rightarrow aabb \in L^{\heartsuit}$. Therefore, $aab \in (L^{\heartsuit})^{\spadesuit}$, but $aab \notin L$.

Previous work focused on the reflexive transitive closure of the duplication operation, which we will refer to as duplication closure. All occurrences of $\heartsuit$, duplication, $\spadesuit$, and repeat-deletion refer to the *single step* variations of the operations.

3 Closure Properties

Much of the work on duplication has been concerned with determining which of the families of languages on the Chomsky hierarchy are closed under duplication closure. It is known that on a binary alphabet, the family of regular languages is closed under duplication closure. In contrast, on a larger alphabet, REG is still closed under n-bounded duplication closure for $n \leq 2$, but REG is not closed under n-bounded operation closure for any $n \geq 4$. The family of context-free languages is closed under (uniformly) bounded duplication closure. The readers are referred to [9] for these results.

It is a natural first step to determine these closure properties under (single step) duplication. In this section, we show that the family of regular languages is closed under repeat-deletion but not duplication, the family of context-free languages is not closed under either operation, and the family of context-sensitive languages is closed under both operations.

The following two propositions are due to [21] (without proofs).

Proposition 1 *The family of regular languages is not closed under duplication.*

Proof Let $L = ab^*$ and suppose that $L^{\heartsuit}$ is regular. Since the family of regular languages is closed under intersection, $L' = L^{\heartsuit} \cap ab^*ab^*$ is regular. But L' is exactly the language $\{ab^iab^j : i \leq j\}$, which is clearly not regular. So, by contradiction, $L^{\heartsuit}$ is not regular, and the family of regular languages is not closed under duplication. □

Note that the proof of the preceding proposition requires that the alphabet contain at least two letters. As we shall see in Sect. 6, this bound is tight: the family of regular languages over a unary alphabet is closed under duplication.

Proposition 2 *The family of context-free languages is not closed under duplication.*

Proof Let $L = \{a^i b^i \mid i \geq 1\}$, a context-free language. Suppose $L^{\heartsuit}$ is context-free. Since the family of context-free languages is closed under intersection with regular languages, $D = L^{\heartsuit} \cap a^*b^*a^*b^*$ is context free.

Let p be the pumping-lemma constant of the language D. Consider the word $z = a^p b^p a^p b^p \in D$. We can decompose z as $z = uvwxy$ such that vx is a pumped part. Let $z_i = uv^i wx^i y$. Firstly, v must not contain both a and b; otherwise, pumping v results in a word with more than two repetitions of $a^i b^j$ for some $i, j \geq 1$. This also applies to x. Secondly, vx must be within the central $b^p a^p$ part; otherwise, the pumped vx causes a difference between the number of first as and the number of last bs. Now, we know that vwx is within the central $b^p a^p$ part of z, and $v = b^i$ and $x = a^j$ for some $0 \leq i, j \leq p$ (with i, j not both zero). Then $z_2 = a^p b^{p+i} a^{p+j} b^p$, which cannot be generated by duplication of a word in L. Thus, we conclude that $L^{\heartsuit}$ is not context-free. □

Proposition 3 *The family of context-sensitive languages is closed under duplication.*

Proof Let L be a context-sensitive language, and A_L be a linear-bounded automaton for L. Now, we construct a Turing machine $A_{\heartsuit}$ for $L^{\heartsuit}$ and show that $A_{\heartsuit}$ is a linear-bounded automaton. Indeed, for a given input $w \in \Sigma^*$, $A_{\heartsuit}$ nondeterministically choose $w' \in \mathrm{F}(w)$ (let $w = xw'z$ for some $x, z \in \Sigma^*$) and checks whether $w' = yy$ for some $y \in \Sigma^*$. If not, it turns down this choice. Otherwise, it deletes one of y so that the input tape has xyz. Now, $A_{\heartsuit}$ simulates A_L on this tape, and if A_L accepts the given input, xyz, then $A_{\heartsuit}$ accepts $w = xyyz$. Therefore, $A_{\heartsuit}$ accepts w if and only if there exists a nondeterministic choice of the infix with respect to which the simulated A_L accepts the given input. Thus, $\mathcal{L}(A_{\heartsuit}) = L^{\heartsuit}$.

This construction has four steps; the choice of an infix of an input, check of whether the infix is repetitive, deletion, and the simulation of A_L. The first three steps require the workspace linear-proportional to the length of an input. In the fourth step, A_L receives an input which is shorter than the original input to $A_{\heartsuit}$ and A_L is a linear-bounded automaton. As a result, $A_{\heartsuit}$ is also a linear-bounded automaton. □

In the following, we consider the closure properties of the language families in the Chomsky hierarchy under repeat-deletion. Our first goal is to prove that the family of regular languages is closed under repeat-deletion. For this purpose, we define the following binary operation $\natural$ on languages $L, R \subseteq \Sigma^*$:

$$L \natural R = \{xyz \mid xy \in L,\ yz \in R,\ y \neq \lambda\}.$$

Proposition 4 (Due to Z. Ésik) *The family of regular languages is closed under* $\natural$.

Proof Let $L_1, L_2 \subseteq \Sigma^+$ be regular languages. Let # be a new letter (not in Σ) and let h be homomorphism defined by $h(a) = a$ for $a \in \Sigma^*$ and $h(\#) = \lambda$. Let $L'_1 = L_1 \leftarrow \{\#\} = \{u\#v \mid uv \in L_1\}$ ($\leftarrow$ denotes the insertion operation) and $L'_2 = L_2 \leftarrow \{\#\}$. Moreover, let $\overline{L_1} = L'_1\#\Sigma^*$ and let $\overline{L_2} = \Sigma^*\#L'_2$. Then $L_1 \natural L_2 = h(\overline{L_1} \cap \overline{L_2})$. Since the family of regular languages is closed under insertion, concatenation, intersection, and homomorphism, $L_1 \natural L_2$ is regular. □

Let L be a regular language. We can construct a finite automaton $A = (Q, \Sigma, \delta, s, F)$ such that $\mathcal{L}(A) = L$. Recall that for any state $q \in Q$, $\mathcal{L}(A_{(s,q)}) = \{w : sw \vdash_A^* q\}$ and $\mathcal{L}(A_{(q,F)}) = \{w : \exists f \in F \text{ such that } qw \vdash_A^* f\}$. Intuitively, $\mathcal{L}(A_{(s,q)})$ is the set of words accepted "up to q," and $\mathcal{L}(A_{(q,F)})$ is the set of words accepted "after q" so that $\mathcal{L}(A_{(s,q)})\mathcal{L}(A_{(q,F)}) \subseteq L$ is the set of words in L that have a derivation that passes through state q.

Lemma 1 *Let L be a regular language and $A = (Q, \Sigma, \delta, s, F)$ be a finite automaton accepting L. Then $L^{\spadesuit} = \bigcup_{q \in Q} \mathcal{L}(A_{(s,q)}) \natural \mathcal{L}(A_{(q,F)})$.*

Proof Let $L' = \bigcup_{q \in Q} \mathcal{L}(A_{(s,q)}) \natural \mathcal{L}(A_{(q,F)})$. First, we prove that $L^{\spadesuit} \subseteq L'$. Let $\alpha \in L^{\spadesuit}$. Then there exists a decomposition $\alpha = xyz$ for some $x, y, z \in \Sigma^*$ such that $xyyz \in L$ and $y \neq \lambda$. Since A accepts $xyyz$, there exists some $q \in Q$ such that $sxyyz \vdash^* qyz$ and $qyz \vdash^* f$ for some $f \in F$. By construction, $xy \in \mathcal{L}(A_{(s,q)})$ and $yz \in \mathcal{L}(A_{(q,F)})$. This implies that $xyz \in \mathcal{L}(A_{(s,q)}) \natural \mathcal{L}(A_{(q,F)})$, from which we have $L^{\spadesuit} \subseteq L'$.

Conversely, if $\alpha \in L'$, then there exists $q \in Q$ such that $\alpha \in \mathcal{L}(A_{(s,q)}) \natural \mathcal{L}(A_{(q,F)})$. We can decompose α into xyz for some $x, y, z \in \Sigma^*$ such that $xy \in \mathcal{L}(A_{(s,q)})$, $yz \in \mathcal{L}(A_{(q,F)})$, and $y \neq \lambda$. Since $\mathcal{L}(A_{(s,q)})\mathcal{L}(A_{(q,F)}) \subseteq L$, we have that $xyyz$ belongs to L. It follows that $\alpha = xyz \in L^{\spadesuit}$ and $L' \subseteq L^{\spadesuit}$. We conclude that $L' = L^{\spadesuit}$. □

Proposition 5 *The family of regular languages is closed under repeat-deletion.*

Proof Since the family of regular languages is closed under finite union and the $\natural$ operation, it is closed under repeat-deletion (due to Lemma 1). □

Proposition 6 *The family of context-free languages is closed under* $\natural$ *with regular languages.*

Proof Repeat the argument in the proof for Proposition 4. Since the family of context-free languages is closed under insertion, concatenation with regular languages, intersection with regular languages, and homomorphism, the family of context-free languages is closed under $\natural$ with regular languages. □

Lemma 2 *The family of context-free languages is not closed under* $\natural$.

Proof Let $L_1 = \{a^i \# b^i \$ \mid i \geq 0\}$ and $L_2 = \{\# b^j \$ c^j \mid j \geq 0\}$. Although L_1 and L_2 are CFLs, $L_1 \natural L_2 = \{a^i \# b^i \$ c^i \mid i \geq 0\}$, which is not context-free. □

Proposition 7 *The family of context-free languages is not closed under repeat-deletion.*

Proof Let $L = \{a^i \# b^i \# b^j c^j \mid i, j \geq 0\}$, which is context-free. Then $L^{\spadesuit} \cap a^* \# b^* c^* = \{a^i \# b^j c^j \mid i, j \geq 0, i \leq j\}$, which is not context-free. Since the family of context-free languages is closed under intersection with regular languages, and since $L^{\spadesuit} \cap a^* \# b^* c^*$ is not context-free, we may conclude that $L^{\spadesuit}$ is not context-free. Thus, the family of context-free languages is not closed under repeat-deletion. □

However, there do exist context-free (and nonregular) languages whose image under repeat-deletion remains context-free. An example is shown below.

Example 2 Let $L = \{a^n b^n \mid n \geq 0\}$; this is a context-free language. Then $L^{\spadesuit} = \{a^n b^m \mid 1 \leq m < n \leq 2m\} \cup \{a^n b^m \mid 1 \leq n < m \leq 2n\}$. This $L^{\spadesuit}$ is generated by the following context-free grammar, and hence in CFL. Let $G = (\{a, b\}, \{S, X, Y, X_f, Y_f\}, P, S)$, where the set of production rules P is given by

$$\begin{aligned} S &\to X \mid Y, \\ X &\to aXb \mid aaX_f b, \\ Y &\to aYb \mid aY_f bb, \\ X_f &\to aX_f b \mid aaX_f b \mid \lambda, \\ Y_f &\to aY_f b \mid aY_f bb \mid \lambda. \end{aligned}$$

Proposition 8 *The family of context-sensitive languages is closed under repeat-deletion.*

Proof Let L and A_L be defined as we did in Proposition 3. As $A_{\heartsuit}$ in the proposition, we construct a linear-bounded automaton $A_{\spadesuit}$ for $L^{\spadesuit}$ which simulates A_L. In contrast to $A_{\heartsuit}$, $A_{\spadesuit}$ nondeterministically copies an infix of a given input w. Formally speaking, w is regarded as a catenation of x, y, z, and y is duplicated so as to result in $xyyz$ on the input tape. Then $A_{\spadesuit}$ runs A_L on the tape. If A_L accepts $xyyz$, then $A_{\spadesuit}$ accepts $w = xyz$. As shown in Proposition 3, $A_{\spadesuit}$ is a linear-bounded automaton. □

In summary, the following closure properties related to duplication, repeat-deletion, and the $\natural$ operation hold:

	$\heartsuit$	$\spadesuit$	$\natural$	$\natural$ with regular
FIN	Y	Y	Y	N
REG	N	Y	Y	Y
CFL	N	N	N	Y
CSL	Y	Y	Y	Y

4 Language Equations

We now consider the language equation problem posed by the duplication operation: for a given language $L \subseteq \Sigma^*$, can we find a language $X \subseteq \Sigma^*$ such that $X^\heartsuit = L$? In the following, we show that if L is a regular language and there exists a solution to $X^\heartsuit = L$, then we can compute a maximal solution. We note that the solution to the language equation is not unique in general.

Example 3 $\{aaa, aaaa, aaaaa\}^\heartsuit = \{aaa, aaaaa\}^\heartsuit = \{a^i : 4 \le i \le 10\}$.

In view of the fact that a language equation may have multiple solutions, we define an equivalence relation $\sim_\heartsuit$ on languages as follows:

$$X \sim_\heartsuit Y \Leftrightarrow X^\heartsuit = Y^\heartsuit.$$

For the same reason, we define an equivalence relation $\sim_\spadesuit$ as follows:

$$X \sim_\spadesuit Y \Leftrightarrow X^\spadesuit = Y^\spadesuit.$$

Lemma 3 *If* $[X] \in 2^{\Sigma^*}/\sim_\heartsuit$ *and if* $\Xi \subseteq [X]$ $(\Xi \neq \emptyset)$, *then* $\bigcup_{L \in \Xi} L \in [X]$.

Proof Let $[X] \in 2^{\Sigma^*}/\sim_\heartsuit$ and $\Xi \subseteq [X]$ $(\Xi \neq \emptyset)$. Prove that $L_\Xi = \bigcup_{L \in \Xi} L \in [X]$.

Let $Y \in \Xi$. Clearly, $Y \subseteq L_\Xi$ and so $Y^\heartsuit \subseteq {L_\Xi}^\heartsuit$. Now let $w \in {L_\Xi}^\heartsuit$. Then $\exists x, z \in \Sigma^*, y \in \Sigma^+, v \in L_\Xi$ such that $w = xyyz$ and $v = xyz$. Then there exists $Z \in \Xi$ such that $v \in Z$. Since $Y, Z \in \Xi$, $v^\heartsuit \subseteq Z^\heartsuit = Y^\heartsuit$. Then $w \in v^\heartsuit$ implies $w \in Y^\heartsuit$. Thus, ${L_\Xi}^\heartsuit \subseteq Y^\heartsuit$. We conclude that $Y^\heartsuit = {L_\Xi}^\heartsuit$ and $L_\Xi \in [X]$. □

Corollary 1 *For an equivalence class* $[X] \in 2^{\Sigma^*}/\sim_\heartsuit$, *there exists a unique maximal element* $X_{\max}$ *with respect to the set inclusion partial order defined as follows*:

$$X_{\max} = \bigcup_{L \in [X]} L.$$

We provide a way to construct the maximum element of a given equivalence class. First, we prove a more general result.

Proposition 9 *Let* $L \subseteq \Sigma^*$, *and let* $f, g : \Sigma^* \to 2^{\Sigma^*}$ *be any functions such that* $u \in g(v) \Leftrightarrow v \in f(u)$ *for all* $u, v \in \Sigma^*$. *If a solution to the language equation* $\bigcup_{x \in X} f(x) = L$ *exists, then the maximum solution* (*with respect to the set inclusion partial order*) *is given by* $X_{\max} = (\bigcup_{y \in L^c} g(y))^c$.

Proof For two languages $X, Y \subseteq \Sigma^*$ such that $\bigcup_{x \in X} f(x) = L$ and $\bigcup_{y \in Y} f(y) = L$, $\bigcup_{z \in X \cup Y} f(z) = L$ holds. Hence, the assumption implies the existence of $X_{\max}$.

($\subseteq$) Suppose $\exists w \in g(v) \cap X_{\max}$ for some $v \in L^c$. This means that $v \in f(w)$. However, $f(w) \subseteq \bigcup_{x \in X_{\max}} f(x) = L$, and hence $v \in L$, a contradiction. ($\supseteq$) Suppose that $\exists w \in X_{\max}^c \cap (\bigcup_{y \in L^c} g(y))^c$. If $f(w) \subseteq L$, then $w \in X_{\max}$ (by the maximality of $X_{\max}$). Otherwise, $\exists v \in f(w) \cap L^c$. This implies that $w \in g(v) \subseteq$

$\bigcup_{y \in L^c} g(y)$. In both cases, we have a contradiction. Therefore, we have $X^c_{\max} = \bigcup_{y \in L^c} g(y)$, i.e., $X_{\max} = (\bigcup_{y \in L^c} g(y))^c$. □

Lemma 4 *Let $u, v \in \Sigma^*$. Then $u \in v^{\heartsuit}$ if and only if $v \in u^{\spadesuit}$.*

Proof ($\Rightarrow$) If $u \in v^{\heartsuit}$, then there exist $x, z \in \Sigma^*$ and $y \in \Sigma^+$ such that $v = xyz$ and $u = xyyz$. Then $u^{\spadesuit}$ contains $xyz = v$. ($\Leftarrow$) If $v \in u^{\spadesuit}$, then there exist $x', z' \in \Sigma^*$ and $y' \in \Sigma^+$ such that $v = x'y'z'$ and $u = x'y'y'z'$. Then $x'y'y'z' = u \in v^{\heartsuit}$. □

Proposition 9 and Lemma 4 imply the following corollaries.

Corollary 2 *Let $L \subseteq \Sigma^*$. If there exists a language $X \subseteq \Sigma^*$ such that $X^{\spadesuit} = L$, then the maximum element $X_{\max}$ of $[X]_{\sim_{\spadesuit}}$ is given by $((L^c)^{\heartsuit})^c$.*

Corollary 3 *Let $L \subseteq \Sigma^*$. If there exists a language $X \subseteq \Sigma^*$ such that $X^{\heartsuit} = L$, then the maximum element $X_{\max}$ of $[X]_{\sim_{\heartsuit}}$ is given by $((L^c)^{\spadesuit})^c$.*

Proposition 10 *Let L, X be regular languages satisfying $X^{\heartsuit} = L$. Then it is decidable whether X is the maximal solution for this language equation.*

Proof Since L is regular and REG is closed under repeat-deletion and complement, the maximum solution of $X^{\heartsuit} = L$ given in Corollary 3, $((L^c)^{\spadesuit})^c$, is regular. Since the equivalence problem for regular languages is decidable, it is decidable whether a given solution to the duplication language equation is maximal. □

Due to the fact that REG is not closed under duplication, we cannot obtain a similar decidability result for the $X^{\spadesuit} = L$ language equation. This motivates our investigation in the next two sections of necessary and sufficient conditions for the duplication of a regular language to be regular. Indeed, in the cases when the duplication language $(L^c)^{\heartsuit}$ is regular, the solution to language equations $X^{\spadesuit} = L$, $L \in$ REG, can be constructed as described in Corollary 2.

5 Controlled Duplication

In Sect. 4, we showed that for a given language $L \subseteq \Sigma^*$, the maximal solution of the repeat-deletion language equation $X^{\spadesuit} = L$ is given by $((L^c)^{\heartsuit})^c$. However, unlike the duplication language equation, we do not have an efficient algorithm to compute this language due to the fact that the family of regular languages is not closed under duplication. This motivates "controlling" the duplication in such a manner that duplications can occur only for some specific words.

Let L, C be languages over Σ. We define the duplication of L using the control set C as follows:

$$L^{\heartsuit(C)} = \{xyyz \mid xyz \in L,\ y \in C\}.$$

Note that this generalization of the duplication operation can express two variants of duplication that appear in previous literature, namely uniform and length-bounded duplication ([12, 13]). Indeed, using the notation in [13], we have

$$D^1_{\{n\}}(L) = L^{\heartsuit(\Sigma^n)} \quad \text{and} \quad D^1_{\{0,1,\ldots,n\}}(L) = L^{\heartsuit(\Sigma^{\leq n})}.$$

This section presents basic properties of controlled duplications, some of which will turn out to be useful in Sect. 6. For symmetry, we also investigate properties of controlled repeat-deletion.

Lemma 5 *Let $L \subseteq \Sigma^*$ be a language and $C_1, C_2 \subseteq \Sigma^*$ be control sets. If $C_1 \subseteq C_2$, then $L^{\heartsuit(C_1)} \subseteq L^{\heartsuit(C_2)}$.*

Lemma 6 *Let $L \subseteq \Sigma^*$ be a language and $C_1, C_2 \subseteq \Sigma^*$ be control sets. Then $L^{\heartsuit(C_1 \cup C_2)} = L^{\heartsuit(C_1)} \cup L^{\heartsuit(C_2)}$.*

Let $L \subseteq \Sigma^*$ be a language, $C \subseteq \Sigma^*$ be a control set, and $w \in C$. Then w is said to be *useful with respect to L* if $w \in \mathrm{F}(L)$; otherwise, it is called *useless* with respect to L. The control set C is said to *contain an infinite number of useful words with respect to L* if and only if $|\mathrm{F}(L) \cap C| = \infty$.

Lemma 7 *Let $L \subseteq \Sigma^*$ be a language, $C \subseteq \Sigma^*$ be a control set, and C' be the set of all useless words in C with respect to L. Then $L^{\heartsuit(C)} = L^{\heartsuit(C \setminus C')}$.*

Proof Lemma 6 implies $L^{\heartsuit(C)} = L^{\heartsuit(C \setminus C')} \cup L^{\heartsuit(C')}$. Since $L^{\heartsuit(C')} = \emptyset$, $L^{\heartsuit(C)} = L^{\heartsuit(C \setminus C')}$ □

Proposition 11 *For a regular language $L \subseteq \Sigma^*$ and a regular control set $C \subseteq \Sigma^*$, it is decidable whether C contains an infinite number of useful words with respect to L.*

Proof Since L and C are regular, $\mathrm{F}(L)$, and hence $\mathrm{F}(L) \cap C$ are also regular. Since finiteness of a regular language is decidable, it is decidable whether or not a regular control set C contains an infinite number of useful words with respect to a language L. □

Note that if $L \subseteq \Sigma^*$, $C \subseteq \Sigma^*$ is a control set, and C contains at most a finite number of useful words with respect to L, then $C' = C \cap \mathrm{F}(L)$ is a finite language and satisfies $L^{\heartsuit(C)} = L^{\heartsuit(C')}$. In particular, for any finite language L and any control set C, there exists a finite control set $C' \subseteq C$ satisfying $L^{\heartsuit(C)} = L^{\heartsuit(C')}$.

We now extend our previous results on the closure properties of language families so as to accommodate the controlled duplication. Since $\heartsuit = \heartsuit_{\Sigma^*}$, we trivially have the following:

- The family of regular languages is not closed under controlled duplication.

- The family of context-free languages is not closed under controlled duplication, repeat-deletion, or $\natural$.

We conclude this section with definitions of repeat-deletion and the $\natural$ operation using control sets, and by providing a few results of them.

Let $L, L_1, L_2, C \subseteq \Sigma^*$. Then

$$L^{\spadesuit(C)} = \{xyz \mid xyyz \in L,\ y \in C\},$$

$$L_1 \natural_C L_2 = \{xyz \mid xy \in L_1,\ yz \in L_2,\ y \in C\}.$$

It is straightforward to prove that the family of regular languages is closed under $\natural_C$ for any regular language C. Let L_1, L_2 be regular languages and form $\overline{L_1}$ and $\overline{L_2}$ as defined in the proof of Proposition 4. We see that $L_1 \natural_C L_2 = h(\overline{L_1} \cap \overline{L_2} \cap \Sigma^* \# C \# \Sigma^*)$. Furthermore, by repeating the argument in the proof of Proposition 5, we have that the family of regular languages is closed under $\spadesuit_C$ for any regular control set C.

It is simple to check that if each word in L contains a subword that is in C, $\heartsuit_C$ and $\spadesuit_C$ satisfy the requirements of Proposition 9, so that we have a procedure to find X such that $X^{\heartsuit(C)} = L$ if such an X exists.

Proposition 12 *Let $L \subseteq \Sigma^*$ be a context-free language and let $C \subseteq \Sigma^+$ be a finite control set. Then $L^{\spadesuit(C)}$ is context-free.*

Proof Let h be the homomorphism defined by $h(a) = h(\overline{a}) = a$ for $a \in \Sigma, \overline{a} \in \overline{\Sigma}$. Then $L' = h^{-1}(L)$ is context-free. Consider $L'' = L' \cap (\Sigma^* \{u\overline{u} \mid u \in C\} \Sigma^*)$. Then L'' is context-free. Now, let θ be the homomorphism defined by $\theta(a) = a$ and $\theta(\overline{a}) = \lambda$ for $a \in \Sigma$. Then $\theta(L'') = L^{\spadesuit(C)}$, and hence $L^{\spadesuit(C)}$ is context-free. □

6 Conditions for $L^{\heartsuit(C)}$ to Be Regular

For a regular language L and a control set C, we now investigate a necessary and sufficient condition for $L^{\heartsuit(C)}$ to be regular. As suggested in the following example, even for a "simple" language L and a control set C, $L^{\heartsuit(C)}$ can be nonregular.

Example 4 Let $\Sigma = \{a, b\}$ and $L = \{w \in \Sigma^* \mid |w| = 0 \pmod 3\}$ and $C = \Sigma^*$. Then $L^{\heartsuit(C)} \notin \mathrm{REG}$.

Given a regular language L, a sufficient condition for $L^{\heartsuit(C)}$ to be regular is a corollary of the following result in [3]. A family of languages is called a *trio* if it is closed under λ-free homomorphism, inverse homomorphism, and intersection with regular languages. Note that both the families of regular languages and of context-free languages are trio.

Theorem 1 ([3]) *Any trio is closed under duplication with a finite control set.*

Corollary 4 *Let $L \subseteq \Sigma^*$ be a regular language and $C \subseteq \Sigma^*$. If there exists a finite control set $C' \subseteq \Sigma^*$ such that $L^{\heartsuit(C)} = L^{\heartsuit(C')}$, then $L^{\heartsuit(C)}$ is regular.*

Given a regular language L, we now investigate necessary conditions for $L^{\heartsuit(C)}$ to be regular. Results in [19] stating that infinite repetitive languages cannot be even context-free indicate that the converse of Corollary 4 may also be true. Hence, in the remainder of this section, we shall investigate the following claim:

Claim *Let $L \subseteq \Sigma^*$ be a regular language and $C \subseteq \Sigma^*$ be a control set. If $L^{\heartsuit(C)}$ is regular, then there exist a finite control set $C' \subseteq \Sigma^*$ such that $L^{\heartsuit(C)} = L^{\heartsuit(C')}$.*

As shown in the following example, this claim generally does not hold.

Example 5 Let $\Sigma = \{a, b\}$, $L = ba^+b$, and $C = ba^+ \cup a^+b$. We can duplicate a prefix ba^i of a word $ba^jb \in L$ $(i \le j)$ to obtain a word $ba^iba^jb \in L^{\heartsuit(C)}$. In the same way, the duplication of a suffix $a^\ell b$ of a word ba^kb $(k \ge \ell)$ results in a word $ba^kba^\ell b \in L^{\heartsuit(C)}$. Thus, $L^{\heartsuit(C)} = ba^+ba^+b$. Note that L and $L^{\heartsuit(C)}$ are regular. However, there exists no finite control set C' satisfying $L^{\heartsuit(C)} = L^{\heartsuit(C')}$. This is because ba^+ba^+b can have arbitrary long repetitions of a's, and hence arbitrary long control factors are required to generate it.

Nevertheless, this claim holds for several interesting cases: the case where L is finite or C contains at most a finite number of useful words with respect to L, the case of a unary alphabet $\Sigma = \{a\}$, the case $L = \Sigma^*$, and the case where the control set is "marked," i.e., there exists $a \in \Sigma$ such that $C \subseteq a(\Sigma \setminus \{a\})^*a$. Moreover, it turned out that the proof technique we employ for this fourth case can be utilized to prove that the claim holds for the case where C is nonoverlapping and an infix code, which is more general than the fourth case. In the following, we prove the direct implication of the claim for these cases (the reverse one is clear from Corollary 4).

In the case where L is finite, $L^{\heartsuit(C)}$ is finite, and hence regular. Since $\mathrm{F}(L)$ is finite, by letting $C' = C \cap \mathrm{F}(L)$, we have $L^{\heartsuit(C)} = L^{\heartsuit(C')}$. Thus, the claim holds for this case. Moreover, even for an infinite L, we can say that if C contains at most a finite number of useful words with respect to L, then the claim holds because C', defined in the same manner as above, is finite. Therefore, in the following, we assume that L is infinite and C contains an infinite number of useful words with respect to L.

Next, we show that the claim holds in the case of a unary alphabet. We employ the following known result for this purpose.

Proposition 13 ([6]) *Let $\Sigma = \{a\}$ be a unary alphabet, and L be a language over Σ. L is regular if and only if there exists a finite set $\mathcal{N}$ of pairs of integers such that $L = \bigcup_{k \ge 0, (n,m) \in \mathcal{N}} a^{kn+m}$.*

Proposition 14 *Let Σ be a unary alphabet, say $\Sigma = \{a\}$, $L \subseteq \Sigma^*$ be a regular language, and $C \subseteq \Sigma^*$ be an arbitrary language. Then $L^{\heartsuit(C)}$ is regular, and there exists a finite context $C' \in$ FIN such that $L^{\heartsuit(C)} = L^{\heartsuit(C')}$.*

Proof L being regular, there exists a finite set of pairs of integers $\mathcal{N} = \{(p_i, q_i) \mid p_i, q_i \in \mathbb{N}_0, 1 \leq i \leq n\}$ for some $n \in \mathbb{N}$ such that $L = \bigcup_{x \geq 0, (p_i, q_i) \in \mathcal{N}} a^{p_i x + q_i}$.

Let $L_i = \bigcup_{x \geq 0} a^{p_i x + q_i}$, and consider a word $a^k \in C$, where $k \in \mathbb{N}$. For some $x \geq 0$, we can apply the duplication with respect to a^k to $a^{p_i x + q_i}$ if and only if $p_i x + q_i \geq k$. The application generates $a^{p_i x + q_i + k} \in L^{\heartsuit(C)}$. Note that for $x_1, x_2 \in \mathbb{N}_0$, $p_i x_1 + q_i + k = p_i x_2 + q_i + k \pmod{p_i}$. We define a function $\psi_i : C \mapsto \{0, 1, \ldots, p_i - 1\}$ such that for $a^k \in C$, $\psi_i(a^k) = q_i + k \pmod{p_i}$. Hence, we can partition C into p_i disjoint sets depending on ψ_i. Formally speaking, $C = \bigcup_{0 \leq m < p_i} C_{i,m}$, where $C_{i,m} = \{w \in C \mid \psi_i(w) = m\}$. Now, the necessary and sufficient condition mentioned above as to the applicability implies that for $a^j, a^k \in C_{i,m}$, if $j \leq k$, then $L_i^{\heartsuit(\{a^j\})} \supseteq L_i^{\heartsuit(\{a^k\})}$. Let $w_{i,m}$ be the shortest word in $C_{i,m}$. Then $L_i^{\heartsuit(\{w_{i,m}\})} = L_i^{\heartsuit(C_{i,m})}$ holds. Thus, by letting $C' = \{w_{i,m} \mid 1 \leq i \leq n, 0 \leq m < p_i\}$, we have $L^{\heartsuit(C)} = L^{\heartsuit(C')}$. Clearly, C' is finite, and hence $L^{\heartsuit(C')}$ is regular. □

By letting $C = \Sigma^*$, Proposition 14 implies that the family of regular languages is closed under duplication when Σ is unary.

Next, we show that the claim holds for the case when $L = \Sigma^*$ (Corollary 5). This requires the following known two lemmata. A word $w \in \Sigma^+$ is said to be *primitive* if $w = v^n$ implies that $n = 1$, i.e., $w = v$. A word $v \in \Sigma^+$ is called a *conjugate* of w if $v = xy$ and $w = yx$ for some $x, y \in \Sigma^*$.

Lemma 8 ([14]) *For a primitive word p, any conjugate of p is primitive.*

Lemma 9 ([15]) *Let p and q be primitive words with $p \neq q$ and let $i, j \geq 2$. Then $p^i q^j$ is primitive.*

For a language $C \subseteq \Sigma^*$, we define $\mathrm{Dup}(C) = \{ww \mid w \in C\}$.

Proposition 15 *Let $C \subseteq \Sigma^*$. Then $\Sigma^*\mathrm{Dup}(C)\Sigma^*$ is regular if and only if there exists a finite language C' such that $\Sigma^*\mathrm{Dup}(C')\Sigma^* = \Sigma^*\mathrm{Dup}(C)\Sigma^*$.*

Proof The proof of "if"-part is obvious since $\Sigma^*\mathrm{Dup}(C')\Sigma^*$ is regular. Now, consider the proof of "only if"-part. Assume $L = \Sigma^*\mathrm{Dup}(C)\Sigma^*$ is regular and consider the regular language $L \cap (\Sigma^* \setminus L\Sigma^+) \cap (\Sigma^* \setminus \Sigma^+ L)$. All words in this language have a representation ww for some $w \in C$. Hence, there exists $C' \subseteq C$ such that $\mathrm{Dup}(C') = L \cap (\Sigma^* \setminus L\Sigma^+) \cap (\Sigma^* \setminus \Sigma^+ L)$. Notice that for any $w \in C$ there exist $w' \in C'$ and $x, y \in \Sigma^*$ such that $ww = xw'w'y$. Therefore, $\Sigma^*\mathrm{Dup}(C)\Sigma^* = \Sigma^*\mathrm{Dup}(C')\Sigma^*$.

Suppose C' is infinite. Then there exists a word $uu \in \mathrm{Dup}(C')$ with length twice that of the pumping lemma constant for $\mathrm{Dup}(C')$. So, by the pumping lemma, there exists a decomposition $uu = u_1u_2u_3u_1u_2u_3$, of uu such that $u_1, u_3 \in \Sigma^*$, $u_2 \in \Sigma^+$ and $u_1u_2^iu_3u_1u_2u_3 \in \mathrm{Dup}(C')$ for any $i \in \mathbb{N}$. Notice that for any $i \in \mathbb{N}$, $u_1u_2^iu_3u_1u_2u_3$ is not primitive because it is in $\mathrm{Dup}(C')$. Consider the case $i \geq 3$. By Lemma 8, $u_2^{i-1}(u_2u_3u_1)^2$ is not primitive. Then Lemma 9 implies that u_2

and $u_2u_3u_1$ share a primitive root, say $p \in \Sigma^+$. We may now write $u_2 = p^n$ and $u_2u_3u_1 = p^m$ for some $n, m \geq 1$. Hence, $u_2^{i-1}(u_2u_3u_1)^2 = p^{n(i-1)+2m}$. From Lemma 8, it follows that $u_1u_2^iu_3u_1u_2u_3 = q^{n(i-1)+2m}$, where q is a conjugate word of p. Now, we have that $u_1u_2^iu_3u_1u_2u_3 = q^{n(i-1)+2m}$ is a proper prefix (and suffix) of $u_1u_2^{i+1}u_3u_1u_2u_3 = q^{ni+2m}$, which contradicts with the definition of $\mathrm{Dup}(C')$. Thus, C' must be finite. □

Lemma 10 *Let $C \subseteq \Sigma^*$. Then $(\Sigma^*)^{\heartsuit(C)} = \Sigma^*\mathrm{Dup}(C)\Sigma^*$.*

Proof Let $w \in (\Sigma^*)^{\heartsuit(C)}$. Then there exist $x, y, z \in \Sigma^*$ such that $y \in C$ and $w = xyyz$. Thus, $w \in \Sigma^*\mathrm{Dup}(C)\Sigma^*$. Conversely, let $v \in \Sigma^*\mathrm{Dup}(C)\Sigma^*$. Then v is of the form $xyyz$ such that $x, z \in \Sigma^*$ and $yy \in \mathrm{Dup}(C)$ (i.e., $y \in C$). The duplication of y in $xyz \in \Sigma^*$ results in $xyyz = v$, and hence $v \in (\Sigma^*)^{\heartsuit(C)}$. □

The following corollary derives from Lemma 10 and Proposition 15. In fact, this corollary asserts the claim in the case when $L = \Sigma^*$.

Corollary 5 *Let $C \subseteq \Sigma^*$. Then $(\Sigma^*)^{\heartsuit(C)}$ is regular if and only if there exists a finite subset $C' \subseteq C$ such that $(\Sigma^*)^{\heartsuit(C')} = (\Sigma^*)^{\heartsuit(C)}$.*

The last case we consider is that of marked duplication, where given a word w in $L^{\heartsuit(C)}$, we can deduce or at least guess the factor whose duplication generates w from a word in L, according to some mark of a control set C. Here, we consider a mark which shows the beginning and end of a word in C, that is, $C \subseteq \#(\Sigma \setminus \{\#\})^*\#$ for some character #. For a strongly-marked duplication, where $\# \notin \Sigma$ and $L \subseteq \Sigma^*\#\Sigma^*\#\Sigma^*$, we can easily show that the existence of a finite control set provided $L^{\heartsuit(C)}$ is regular, using the pumping lemma for the regular language. Hence, we consider the case when the mark itself is a character in Σ, say $\# = a$ for some $a \in \Sigma$.

It turned out that we could employ the proof of the claim in the case of the marked duplication for the more general case when C is a nonoverlapping and an infix code. A language L is called *nonoverlapping* if $vx, yv \in L$ implies $x = y = \lambda$, and L is called *infix-code* if $L \cap (\Sigma^*L\Sigma^+ \cup \Sigma^+L\Sigma^*) = \emptyset$. That is, any elements of the language which is nonoverlapping and an infix-code do not overlap each other. In the following, we prove the claim for this case.

We introduce several notions and notations used in the proof. For a word $w \in L^{\heartsuit(C)}$, we call a tuple (x, y, z) a *dup-factorization of w with respect to L and C* if $w = xyyz$, $xyz \in L$, and $y \in C$. When L and C are clear from the context, we simply say that (x, y, z) is a dup-factorization of w. Let $\delta(w)$ be the number of dup-factorizations of w with respect to L and C. For $y \in C$, if there are $x, z \in \Sigma^*$ such that (x, y, z) is a dup-factorization of w, then we call y a *dup-factor* of w. Let $F_d(w)$ be the set of all dup-factors of w. Note that $|F_d(w)| \leq \delta(w)$ but the inequality may be strict.

Proposition 16 *Let L be a regular language and C be a control set which is nonoverlapping and an infix-code. Then the regularity of $L^{\heartsuit(C)}$ implies the existence of a finite control set C' such that $L^{\heartsuit(C)} = L^{\heartsuit(C')}$.*

Proof Let $\equiv_L$ and $\equiv_\heartsuit$ be the syntactic congruences of L and $L^{\heartsuit(C)}$, respectively, and we define $\equiv = \equiv_L \cap \equiv_\heartsuit$. Since both L and $L^{\heartsuit(C)}$ are regular, $C/\equiv$ is finite. Let $\Gamma_2 = \{[c] \in C/\equiv \text{ s.t. } |[c]| \leq 2\}$. Using induction on the number of dup-factorizations, we prove that (i) $\Gamma_2 \neq \emptyset$, and (ii) any word in $L^{\heartsuit(C)}$ has a dup-factor which is in an equivalence class in Γ_2.

Firstly, we consider a word w in $L^{\heartsuit(C)}$ which has the smallest number of dup-factorizations among the elements of $L^{\heartsuit(C)}$. Suppose that no dup-factor of w is in equivalence classes in Γ_2. Let (x, y, z) be a dup-factorization of w. Then there exists $y' \in C$ such that $y' \equiv y$, $y' \neq y$, and $y' \notin \mathrm{Suff}(x)$. Let $w' = xy'yz$. This is in $L^{\heartsuit(C)}$, and hence w' must have a dup-factorization, say (α, β, γ) for some $\alpha, \beta, \gamma \in \Sigma^*$. Due to the nonoverlapping and infix-code properties of C, β^2 is an infix of either x or yz. Here, we assume that it is in x, and let $x = \alpha\beta^2 v$, $\gamma = vy'yz$ for some $v \in \Sigma^*$. Then

$$\begin{aligned} w' = \alpha\beta^2\gamma \in L^{\heartsuit(C)} &\Rightarrow \alpha\beta v y' y z \in L \\ &\Rightarrow \alpha\beta v y y z \in L \\ &\Rightarrow \alpha\beta^2 v y y z = w \in L^{\heartsuit(C)}. \end{aligned}$$

Thus, $(\alpha, \beta, vyyz)$ is a dup-factorization of w. Generally speaking, for a dup-factorization (α, β, γ) of w', w has a corresponding dup-factorization (α', β, γ) if y' is an infix of α, or (α, β, γ') otherwise (i.e., y' is an infix of γ). Indeed, this means that $\delta(w') < \delta(w)$ and $F_d(w') \subseteq F_d(w)$. The first consequence is a contradiction while the second one is of importance in the induction step. The second is clear from the above discussion. In order to show the first, it is enough to prove that there do not exist two distinct dup-factorizations of w' which correspond to the same dup-factorization of w, and there exists no dup-factorization of w' which corresponds to (x, y, z).

Let $(\alpha_1, \beta_1, \gamma_1)$ and $(\alpha_2, \beta_2, \gamma_2)$ be two distinct dup-factorizations of w', and consider dup-factorizations of w which correspond to them respectively (either $(\alpha_i', \beta_i, \gamma_i)$ or $(\alpha_i, \beta_i, \gamma_i')$ for each $i = 1, 2$). Firstly, we prove that $(\alpha_1, \beta_1, \gamma_1') \neq (\alpha_2, \beta_2, \gamma_2')$. Suppose not, then since $w' = \alpha_1\beta_1^2\gamma_1 = \alpha_2\beta_2^2\gamma_2$, we have $\gamma_1 = \gamma_2$, a contradiction. Next, we compare $(\alpha_1, \beta_1, \gamma_1')$ and $(\alpha_2', \beta_2, \gamma_2)$ (see Fig. 1). Their construction shown above implies that γ_1 and α_2 must contain y' as their infix. Hence, $|\alpha_1\beta_1^2| + |y'| \leq |\alpha_2|$. Since α_2' is generated by replacing y' in α_2 with y and $\beta \neq \lambda$, we have $|\alpha_1| < |\alpha_2|$. Thus, $(\alpha_1, \beta_1, \gamma_1') \neq (\alpha_2', \beta_2, \gamma_2)$. Using the same way, we can easily check that $(\alpha_i', \beta_i, \gamma_i), (\alpha_i, \beta_i, \gamma_i') \neq (x, y, z)$.

Now, we assume that for all words in $L^{\heartsuit(C)}$ which have at most n dup-factorizations have a dup-factor which is in the equivalence class in Γ_2. Suppose that there were $v \in L^{\heartsuit(C)}$ with $n + 1$ dup-factorizations and without any dup-factor which is in the equivalence class of size at most 2. Then we can construct a word

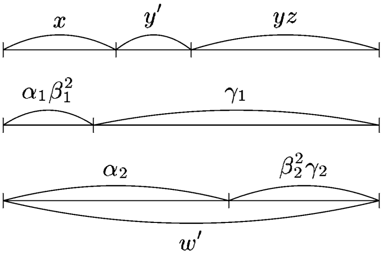


Fig. 1 The comparison between two dup-factorizations, $(\alpha_1, \beta_1, \gamma_1)$ and $(\alpha_2, \beta_2, \gamma_2)$, of w'

v' as above which satisfies $\delta(v') \le n$ and $F_d(v') \subseteq F_d(v)$, which contradict with the induction assumption. □

Corollary 6 *Let L be a regular language and C be a control set. If there exists a finite set $C_1 \subset C$ such that $C \setminus C_1$ is nonoverlapping and an infix-code, then the regularity of $L^{\heartsuit(C)}$ implies the existence of a finite control set C' such that $L^{\heartsuit(C)} = L^{\heartsuit(C')}$.*

Proof Note that $L^{\heartsuit(C)} = L^{\heartsuit(C_1)} \cup L^{\heartsuit(C \setminus C_1)}$. Proposition 16 implies the existence of a finite control set C_2 such that $L^{\heartsuit(C \setminus C_1)} = L^{\heartsuit(C_2)}$. Then by letting $C' = C_1 \cup C_2$, which is finite, we have $L^{\heartsuit(C)} = L^{\heartsuit(C')}$. □

Indeed, we can prove that $\Gamma_1 = \{[c] \in C/\equiv \text{ s.t. } |[c]| = 1\}$ is enough to generate $L^{\heartsuit(C)}$, that is, for a finite control set $C' = \{c \mid [c] \in \Gamma_1\}$, $L^{\heartsuit(C)} = L^{\heartsuit(C')}$.

Proposition 17 *Let L be a regular language and $C \subseteq \Sigma^*$ be a nonoverlapping and an infix code. If $L^{\heartsuit(C)}$ is regular, then $L^{\heartsuit(C)} = L^{\heartsuit(C')}$, where $C' = \{c \mid [c] \in \Gamma_1\}$.*

Proof All we have to prove is that for $w \in L^{\heartsuit(C)}$, unless w has a dup-factor which is in C', there exists $w' \in L^{\heartsuit(C)}$ such that $\delta(w') < \delta(w)$ and $F_d(w') \subseteq F_d(w)$.

Let (x, y, z) be a dup-factorization of w, and let $y' \in C$ such that $y \neq y'$ but $y \equiv y'$. Then let $w_0 = xy'yz$, which is in $L^{\heartsuit(C)}$. The proof of Proposition 16 implies that if either (1) $y' \notin \text{Suff}(x)$ or (2) $x = x_1y'$ for some $x_1 \in \Sigma^*$ but (x_1, y', yz) is not a dup-factorization of w_0, then $\delta(w_0) < \delta(w)$. Even otherwise ($w_0 = x_1y'y'yz$), $\delta(w_0) \le \delta(w)$. If this holds with equality, consider $w_1 = x_1yy'yz \in L^{\heartsuit(C)}$. If either (1) $y \notin \text{Suff}(x_1)$ or (2) $x_1 = x_2y$ for some $x_2 \in \Sigma^*$ but $(x_2, y, y'yz)$ is not a dup-factorization of w_1, then $\delta(w_1) < \delta(w_0) = \delta(w)$. Otherwise, let $w_2 = x_2y'yy'yz$. Note that x_k is getting strictly shorter. Hence, repeating this process, we eventually reach an integer $i \ge 0$ such that either (1) or (2) holds for w_i. We can check that $\delta(w_i) < \delta(w_{i-1}) \le \cdots \le \delta(w_0) \le \delta(w)$ and $F_d(w_i) \subseteq F_d(w)$ as follows: Let $w_i = x_i(y'y)^{i/2+1}z \in L^{\heartsuit(C)}$ (for even i; the odd case is essentially same, and hence omitted). Let $w_i = \alpha\beta^2\gamma$, where (α, β, γ) is a dup-factorization of w_i. Since either (1) or (2) holds, β^2 is an infix of x_i or that of yz. Assume the former and let $x_i = \alpha\beta^2\gamma'$ and $\gamma = \gamma'(y'y)^{i/2+1}z$. Then $\alpha\beta\gamma'(y'y)^{i/2+1}z \in L$. Using $y \equiv y'$, we can say that $\alpha\beta\gamma'(yy')^{i/2}yyz \in L$, and hence $\alpha\beta^2\gamma'(yy')^{i/2}yyz \in L^{\heartsuit(C)}$. The left-hand side is $x_i(yy')^{i/2}yyz = x_{i-1}y'(yy')^{i/2-1}yyz = \cdots = xyyz = w$. □

Consequently, we can say that if we let $m = |C/\equiv|$, then the size of finite control set C' is at most $m - 1$ because at least one equivalence class in $C/\equiv$ must have infinite cardinality.

7 Duplication and Primitivity

Recall that a word $w \in \Sigma^*$ is primitive if there exists no $u \in \Sigma^*$ such that $w = u^k$ for some $k \geq 2$. We denote by Q the set of all primitive words over the alphabet Σ. There is evidently a connection between duplication, repeat-deletion, and primitive words, but the nature of this relationship is unclear. The following section elucidates some of the properties of this relationship.

Proposition 18 (See, for instance, [18]) *Let $u, v \in \Sigma^+$ such that uv is primitive. Then both $u(uv)^n$ and $v(uv)^n$ are primitive for any $n \geq 2$.*

Proposition 19 *Let $w \in \Sigma^*$ be a nonprimitive word. If we duplicate an infix of w which is strictly shorter than the primitive root of w, then the resulting word is primitive.*

Proof Let $w = f^n$ for $f \in Q$ and $n \geq 2$. We also denote $w = xyz$ for $x, y, z \in \Sigma^*$, where y is the infix we duplicate so that the resulting word is $xyyz$. Since $w = f^n = xyz$, there exist $f_s \in \text{Suff}(f)$ and $f'_p \in \text{Pref}(f)$ satisfying $y = f_s f'_p$. Then yzx, a conjugate of xyz, is written as $yzx = (f_s f_p)^n$, where $f_p \in \text{Pref}(f)$ satisfying $f = f_p f_s$. Let $g = f_s f_p$. Clearly $g \in Q$. Now, we prove that $yyzx$ is primitive, and hence $xyyz$ is also primitive.

We have $yyzx = f_s f'_p yzx = f_s f'_p g^n$. Since $|y| < |f|$, there exists a word $v \in \Sigma^+$ such that $f_p = f'_p v$. Then $yyzx = y(yv)^n$ and Proposition 18 implies that $yyzx$ is primitive. □

Proposition 20 *Let $x, y, z \in \Sigma^*$. If xyz is primitive and $xyyz$ is not primitive, then xz is primitive.*

Proof Let f be the primitive root of y, i.e., $y = f^m$ for some $m \geq 1$. Since $xyyz \notin Q$, its conjugate $zxyy$ is also not primitive. Suppose zx were not primitive, i.e., $zx = g^n$ for some $n \geq 2$ and $g \in Q$. If $g \neq f$, then $zxyy = g^n f^{2m}$. Lemma 9 implies that $zxyy \in Q$, a contradiction. If $g = f$, then $y = g^m$, and hence $zxy = g^{n+m} \notin Q$. Thus, $xyz \notin Q$, a contradiction. As a result, $zx \in Q$, that is, $xz \in Q$. □

8 Discussion

In this paper, we studied duplication and repeat-deletion, two formal language theoretic models of insertion and deletion errors occurring during DNA replication.

Specifically, we obtained the closure properties of the families of languages in the Chomsky hierarchy under these operations, the language equations of the form $X^{\heartsuit} = L$ and $X^{\spadesuit} = L$ for a given language L, and the operation of controlled duplication. In addition, we made steps toward finding a necessary and sufficient condition for a controlled duplication of a regular language to be regular.

Two problems for further investigation are: the problem of how to decide for a given language L whether the language equation $X^{\heartsuit} = L$ has a solution, and the problem of finding a necessary condition for the controlled duplication of a regular language to be regular, in the general case.

Acknowledgements We wish to express our gratitude to Dr. Zoltán Ésik for the concise proof of Proposition 4. We would also like to thank Dr. Helmut Jürgensen for our discussion on the claim and Dr. Kathleen Hill for extended discussions on the biological motivation for duplication and repeat-deletion.

References

1. Bichara M, Wagner J, Lambert IB (2006) Mechanisms of tandem repeat instability in bacteria. Mut Res 598(1–2):144–163
2. Dassow J, Mitrana V, Păun Gh (1999) On the regularity of duplication closure. Bull EATCS 69:133–136
3. Dassow J, Mitrana V, Salomaa A (2002) Operations and language generating devices suggested by the genome evolution. Theor Comput Sci 270:701–738
4. Garcia-Diaz M, Kunkel TA (2006) Mechanism of a genetic glissando: structural biology of indel mutations. Trends Biochem Sci 31(4):206–214
5. Gu Z, Steinmetz LM, Gu X, Scharfe G, Davis RW, Li W-H (2003) Role of duplicate genes in genetic robustness against null mutations. Nature 421:63–66
6. Harrison MA (1978) Introduction to formal language theory. Addison–Wesley, Reading
7. Ito M (2004) Algebraic theory of automata and languages. World Scientific, Singapore
8. Ito M, Kari L, Kincaid Z, Seki S (2008) Duplication in DNA sequences. In: Ito M, Toyama M (eds) DLT 2008. Lecture notes in computer science, vol 5257. Springer, Berlin, pp 419–430
9. Ito M, Leupold P, S-Tsuji K (2006) Closure of language classes under bounded duplication. In: Ibarra OH, Dang Z (eds) DLT 2006. Lecture notes in computer science, vol 4036. Springer, Berlin, pp 238–247
10. Leupold P (2007) Duplication roots. In: Harju T, Karhumäki J, Lepistö A (eds) DLT 2007. Lecture notes in computer science, vol 4588. Springer, Berlin, pp 290–299
11. Leupold P (2006) Languages generated by iterated idempotencies and the special case of duplication. PhD thesis, Department de Filologies Romaniques, Facultat de Lletres, Universitat Rovira i Virgili, Tarragona, Spain
12. Leupold P, M-Vide C, Mitrana V (2005) Uniformly bounded duplication languages. Discrete Appl Math 146(3):301–310
13. Leupold P, Mitrana V, Sempere J (2004) Formal languages arising from gene repeated duplication. In: Aspects of molecular computing. Essays in honour of Tom Head on his 70th birthday. Lecture notes in computer science, vol 2950. Springer, Berlin, pp 297–308
14. Lothaire M (1983) Combinatorics on words. Encyclopedia of mathematics and its applications, vol 17. Addison–Wesley, Reading
15. Lyndon RC, Schützenberger MP (1962) On the equation $a^M = b^N c^P$ in a free group. Mich Math J 9:289–298
16. M-Vide C, Păun Gh (1999) Duplication grammars. Acta Cybern 14:151–164

17. Mitrana V, Rozenberg G (1999) Some properties of duplication grammars. Acta Cybern 14:165–177
18. Reis CM, Shyr HJ (1978) Some properties of disjunctive languages on a free monoid. Inf Control 37:334–344
19. Ross R, Winklmann K (1982) Repetitive strings are not context-free. RAIRO Inform Theor 16(3):191–199
20. Rozenberg G, Salomaa A (eds) (1997) Handbook of formal languages. Springer, Berlin
21. Searls DB (1993) The computational linguistics of biological sequences. In: Hunter L (ed) Artificial intelligence and molecular biology. AAAI Press/MIT Press, Menlo Park, pp 47–120
22. Yu SS (2005) Languages and codes. Lecture notes. Department of Computer Science, National Chung-Hsing University, Taichung, Taiwan 402

Sequence and Structural Analyses for Functional Non-coding RNAs

Yasubumi Sakakibara and Kengo Sato

Abstract Analysis and detection of functional RNAs are currently important topics in both molecular biology and bioinformatics research. Several computational methods based on stochastic context-free grammars (SCFGs) have been developed for modeling and analysing functional RNA sequences. These grammatical methods have succeeded in modeling typical secondary structures of RNAs and are used for structural alignments of RNA sequences. Such stochastic models, however, are not sufficient to discriminate member sequences of an RNA family from non-members, and hence to detect non-coding RNA regions from genome sequences. Recently, the support vector machine (SVM) and kernel function techniques have been actively studied and proposed as a solution to various problems in bioinformatics. SVMs are trained from positive and negative samples and have strong, accurate discrimination abilities, and hence are more appropriate for the discrimination tasks. A few kernel functions that extend the string kernel to measure the similarity of two RNA sequences from the viewpoint of secondary structures have been proposed. In this article, we give an overview of recent progress in SCFG-based methods for RNA sequence analysis and novel kernel functions tailored to measure the similarity of two RNA sequences and developed for use with support vector machines (SVM) in discriminating members of an RNA family from non-members.

1 Introduction

Since the number of known RNA sequences, structures, and families is growing rapidly, computational methods for finding non-protein-coding RNA regions in the genome have garnered much attention and research [6]. Compared with the gene finding difficulties inherent in protein-coding regions, identifying non-coding RNA regions computationally is essentially more problematic as these sequences do not have strong statistical signals. Currently, a general finding algorithm does not exist.

In RNA sequence analysis, the specific form of the secondary structures in the cell is an important feature for modeling and detecting RNA sequences. The folding of an RNA sequence into a functional molecule is largely governed by the formation of the standard Watson–Crick base pairs A-U and C-G as well as the wobble pair G-U.

Y. Sakakibara (✉)
Department of Biosciences and Informatics, Keio University, 3-14-1 Hiyoshi,
Kohoku-ku, Yokohama 223-8522, Japan
e-mail: yasu@bio.keio.ac.jp

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_5, © Springer-Verlag Berlin Heidelberg 2009

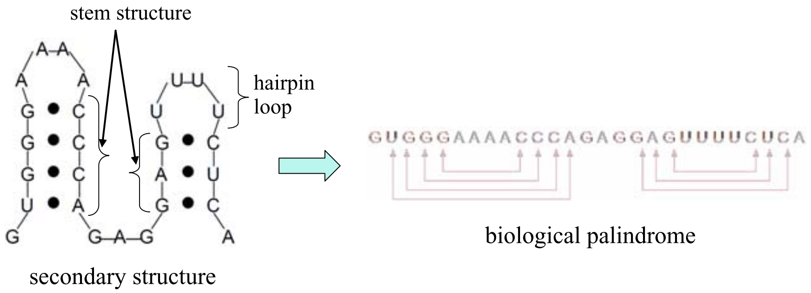


Fig. 1 A typical secondary structure including stem structure of an RNA sequence (*left*) constitutes the so-called biological palindrome (*right*)

Such base pairs constitute "biological palindromes" in the genome (see Fig. 1). The secondary structures of RNAs are generally composed of *stems*, *hairpins*, *bulges*, *interior loops*, and *multi-branches*. A stem is a double-stranded (paired) region of base-pair stacks (see Fig. 1). A hairpin loop occurs when RNA folds back on itself.

To capture such secondary structure features, stochastic context-free grammars (SCFGs) for RNAs have been proposed. SCFGs have been used successfully to model typical secondary structures of RNAs and are also used for structural alignment of RNA sequences [1, 4, 5, 20, 25, 27, 29]. However, a serious drawback of the SCFG method is the requirement of prior knowledge. A known typical secondary structure of the target RNA family is needed to design the grammars.

Furthermore, stochastic models such as SCFGs and hidden Markov models (HMMs) have limitations in discriminating member sequences of an RNA family from non-members by only examining the probabilistic scores. Hence, we require stronger discriminative methods to detect and find non-coding RNA sequences.

Recently, the support vector machine (SVM) and kernel function techniques have been actively studied and used to propose solutions to various problems in bioinformatics [17, 30, 31]. SVMs are trained from positive and negative samples and have strong, accurate discrimination abilities. They are, therefore, better suited to the discrimination tasks. For protein sequence analyses, string kernels [31] have been proposed for use with SVMs to classify a protein family. In addition, string kernels are proven to work for remote homology detections of protein sequences, i.e. a superfamily.

Several kernel functions have been proposed which enhance the ability to measure the similarity of two RNA sequences from the viewpoint of secondary structure.

2 RNA Sequence Alignment and Secondary Structure Prediction Using Stochastic Grammar

In RNA, the nucleotides adenine (A), cytosine (C), guanine (G), and uracil (U) interact in specific ways to form characteristic secondary-structure motifs such as helices, loops, and bulges. In general, the folding of an RNA chain into a functional molecule is largely governed by the formation of intramolecular A-U and G-C Watson–Crick pairs. Such base pairs constitute the so-called biological palindromes in the genome.

2.1 SCFG: Stochastic Context-Free Grammar

Eddy and Durbin [5] and Sakakibara et al. [27] have shown that base pairing in RNA can be described by a context-free grammar (CFG). In particular, productions of the forms $X \rightarrow \mathtt{A}\ Y\ \mathtt{U}$, $X \rightarrow \mathtt{U}\ Y\ \mathtt{A}$, $X \rightarrow \mathtt{G}\ Y\ \mathtt{C}$, and $X \rightarrow \mathtt{C}\ Y\ \mathtt{G}$ describe the structure in RNA due to Watson–Crick base pairing. Using productions of these types, a CFG can specify the language of biological palindromes. For example, the application of productions in the grammar shown in Fig. 2 could generate the RNA sequence CAUCAGGGAAGAUCUCUUG and the derivation can be arranged in a tree structure called a *derivation tree* (Fig. 3, left). A derivation tree represents the syntactic structure of a sequence produced by a grammar. For the RNA sequence, this syntactic structure corresponds to the physical secondary structure (Fig. 3, right).

A *stochastic context-free grammar* (SCFG) G consists of a set of non-terminal symbols N, a terminal alphabet Σ, a set P of production rules with associated probabilities, and the start symbol S. The associated probability for every production $A \rightarrow \alpha$ in P is denoted $\Pr(A \rightarrow \alpha)$, and a probability distribution exists over the set of productions which have the same non-terminal on the left-hand sides.

Sakakibara et al. [27] have extended the notion of profile HMMs to Profile SCFGs so that profile SCFGs can represent motifs, calculate multiple alignments, and predict secondary structures of RNA sequences. Sakakibara et al. [27] have assessed the ability of the trained SCFGs to perform three tasks: to discriminate transfer RNA (tRNA) sequences from non-tRNA sequences; to produce multiple alignments; and to ascertain a secondary structure of new sequences. The results have shown that after having been trained on tRNA sequences, the trained grammar can identify general tRNA from similar-length RNA sequences of other kinds; can find secondary structure of new tRNA sequences (as shown in Table 1); and can produce multiple alignments of large sets of tRNA sequences.

$$
\begin{aligned}
G_{rna} &= (N, \Sigma, P, S) \\
N &= \{S, X_1, \ldots, X_{16}\} \\
\Sigma &= \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\} \\
P &= \left\{ \begin{array}{lll}
S \rightarrow \mathtt{A}X_1\mathtt{U}, & & \\
X_1 \rightarrow \mathtt{G}X_2\mathtt{C}, & X_2 \rightarrow X_3X_4, & X_3 \rightarrow \mathtt{A}X_5\mathtt{U}, \\
X_5 \rightarrow \mathtt{A}X_6\mathtt{U}, & X_6 \rightarrow X_7X_8, & X_7 \rightarrow \mathtt{A}X_9\mathtt{U}, \\
X_9 \rightarrow \mathtt{G}X_{10}\mathtt{C}, & X_{10} \rightarrow \mathtt{A}X_{11}, & X_{11} \rightarrow \mathtt{UG}, \\
X_8 \rightarrow \mathtt{G}X_{12}\mathtt{C}, & X_{12} \rightarrow \mathtt{A}X_{13}\mathtt{U}, & X_{13} \rightarrow \mathtt{A}X_{14}, \\
X_{14} \rightarrow \mathtt{GC}, & X_4 \rightarrow \mathtt{G}X_{15}\mathtt{C}, & X_{15} \rightarrow \mathtt{C}X_{16}\mathtt{G}, \\
X_{16} \rightarrow \mathtt{UG} & &
\end{array} \right\}
\end{aligned}
$$

Fig. 2 This set of productions P generates RNA sequences with a certain restricted structure. $S, X_1, \ldots, X_{16}$ are non-terminals; A, U, G, and C are terminals representing the four nucleotides

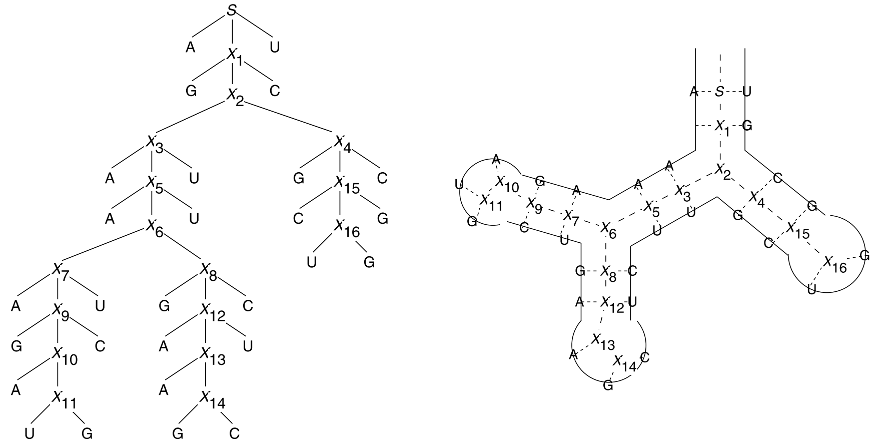


Fig. 3 A derivation tree (*left*) generated by a simple CFG for RNA molecules and the physical secondary structure (*right*) of the RNA sequence which is a reflection of the derivation tree

Table 1 Prediction results of secondary structures for unfolded tRNA sequences. Percentage of correct base-pairs predicted by the structural alignment algorithm based on PHMMTSs (upper), by Clustal-W using multiple alignment (middle), and by the fully trained profile SCFG (lower)

tRNA type	ARCHAE	CY	CYACHL	EUBACT	VIRUS	MT	PART III
PHMMTS	95.47%	99.71%	97.59%	98.65%	100.00%	85.38%	50.22%
Clustal-W	90.06%	94.29%	98.13%	94.12%	65.08%	57.78%	13.64%
Profile SCFG	100.00%	99.87%	99.79%	99.86%	100.00%	98.93%	83.00%

2.2 PHMMTS: Pair Hidden Markov Model on Tree Structure

Recently, Sakakibara [25] has proposed Pair HMMs on tree structures (PHMMTSs), which is an extension of PHMMs defined on alignments of trees and which provides a unifying framework and an automata-theoretic model for alignments of trees, structural alignments, and Pair SCFGs. Based on the observation that a secondary structure of RNA can be represented by a tree, PHMMTSs are applied to the problem of structural alignments of RNAs. By structural alignment, we mean a pairwise alignment to align an unfolded RNA sequence into an RNA sequence of known secondary structure, as illustrated in Fig. 4. The PHMMTS is modified so that it takes as input a pair, consisting of a linear sequence and a tree representing a secondary structure of RNA, and produces a structural alignment.

Sakakibara [25] has presented some computational experiments on RNA families, which show the effectiveness of PHMMTS methods for structural alignment. In the experiments, one RNA sequence annotated with the known secondary structure is randomly chosen from the group EUBACT in the database, and then all other "unfolded" tRNA sequences are structurally aligned into the "folded" tRNA sequence. The fraction of base pairs specified by the trusted alignment that matched in the

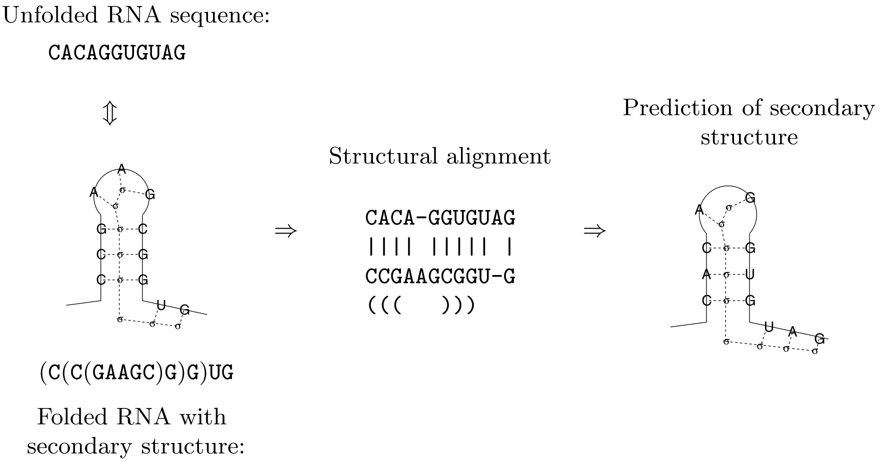


Fig. 4 Structural pairwise alignment problem of RNA sequences

predictions of the PHMMTS-based structural alignment algorithm is counted. The prediction results of the PHMMTS-based algorithm was also compared with the predictions based on multiple alignments by Clustal-W [33] and the profile SCFG-method. Clustal-W was given as input, a set of unfolded tRNA sequences in each group together with the target tRNA sequence. Secondary structures for each set of tRNA sequences were predicted based on a multiple alignment made by Clustal-W. Clustal-W does not take into account the secondary structure of the target tRNA sequence when it makes alignments, but has an advantage of taking multiple alignment instead of pairwise alignment. The results are shown in Table 1. The predicted secondary structures of the PHMMTS-based structural alignment algorithm agree extremely well with the trusted alignment, and this algorithm outperforms Clustal-W, especially for the groups VIRUS, MT, and PART III. The group PART III is most difficult in the sense that it contains 58 tRNA sequences of unusual secondary structures lacking a whole loop of the D-domain. For this group, the PHMMTS-based prediction accuracy is still 50% while Clustal-W predicts very poorly. From these comparisons, it is very clear that structural alignment is essential to align RNA sequences and predict the secondary structures.

On the other hand, the Profile SCFG, which was fully trained and designed specifically for the tRNA family, always predicts best. It is important to note that the PHMMTS method does not require any training process and predicts secondary structures based only on structural alignment into one single folded RNA sequence. In this sense, these experiments illustrate the trade-off between computational and resource costs and prediction accuracy.

2.3 RNA Secondary Structural Alignment with Conditional Random Fields

Besides pairwise alignments on amino acid sequences, one of the most important factors in RNA secondary structural alignments is substitution score matrices of nucleotides. These matrices should include scores for both mutation on a single base and covariation on a base pair. The previous implementation of PHMMTSs cannot calculate sufficient structural alignments due to an ad-hoc substitution score matrix.

To improve substitution score matrices for non-coding RNAs, Sato et al. [29] have proposed RNA secondary structural alignment based on conditional random fields (CRFs). CRFs proposed by Lafferty et al. [16] have several advantages over traditional HMMs and stochastic grammars because CRFs can build discriminative models with flexible state transitions from annotated training data. Our approach has some specific features compared to previous methods in the sense that the parameters for structural alignment are estimated such that the model can most probably discriminate between correct alignments and incorrect alignments, and has the generalization ability so that a satisfiable score matrix can be obtained even with a small number of sample data without overfitting.

Experimental results show that the parameter estimation with CRFs can outperform all the other scoring methods for structural alignments of RNA sequences such as RIBOSUM [14], which is calculated by an analogous method to the BLOSUM matrices based on the maximum likelihood estimation by relative frequencies. Furthermore, structural alignment search based on CRFs is more accurate for predicting non-coding RNA regions than the other scoring methods. These experimental results strongly support our discriminative method employing CRFs to estimate the score matrix parameters.

Recently, Do et al. [3] have proposed a method to predict RNA secondary structures using conditional log-linear models, almost identical to CRFs, and have shown that the proposed method outperforms all the other methods such as structure predictions based on the minimum free energy [38].

2.4 PSTAG: Pair Stochastic Tree Adjoining Grammars for Aligning and Predicting Pseudoknot RNA Structures

Pseudoknot secondary structures of non-coding RNA molecules play important roles for their own functions such as catalytic functions (see Fig. 5). From a computational point of view, pseudoknots are difficult to handle because modeling the pseudoknot structures of RNAs is beyond the generative power of context-free grammars, and inevitably involves the hard complexity of context-sensitivity. Uemura et al. [35] have applied tree adjoining grammars (TAG) to represent pseudoknot structures of RNA sequences. Rivas and Eddy [23] and Cai et al. [2] have

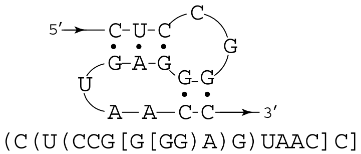


Fig. 5 Pseudoknot secondary structure

Table 2 The accuracy of predicting base pairs for HDV_ribozyme by PSTAG, PHMMTS, and Clustal-W

	Specificity (%)	Sensitivity (%)
PSTAG	88.9	96.0
PHMMTS	46.4	52.0
Clustal-W	25.9	28.0

also attempted grammatical approaches to modeling pseudoknot structures of RNA sequences.

Matsui et al. [20] have proposed the pair stochastic tree adjoining grammars (PSTAGs) for modeling RNA secondary structures including pseudoknots. First, PHMMTSs defined on alignment of "trees" have been extended to PSTAGs defined on alignment of "TAG trees", which represent the derivation process of TAGs and are functionally equivalent to the derived trees of TAGs. Then an efficient dynamic programming algorithm of PSTAGs has been developed for obtaining an optimal structural alignment including pseudoknots. Matsui et al. have provided strong experimental evidence that modeling pseudoknot structures significantly improves the prediction accuracies of RNA secondary structures (as shown in Table 2).

2.5 Structural RNA Sequence Alignment Based on Sankoff's Algorithm

Sankoff [28] has proposed a novel algorithm which simultaneously folds and aligns RNA sequences, and is identical to pair stochastic context-free grammars. The Sankoff algorithm can more accurately predict a common secondary structure for RNA sequences than individual secondary structures for each RNA sequence. Furthermore, a predicted alignment of RNA sequences is more reliable due to considering secondary structures.

The Sankoff algorithm calculates alignments between all possible substructures on each sequence using dynamic programming (DP). For two RNA sequences A and B to be aligned, a DP table with four indexes is required to calculate alignments between $A[i, j]$, a substring of A from the ith to the jth base, and $B[k,l]$, a substring of B from the kth to the lth base. To divide each $A[i, j]$ and $B[k,l]$ into its substructures, optimal bifurcating points $i < a < j$ and $k < b < l$ are determined for each substructure. Thus, the Sankoff algorithm requires $O(n^3m^3)$ time and $O(n^2m^2)$ space for sequences of length n and m.

Due to the expensive computational complexity, no naive implementation of the Sankoff algorithm can align RNA sequences of length greater than about 100 bases using current computers. As a result, more efficient Sankoff-based methods have been proposed in which the search space for alignments is constrained by effective heuristics; Dynalign [18], PMcomp [10], FOLDALIGN [8], Stemloc [11]. However, even these methods are not fast enough to be applied to long sequences, such as screening the whole genome.

2.6 SCARNA: Stem Candidate Aligner for RNAs

As mentioned above, the Sankoff-based algorithms are not practical to apply to long sequences due to the expensive computational complexity. Therefore, some different approaches to structural alignments have been proposed. Tabei et al. [32] have proposed one such algorithm, Stem Candidate Aligner for RNAs (SCARNA).

SCARNA takes two unaligned RNA sequences. First, the base pair probability matrices are calculated by the McCaskill algorithm [21] for each RNA sequence. The base pair probability matrix contains probabilities of forming the base pair (i, j) for all positions $0 \leq i < j < m$ on a sequence of length m. Next, k continuous fragments of base pairs $(i, j), (i+1, j-1), \ldots, (i+k-1, j-k+1)$, each of which has the base pair probability above the threshold τ, are extracted as stem candidates from the base pair probability matrix for each sequence. Each stem candidate is decomposed into two stem components, $i, \ldots, i+k-1$ (left) and $j, \ldots, j-k+1$ (right). A stem component sequence (SCS) is a sequence of all stem components sorted by their positions in the sequence. Then two SCSs $X_1, \ldots, X_m$ and $Y_1, \ldots, Y_n$ for given sequences are aligned according to the following recurrence equations:

$$M(i, j) = \max \begin{cases} M(\alpha_i, \beta_j) + \Delta_s(i, j), \\ M(p_i, q_j) + s(i, j), \\ G(p_i, q_j) + s(i, j), \end{cases} \tag{1}$$

$$G(i, j) = \max \begin{cases} M(i-1, j), \\ M(i, j-1), \\ G(i-1, j), \\ G(i, j-1), \end{cases} \tag{2}$$

$$M(0,0) = 0,$$

$$M(0, \cdot) = M(\cdot, 0) = -\infty,$$

$$G(0,0) = G(0, \cdot) = G(\cdot, 0) = -\infty,$$

where $M(i, j)$ is the best score up to a pair of X_i and Y_j given that X_i matches Y_j and $G(i, j)$ is the best score given that X_i mismatches Y_j. X_{α_i} is the stem component which is 1-continuous to X_i, Y_{β_j} is that of Y_j, X_{p_i} is the nearest component which does not overlap with X_i and Y_{q_j} is that of Y_j. $s(i, j)$ is the match score for X_i and Y_j. $\Delta_s(i, j)$ is the difference of the match score for extending $X_{\alpha_i}, Y_{\beta_j}$

to X_i, Y_j. $s(i, j)$ and $\Delta_s(i, j)$ are derived from the RIBOSUM substitution matrix, the base pair probability matrix of each sequence, stacking energy of each stem and the loop length. Finally, the nucleotide sequences which do not align as SCSs are aligned by the simple pairwise alignment algorithm.

Since stem candidates extracted from base pair probability matrices of given sequences are decomposed into left and right stem components, the consistency in the sense of base pairs cannot be guaranteed. However, this violation creates the efficient structural alignment algorithm with $O(n^3)$ for time and $O(n^2)$ for space. Furthermore, by considering the loop length in calculating the match score, few base pairs would be broken in practice. As a result, the accuracy of the alignments produced by SCARNA is better than or at least comparable to the Sankoff-based alignment methods in many cases.

3 Discrimination of Functional RNA Sequences Using Support Vector Machines

Stochastic models such as SCFGs and hidden Markov models (HMMs) are limited in discriminating member sequences of an RNA family from non-members by only examining the probabilistic scores. Hence, we require stronger discriminative methods to detect and find non-coding RNA sequences.

Recently, the support vector machine (SVM) and kernel function techniques have been actively studied and used to propose solutions to various problems in bioinformatics [17, 30, 31]. SVMs are trained from positive and negative samples and have strong, accurate discrimination abilities. Hence, they are better suited to the discrimination tasks.

3.1 String Kernel

A brief overview of the string kernel [31] follows. Special attention is paid to the all-subsequences kernel as the kernel function for RNA sequences is a natural extension of the string kernel in measuring the similarity of two RNA sequences from the viewpoint of secondary structures.

General feature mapping for measuring the similarity between two biological sequences is defined by counting all contiguous and non-contiguous subsequences of the given sequences. For example, two DNA sequences `CTG` and `CAT` have 4 common subsequences: ϵ (the empty string), `C`, `T`, and `C`-`T`. The all-subsequences kernel calculates the inner product of the feature vectors by counting all commonly held non-contiguous subsequences. The inner product of the two sequences `CTG` and `CAT` is 4.

In protein sequence analysis, string kernels [31] have been proposed for use with SVMs to classify a protein family. In addition, string kernels are proven to work for remote homology detections of protein sequences, i.e. a superfamily.

3.2 Feature Space for RNA Sequences

First, we need to define a feature mapping the space of RNA sequences to a vector space such that the relative distance in the mapped vector space reflects the similarity between two RNA sequences. We consider a notion of similarity between RNA sequences in terms of common secondary structures. The simplest similarity feature is the count of base pair occurrences that the two RNA sequences have in common. That is, counting four kinds of base pairs: A-U, U-A, C-G, G-C, the feature space becomes a 4-dimensional vector space. This feature mapping is easily constructed when the secondary structure of a target RNA sequence is available. In this paper, we consider the more general case in which secondary structure information is not available. The strategy is to count all possible base pair candidates in the RNA sequences.

Example An RNA sequence AUCGAGUCG contains 3 occurrences of possible A-U base-pairs, 1 occurrence of a U-A base-pair, 4 occurrences of C-G base-pairs, and 2 occurrences of G-C base-pairs. (See Fig. 6 as illustration.) The feature space is a 4-dimensional vector space: (# of A-U base-pairs, # of U-A base-pairs, # of C-G base-pairs, # of G-C base-pairs), and the RNA sequence AUCGAGUCG is mapped into a vector $(3, 1, 4, 2)$.

A method better suited to measuring the similarity of secondary structures which two RNA sequences have in common is to count the occurrences of possible stacking base-pairs, called *stem structures*. For example, stems of length 2 constitute a 16-dimensional vector space (A(A-U)U), (A(U-A)U), (A(C-G)U), (A(G-C)U), (U(A-U)A), (U(U-A)A), (U(C-G)A), (U(G-C)A), (C(A-U)G), (C(U-A)G), (C(C-G)G), (C(G-C)G), (G(A-U)C), (G(U-A)C), (G(C-G)C), (G(G-C)C).

Example An RNA sequence AUCGAGUCG is mapped into a 16-dimensional vector space $(0, 1, 2, 0, 0, 0, 1, 0, 1, 0, 0, 2, 1, 0, 0, 0)$ for counting the occurrences of non-contiguous stems of length 2.

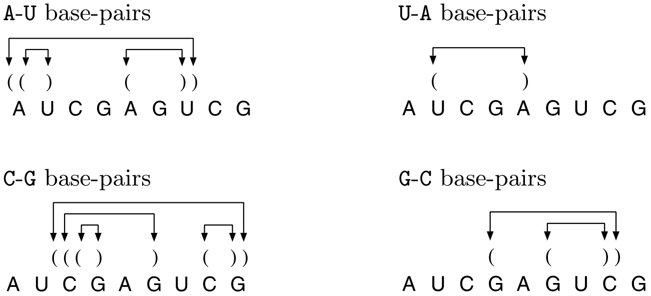


Fig. 6 Occurrences of A-U, U-A, C-G, G-C possible base-pairs contained in AUCGAGUCG

Sakakibara et al. [26] have proposed the similarity feature defined by all possible non-contiguous stems of arbitrary length for an RNA sequence, and a novel kernel function, called *stem kernel*, to calculate the inner product of two vectors in the feature space of two RNA sequences.

3.3 Stem Kernels

The dimension of the feature space for counting the occurrences of non-contiguous stems increases exponentially with the length of stem. In addition, the inner product of the feature vectors mapping from two RNA sequences requires a sum over all common stems. Hence, the direct computation of these features is not computationally efficient.

In order to compute the inner product of the feature vectors of two RNA sequences efficiently, the stem kernel function defined in recursive form has been proposed [26]. For an RNA sequence $v = a_1 a_2 \cdots a_n$ where a_i is a base (nucleotide), we denote a_k by $v[k]$, a contiguous subsequence $a_j \cdots a_k$ by $v[j, k]$, and the length of v by $|v|$. The empty sequence is indicated by ϵ. For base a, the complementary base is denoted as $\bar{a}$. For two RNA sequences v and w, the stem kernel K is defined recursively as follows:

$$\begin{aligned}
K(\epsilon, w) &= K(v, \epsilon) = 1, \quad \text{for all } v, w,\\
K(va, w) &= K(v, w)\\
&\quad + \sum_{v[k]=\bar{a}} \sum_{i<j \text{ s.t. } w[i]=\bar{a}, w[j]=a} K\bigl(v[k+1, |v|], w[i+1, j-1]\bigr).
\end{aligned}$$

To complete the recursive equations, the following recursive equation is required:

$$\begin{aligned}
K(v, wa) &= K(v, w)\\
&\quad + \sum_{w[k]=\bar{a}} \sum_{i<j \text{ s.t. } v[i]=\bar{a}, v[j]=a} K\bigl(v[i+1, j-1], w[k+1, |v|]\bigr).
\end{aligned}$$

Example We illustrate a one step calculation of the above recursive equation with two RNA sequences $v = \mathtt{AUCCUG}$ and $w = \mathtt{CACUAGG}$.

First, the two RNA sequences $v\mathtt{G} = \mathtt{AUCCUGG}$ and $w = \mathtt{CACUAGG}$ have (A-U), (C-G), and (C-(C-G)-G) in common. Second, assuming that $K(v, w) = 11$, we calculate $K(v[4, 6], w[2, 5]) = 1$, $K(v[4, 6], w[2, 6]) = 2$, $K(v[4, 6], w[4, 5]) = 1$, $K(v[4, 6], w[4, 6]) = 1$, $K(v[5, 6], w[2, 5]) = 1$, $K(v[5, 6], w[2, 6]) = 1$, $K(v[5, 6], w[4, 5]) = 1$, and $K(v[5, 6], w[4, 6]) = 1$. Then $K(v\mathtt{G}, w)$ is inductively calculated as follows:

$$\begin{aligned}
K(v\mathtt{G}, w) = K(v, w) &+ K\bigl(v[4, 6], w[2, 5]\bigr) + K\bigl(v[4, 6], w[2, 6]\bigr)\\
&+ K\bigl(v[4, 6], w[4, 5]\bigr) + K\bigl(v[4, 6], w[4, 6]\bigr) + K\bigl(v[5, 6], w[2, 5]\bigr)
\end{aligned}$$

$$+ K\big(v[5,6], w[2,6]\big) + K\big(v[5,6], w[4,5]\big) + K\big(v[5,6], w[4,6]\big)$$
$$= 11 + 9 = 20.$$

The computational complexity in calculating the stem kernel function using dynamic programming is estimated. Let the two input RNA sequences v and w be of length $m(=|v|)$ and $n(=|w|)$. The computational complexity of the stem kernel $K(v, w)$ for v and w is equal to calculating a table of $m^2 \times n^2$ elements for the stem kernel function $K(v', w')$ for all subsequences v' of v and w' of w. The calculation of this table is done in time proportional to $m^2 \times n^2$ by using a parsing algorithm and an auxiliary table.

3.4 Computational Experiments

Sakakibara et al. [26] have tested the abilities of the stem kernel with SVMs to discriminate member sequences of an RNA family from non-members in several different experiments. The discrimination performances were compared with the string kernel, specifically the all-subsequences kernel with gap weight (decay factor).

All datasets were taken from the RNA families database “Rfam” at Sanger Institute [7]. The negative sample was generated by randomly shuffling sequences with the same nucleotide composition as the positive sample sequence for a target RNA family. The shuffling of the sequences was accomplished while preserving the dinucleotide distribution. The discrimination performances of each method was evaluated by the 10-fold cross validation.

3.4.1 Discriminations of Several RNA Families

The discrimination abilities of the stem kernel and the string kernel were tested using sample sizes ranging from 10 sequences to 100 sequences for the five RNA families. Some of the results of the AUC score plots are shown in Fig. 7.

From these experimental results, it is clear that the stem kernel exhibits a relatively good discrimination performance even if a small number of training sequences is available. Since a small sample size implies that pairwise sequence similarities among sample RNA sequences decreases, the stem kernel shows the ability to tolerate weak sequence similarities. This advantage makes the stem kernel useful in practical situations because only a small number of sequences is currently available for many of the functional RNA families in the Rfam database. On the other hand, the string kernel is sensitive to sequence similarity although the discrimination performance decreased for small samples. It is especially interesting to note in the case of the CD snoRNA family that the discrimination accuracies of the stem kernel and string kernel reverse with the smaller sample size. These results imply that the stem kernel succeeds in capturing the secondary-structure features of RNA sequences for discrimination tasks.

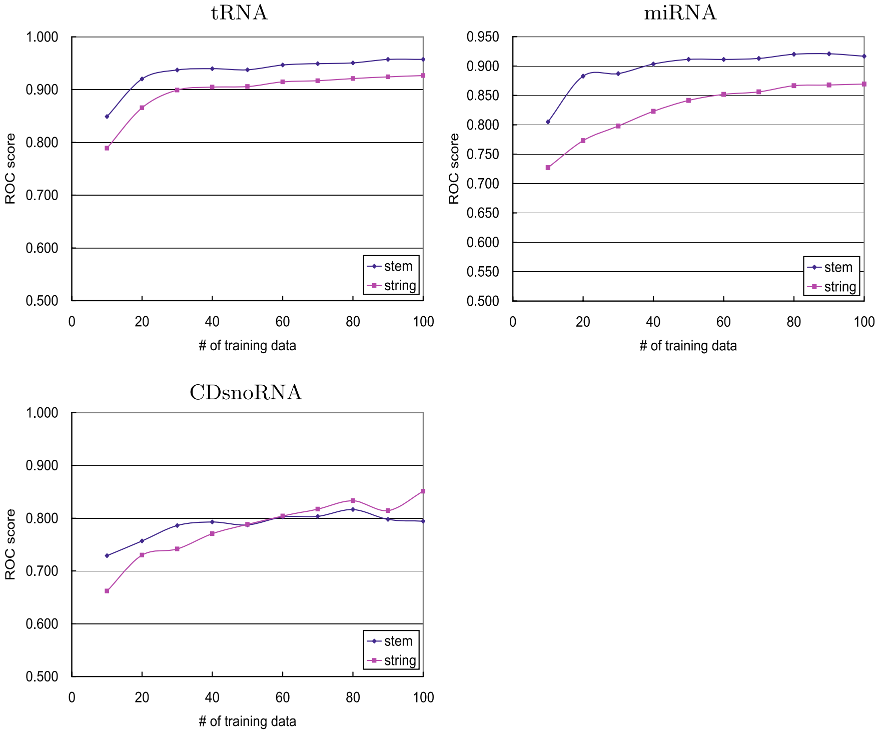


Fig. 7 AUC scores on different sample sizes from 10 sequences to 100 sequences for three RNA families

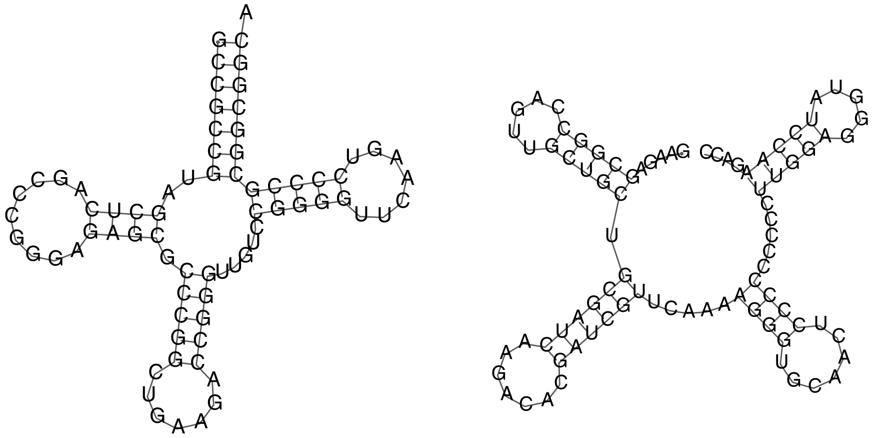
3.4.2 Finding a Remote RNA Family

Experiments applying the stem kernel to finding remote homologs of RNA sequences in terms of secondary structures were performed.

The family of Tymovirus/Pomovirus tRNA-like 3'-UTR element (Tymo_ tRNA-like) was considered to be a remote homolog of tRNAs. The secondary structures of Tymo_tRNA-like elements comprise very similar secondary structures to a cloverleaf structure in tRNAs (see Fig. 8 for both secondary structures). This family represents a tRNA-like structure found in the 3' UTR of Tymoviruses and Pomoviruses and is known to enhance translation. The tRNA-like structure is a highly efficient mimic of tRNA, interacting with tRNA-specific proteins as efficiently as tRNA [19].

SVMs trained from the positive and negative samples of "tRNA sequences" were applied to detect 28 Tymo_tRNA-like sequences.

The results in Table 3 show that the stem kernel achieved significantly greater detections of Tymo_tRNA-like sequences. The string kernel, in contrast, failed to detect Tymo_tRNA-like sequences adequately. This experimental result is a strong proponent for application of the stem kernel to discover novel RNA families from genome sequences.



tRNA

Tymo_ tRNA-like

Fig. 8 Typical secondary structures of tRNA (*left*) and Tymo_tRNA-like element (*right*)

Table 3 Prediction accuracy of detecting Tymo_tRNA-like sequences using SVMs trained from samples for "tRNA sequences"

	Tymo_tRNA-like		
	AUC	Specificity	Sensitivity
Stem Kernel	0.603	0.614	0.964
String Kernel	0.503	0.222	0.143

3.5 Marginalized Kernels on SCFGs

The marginalized kernel is a framework to construct kernels from stochastic models [34]. Kin et al. [13] have proposed marginalized kernels for SCFGs to analyze non-coding RNAs.

First, a distribution of SCFGs for a given RNA grammar is estimated from training RNA sequences using the inside-outside algorithm, a variant of the EM algorithm. Then for each sequence, expected counts for base pairs (state P: $4 \times 4 = 16$ patterns), single bases on the left side (state L: 4 patterns) and single bases on the right side (state R: 4 patterns) can be calculated by parsing the sequence with the inside algorithm and the outside algorithm under the estimated distribution. The first-order marginalized count vector is defined as the vector which consists of the expected counts normalized by the length of the sequence. Similarly, expected counts of any of two states which are continuously co-occurring can be calculated and the second-order marginalized count vector is defined. For example, co-occurring the state P and the state P continuously represents stacking of base pairs with $4^4 = 256$ patterns.

Kin et al. [13] have applied the marginalized count kernels for SCFGs to the kernel principal component analysis (kPCA) and have succeeded in classifying tRNAs accurately by their anti-codons.

4 RNA Gene Finding Based on the Comparative Genomics Approach

Recently, since genome sequencing for various species has been done, many researchers attempt to analyze biological processes and genes, including non-coding RNAs, by applying the comparative genomics approach.

In general, RNA gene finding based on the comparative genomics approach employs the following strategy: (1) construct a pairwise or multiple alignment of two or more RNA sequences; (2) predict that each mutation in the alignment occurs under a structurally conservative model or an independent model.

Rivas and Eddy [24] have developed QRNA which classifies a given pairwise alignment as one of three models: the coding model (COD) in which substitutions between synonymous codons occur frequently to conserve amino acid sequences, the non-coding model (RNA) in which covariances of base pairs occur frequently to conserve secondary structures, and the others (OTH). They have confirmed that non-coding RNAs can be detected sensitively from pairwise alignments of *Escherichia coli* and *Salmonella typhi*.

Washietl et al. [36, 37] have developed RNAz which detects a structurally conserved region from a multiple alignment by support vector machines. RNAz employs the averaged Z-score of minimum free energy (MFE) for each sequence and the structure conservation index (SCI). It is assumed that MFE for the common secondary structure is close to that for each sequence if a given multiple alignment is structurally conserved; SCI is defined as the rate of MFE for the common secondary structure to the averaged MFE for each sequence. MFE values for each sequence and the common secondary structure are calculated by RNAfold and RNAalifold in the Vienna RNA packages [9]. They have applied RNAz to genomic multiple alignments among four mammals (human, mouse, rat, and dog) from the UCSC genome browser [12], predicted 30,000 candidates of non-coding RNAs, and shown that about 40% of these are some kind of transcripts detected by the tiling array technology.

Pedersen et al. [22] have developed EvoFold based on phylo-SCFGs which assume that any mutations on each column of a given multiple alignment would occur under a given phylogenetic tree of sequences, and the mutation on single bases would occur more frequently than that on base pairs in conserved secondary structures. These assumptions improve the accuracy of predicting secondary structures [15]. They have applied EvoFold to genomic alignments among eight vertebrates including humans, predicted 48,000 candidates of functional RNAs, and estimated that 18,500 of them would be correct by statistical analysis.

The methods introduced above are used to detect functional RNAs from multiple alignments among several species. Therefore, NOT-conserved RNAs cannot be detected by these methods. Furthermore, some structurally conserved RNAs would be recognized as not-conserved RNAs on the multiple alignments which do not consider secondary structures. To avoid these drawbacks, we have to calculate multiple alignments with secondary structures by computationally more expensive methods, or develop a robust method against miss alignments.

Acknowledgements This work is supported in part by Grant-in-Aid for Scientific Research on Priority Area No. 17018029 and grants from the Non-coding RNA Project by New Energy and Industrial Technology Development Organization (NEDO) of Japan.

References

1. Akutsu T (2006) Recent advances in RNA secondary structure prediction with pseudoknots. Curr Bioinform 1:115–129
2. Cai L, Malmberg RL, Wu Y (2003) Stochastic modeling of RNA pseudoknotted structures: a grammatical approach. Bioinformatics 19(Suppl 1):i66–i73.
3. Do CB, Woods DA, Batzoglou S (2006) CONTRAfold: RNA secondary structure prediction without physics-based models. Bioinformatics 22:e90–e98
4. Durbin R, Eddy S, Krogh A, Mitchison G (1998) Biological sequence analysis. Cambridge University Press, Cambridge
5. Eddy SR, Durbin R (1994) RNA sequence analysis using covariance models. Nucleic Acids Res 22:2079–2088
6. Eddy SR (2001) Non-coding RNA genes and the modern RNA world. Nat Rev Genet 2:919–929
7. Griffiths-Jones S, Moxon S, Marshall M, Khanna A, Eddy SR, Bateman A (2005) Rfam: annotating non-coding RNAs in complete genomes. Nucleic Acids Res 33:D121–D124. http://www.sanger.ac.uk/Software/Rfam/
8. Havgaard JH, Lyngsø RB, Stormo GD, Gorodkin J (2005) Pairwise local structural alignment of RNA sequences with sequence similarity less than 40%. Bioinformatics 21:1815–1824
9. Hofacker IL (2003) Vienna RNA secondary structure server. Nucleic Acids Res 31:3429–3431
10. Hofacker IL, Bernhart SHF, Stadler PF (2004) Alignment of RNA base pairing probability matrices. Bioinformatics 20:2222–2227
11. Holmes I (2005) Accelerated probabilistic inference of RNA structure evolution. BMC Bioinform 6:73
12. Kent WJ, Sugnet CW, Furey TS, Roskin KM, Pringle TH, Zahler AM, Haussler D (2002) The human genome browser at UCSC. Genome Res 12:996–1006
13. Kin T, Tsuda K, Asai K (2002) Marginalized kernels for RNA sequence data analysis. Genome Inform Ser Workshop Genome Inform 13:112–122
14. Klein RJ, Eddy SR (2003) RSEARCH: finding homologs of single structured RNA sequences. BMC Bioinform 4:44
15. Knudsen B, Hein J (1999) RNA secondary structure prediction using stochastic context-free grammars and evolutionary history. Bioinformatics 15:446–454
16. Lafferty J, McCallum A, Pereira F (2001) Conditional random fields: probabilistic models for segmenting and labeling sequence data. In: Proceedings of the 18th international conference on machine learning, pp 282–289
17. Lodhi H, Saunders C, Shawe-Taylor J, Cristianini N, Watkins C (2002) Text classification using string kernels. J Mach Learn Res 2:419–444
18. Mathews DH, Turner DH (2002) Dynalign: an algorithm for finding the secondary structure common to two RNA sequences. J Mol Biol 317:191–203
19. Matsuda D, Dreher TW (2004) The tRNA-like structure of turnip yellow mosaic virus RNA is a $3'$-translational enhancer. Virology 321:36–46
20. Matsui H, Sato K, Sakakibara Y (2005) Pair stochastic tree adjoining grammars for aligning and predicting pseudoknot RNA structures. Bioinformatics 21:2611–2617
21. McCaskill JS (1990) The equilibrium partition function and base pair binding probabilities for RNA secondary structure. Biopolymers 29:1105–1119
22. Pedersen JS, Bejerano G, Siepel A, Rosenbloom K, Lindblad-Toh K, Lander ES, Kent J, Miller W, Haussler D (2006) Identification and classification of conserved RNA secondary structures in the human genome. PLoS Comput Biol 2:e33

23. Rivas E, Eddy SR (2000) The language of RNA: a formal grammar that includes pseudoknots. Bioinformatics 16:334–340
24. Rivas E, Eddy SR (2001) Noncoding RNA gene detection using comparative sequence analysis. BMC Bioinform 2:8
25. Sakakibara Y (2003) Pair hidden Markov models on tree structures. Bioinformatics 19(Suppl 1):i232–i240.
26. Sakakibara Y, Asai K, Sato K (2007) Stem kernels for RNA sequence analyses. In: 1st international conference on bioinformatics research and development (BIRD 2007). Lecture notes in bioinformatics, vol 4414. Springer, Berlin, pp 278–291
27. Sakakibara Y, Brown M, Hughey R, Mian IS, Sjölander K, Underwood RC, Haussler D (1994) Stochastic context-free grammars for tRNA modeling. Nucleic Acids Res 22:5112–5120
28. Sankoff D (1985) Simultaneous solution of the RNA folding, alignment and protosequence problems. SIAM J Appl Math 45:810–825
29. Sato K, Sakakibara Y (2005) RNA secondary structural alignment with conditional random fields. Bioinformatics 21(Suppl 2):ii237–ii242
30. Schölkopf B, Tsuda K, Vert JP (2004) Kernel methods in computational biology. MIT Press, Cambridge
31. Shawe-Taylor J, Cristianini N (2004) Kernel methods for pattern analysis. Cambridge University Press, Cambridge
32. Tabei Y, Tsuda K, Kin T, Asai K (2006) SCARNA: fast and accurate structural alignment of RNA sequences by matching fixed-length stem fragments. Bioinformatics 22:1723–1729
33. Thompson JD, Higgins DG, Gibson TJ (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Res 22:4673–4680
34. Tsuda K, Kin T, Asai K (2002) Marginalized kernels for biological sequences. Bioinformatics 18(Suppl 1):S268–S275.
35. Uemura Y, Hasegawa A, Kobayashi S, Yokomori T (1999) Tree adjoining grammars for RNA structure prediction. Theor Comput Sci 210:277–303
36. Washietl S, Hofacker IL, Lukasser M, Hüttenhofer A, Stadler PF (2005) Mapping of conserved RNA secondary structures predicts thousands of functional noncoding RNAs in the human genome. Nat Biotechnol 23:1383–1390
37. Washietl S, Hofacker IL, Stadler PF (2005) Fast and reliable prediction of noncoding RNAs. Proc Natl Acad Sci USA 102:2454–2459
38. Zuker M, Stiegler P (1981) Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. Nucleic Acids Res 9:133–148

Part III Gene Assembly in Ciliates

Strategies for RNA-Guided DNA Recombination

Angela Angeleska, Nataša Jonoska,
Masahico Saito, and Laura F. Landweber

Abstract We present a model for homologous DNA recombination events guided by double-stranded RNA (dsRNA) templates, and apply this model to DNA rearrangements in some groups of ciliates, such as *Stylonychia* or *Oxytricha*. In these organisms, differentiation of a somatic macronucleus from a germline micronucleus involves extensive gene rearrangement, which can be modeled as topological braiding of the DNA, with the template-guided alignment proceeding through DNA branch migration. We show that a graph structure, which we refer to as an assembly graph, containing only 1- and 4-valent vertices can provide a physical representation of the DNA at the time of recombination. With this representation, 4-valent vertices correspond to the alignment of the recombination sites, and we model the actual recombination event as smoothing of these vertices.

1 Introduction

Theoretical models for DNA recombination have been proposed for both DNA sequence reorganization [6, 10, 11] and topological processes of knotting [23]. DNA rearrangements in certain types of single-celled eukaryotes with two nuclei were modeled by Landweber, Kari, and subsequently Rozenberg and a group of authors who proposed an abstract model for these rearrangements based on string rewriting operations [6], followed by a model based on involvement of a new molecule called a template [21]. This paper expands on the template model and describes some recent theoretical and empirical results.

Several species of ciliates, including the model organisms *Oxytricha* and *Stylonychia*, undergo massive genome rearrangement during differentiation of an archival germline micronucleus into a somatic macronucleus capable of gene expression. These DNA processing events involve global deletion of 95–98% of the germline DNA, effectively eliminating *all* so-called "junk" DNA, including intergenic DNA as well as hundreds of thousands of intervening DNA segments (internal eliminated sequences, IESs) that interrupt genes. In *Oxytricha*, DNA deletion reduces a 1 Gb germline genome to a somatic genome of only 50 Mb, nearly devoid of noncoding DNA. As a result of this streamlining, the macronuclear "nanochromosomes" in spirotrichous ciliates typically encode only a single gene, each flanked by short regulatory sequences and 20 bp telomeres. A few two- or three-gene chromosomes have

A. Angeleska (✉)
University of South Florida, Tampa, FL, USA
e-mail: aangeles@mail.usf.edu

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_6, © Springer-Verlag Berlin Heidelberg 2009

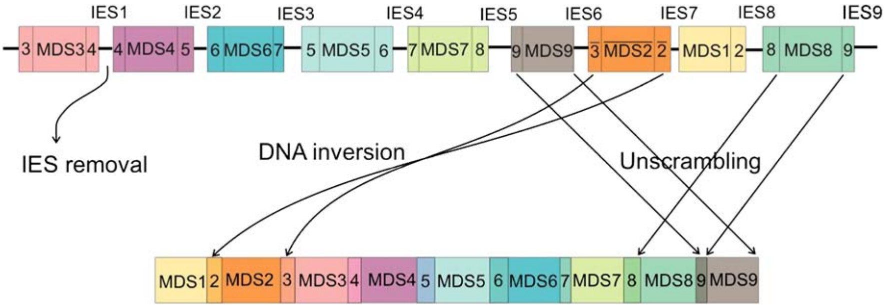


Fig. 1 Schematic representation of the scrambled Actin I micronuclear germline gene in *Oxytricha nova* (*top*) and the correctly assembled macronuclear gene (*bottom*). Each *block* represents an MDS, and each *line* between blocks is an IES. The numbers at the beginning and at the end of each segment represent the pointer sequences. Note that MDS3–MDS8 require permutation and inversion to assemble into the orthodox, linear order MDS1···MDS9 in the macronucleus. The *bars* above MDS2 and its *pointers* indicate that this block is inverted relative to the others, i.e., this sequence is the Watson–Crick reverse complement of the version in the macronucleus; from [20]

also been described (e.g., [16]) and more identified in the *Oxytricha* genome project. Because IESs interrupt coding regions in the micronucleus, each macronuclear gene may appear as several nonconsecutive segments (macronuclear destined sequences, MDSs) in the micronucleus. During macronuclear development, the IESs that interrupt MDSs in the micronucleus are all deleted. Moreover, the MDS segment order for thousands of genes in the micronucleus can be permuted or sequences reversed, with no coding strand asymmetry, as is typical of most genes.

Formation of the macronuclear genes in these ciliates thus requires any combination of the following three events: unscrambling of segment order, DNA inversion, and IES removal. Figure 1 is a diagram of a typical scrambled gene (see [20]) requiring all three events and shows the corresponding assembled macronuclear gene.

Although the general mechanism that guides this process of assembly is not known, there exist pointer-like sequences that are repeated at the end of the nth MDS and at the beginning of the $(n + 1)$st MDS in the micronucleus. Each pointer sequence is retained as only one copy in the macronuclear sequence. Such repetition of sequences suggests "pointer guided" homologous recombination. Several models for these processes have been proposed, including the models based on string rewriting operations in [6, 13] and graph reduction techniques [3]. All these models assume that the DNA rearrangement is performed when a correct pair of pointers align and splice.

Using the DNA recombinations in ciliates as a model system to describe DNA rearrangements that may occur more generally [15], a model based on RNA-guided DNA rearrangements was proposed in [2]. Recently, this model was supported by an experimental observation that maternal RNA templates may guide the DNA rearrangement in the early development of the macronucleus [19]. In this paper, we describe the model for double-stranded-RNA template-guided DNA rearrangements as proposed in [2] and briefly discuss the experimental findings in [19]. As a result of this model, in [2], we suggest that during the recombination process multiple pointers may align simultaneously, allowing an abstract representation of the molecules as a graph structure in space. We also proposed that recombinations are performed simultaneously at the positions where pointers have aligned. This initiates systematic studies of certain types of spatial graphs with rigid 4-valent vertices. The 4-valent vertices are used to represent pointer alignments and the DNA recombinations (splicings) are modeled as smoothings of the vertices. This mathematical model motivates a variety of theoretical questions about characterization of the structure of such spatial graphs. We show a brief description of some of these studies, while a more detailed analysis can be found in [1].

2 RNA-Guided DNA Recombination

Several models for DNA rearrangement processes in ciliates have been proposed, including the models in [6, 13] which assume that a correct pair of pointers align and splice.

Some such pointers, however, are as short as two nucleotides, but usually 2–20 nt, [5]. To account for the alignment of short pointers, Prescott et al. proposed a DNA template-guided recombination model [21]. However, recent evidence [19] suggests, instead of DNA, an involvement of long (single- or double-stranded) RNA molecules in the DNA processing, at least at some stages.

A new explicit model, expanding on the notion of templates proposed in [21] was developed in [2]. We elaborate on the latter one in this section. The model proposes DNA recombination using double- or single-stranded RNA as templates. The templates facilitate (align) the micronuclear DNA and allow recombination, without modifying any portion of the templates.

Our assumption that templates are dsRNA molecules means that the portion of the molecule that plays the role of a template is double-stranded. For easier representation, we depict the double-stranded molecules as ladders, ignoring the helical structure.

Let T be the dsRNA molecule that plays the role of a template, and let X and Y be two portions of a DNA molecule(s) that contain the same pointer.

The figures represent X, Y, and T as ribbons with orientation of the strands $5'$ to $3'$ indicated with an arrow. Base pairs are represented as vertical stripes (Fig. 2). The "upper" strand of X (denoted uX) reading $5'$–$3'$ contains block $\alpha\beta\delta$, where α is a portion of the ith MDS, β is the $(i+1)$st pointer, and δ is a portion of an IES. The "upper" strand of Y (uY), read $3'$–$5'$, contains a block $\bar{\epsilon}\bar{\beta}\bar{\gamma}$, where γ is a portion

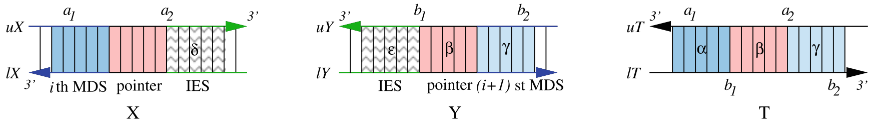


Fig. 2 Two DNA segments X and Y to be recombined, guided by a template T. The regions α and γ indicate portions of two consecutive MDSs. The region β indicates the pointer sequence

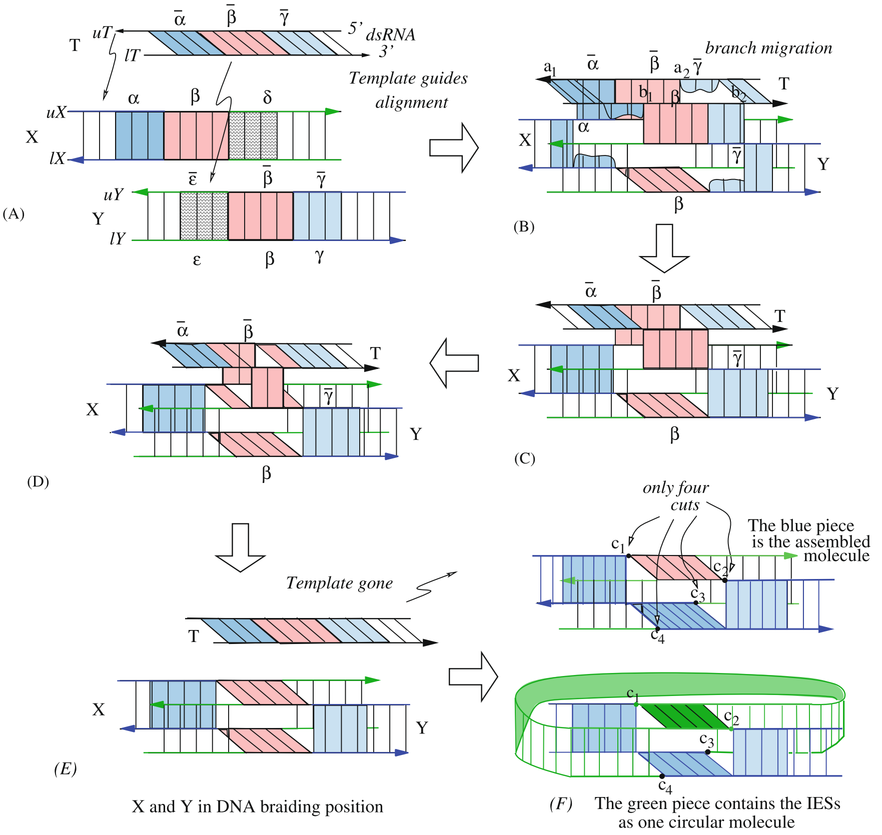


Fig. 3 Step by step model of DNA recombination guided by a dsRNA template

of the ith MDS and ϵ a portion of an IES (barred symbols represent Watson–Crick complements of unbarred symbols).

We propose a dsRNA template T, such that its "upper" strand in direction $3'$–$5'$ (denoted with uT) has a block $\bar{\alpha}\bar{\beta}\bar{\gamma}$ composed of sequences $\bar{\alpha}$, $\bar{\beta}$, and $\bar{\gamma}$. The lower strands of T, X, and Y (denoted lT, lX, lY) are complementary to the upper strands. The proposed steps of the recombination are as follows:

[A.] All three molecules X, Y, and T are present in the environment at the same time and the template strands find their corresponding complements in molecules X and Y as shown in Fig. 3(A). We assume that the template is short enough to initiate branch migration.

Even if the pointer sequence β is as short as two nucleotides and occurs more than twice in the DNA sequence, the context of β in T $(\alpha\gamma)$, the left context in X (α) and the right context in Y (γ) would be sufficient to lead to the alignment of the correct pointer sequences.

[B.] Through branch migration, the ends of the strands of template T, once in a neighborhood of complementary sequences, can easily anneal with their complements. An unzipping of the three double-stranded stripes occurs, from point a_1 to a_2 on X, from b_1 to b_2 on Y, and from a_1 to b_2 on T (see Fig. 2). A portion of lX and a portion of lY, containing $\bar{\beta}$ and β, respectively, become single-stranded. Because they are in close proximity to each other and single-stranded, hydrogen bonds form between the complementary regions connecting lX and lY, as shown in Fig. 3(B).

The original pairing and the new pairing are considered probabilistic. At some point during this process, cuts are made at c_1, c_2, c_3, and c_4 on the lower and upper backbones of X and Y as shown in Fig. 3(F). These cuts may depend on the way in which the pointers align and which portion of the pointer sequence participates in the branch migration process.

[C.] Note that the substrings of α, β, γ, $\bar{\alpha}$, $\bar{\beta}$, and $\bar{\gamma}$ that cannot find their complementary strings might remain unpaired. RNA-DNA hybrids are stronger than DNA-DNA, so the process (to be successful) may be thermodynamically driven. In the situation depicted in Fig. 3(C), only the single-stranded subsequences $\bar{\beta}$ and β from T hybridize to the corresponding complementary sequences of uX and uY.

[D.] In the next step, illustrated in Fig. 3(D), the hydrogen bonds, between uX and uT on one hand and lT and uY on the other hand, start to dissociate. Through branch migration, because RNA duplexes are more stable than RNA-DNA duplexes, the template strands release uX and uY. At the same time, enabled by strand complementarity, new hydrogen bonds develop between uX and uY.

[E.] Branch migration permits the complementary regions of uX and uY corresponding to β to hybridize, releasing the template (shown in Fig. 3(E)). Thus, the template is available to serve for further recombinations if needed.

We refer to the pairing between molecules X and Y shown in Fig. 3(E) as a *DNA vertex*. This portion shown in Fig. 3(E) is a molecule that has been studied and characterized *in-vitro* before (e.g. [22]) as a type of DX molecule known as "double parallel cross over molecule."

[F.] Figure 3(F) shows the resulting molecules obtained after the cuts are introduced at $c_1, \ldots, c_4$. The paired X and Y molecules containing the sequence $\alpha\beta\gamma$ indicate the new recombined molecule. Schematically, (assuming the cuts relieving possible strain have been introduced) the right portion of molecule Y rotates toward molecule X ("falls down") and the left portion of molecule X rotates toward molecule Y (also "falls down"), permitting the nicks to be ligated, forming the product strands.

If the portions that have undergone recombinations, portions X and Y, belong to the same DNA molecule, after recombination, the remaining fragments (containing sequence $\epsilon\beta\delta$) could be released as a circular molecule. Schematically, in this case, the left portion of molecule Y rotates toward molecule X ("goes up") and the right portion of molecule X rotates toward molecule Y (also "goes up") at which point, the nicks are ligated.

We note that the whole process is irreversible, as the resulting products of recombination do not contain the appropriate context for a template to weave new molecules.

3 Brief Summary of Experimental Conclusions

Recent experimental evidence in several ciliates [4, 8, 12, 17] indicates that small ∼27nt RNAs, produced during macronuclear development (and like piRNAs, associated with piwi proteins [9]), are involved in marking germline segments to be deleted or retained through chromatin modification. This suggests a possible involvement of short (single or double stranded) RNA molecules instead of template DNA. However, these experimentally described small RNAs could tag the deleted sequences and, therefore, could not function as templates to align retained DNA and promote MDS reordering.

The steps of DNA tagging for deletion and MDS reordering could be uncoupled, however, with the current experimental observations in [18] that suggest that simple IES excision events generally occur before the more complex reordering events. The results in [19] show that double-stranded RNA copies of (parental) macronuclear chromosomes are indeed present at an early stage of macronuclear development. Furthermore, RNA interference (RNAi) directed against putative RNA templates leads to aberrant gene unscrambling in the resulting progeny, either blocking the process completely or producing incorrect rearrangements, including over-deletion [19]. The RNAi experiments only disrupted rearrangement of the targeted gene. This suggests that the experiments specifically destroyed the proposed RNA templates.

In a key experiment to test the template model, both synthetic DNA and RNA versions of alternatively rearranged full-length nanochromosomes with two MDSs (MDS4 and MDS5 for one gene and MDS7 and MDS8 for a different gene) in the reversed order were introduced. Microinjection of these molecules into the parental macronucleus strikingly leads to epigenetic reprogramming of the new rearrangement pattern in the sexual progeny's genome, effectively scrambling the order of specific MDSs in each experiment (e.g., switching the order of MDS4 and 5, produced a product that reads 1–2–3–**5–4**–6...). Because the DNA versions could be transcribed into RNA along with other parental DNA molecules, and because these experiments worked whether DNA or RNA was microinjected, these results provide the first molecular evidence that maternal RNA templates instruct the process of DNA unscrambling in ciliates. Furthermore, the microinjection experiments revealed that the sequence of the product does not physically incorporate the sequence of the template (both were marked with distinct restriction sites), ruling out the possibility of recombination between template and germline DNA [19]. A small additional observation, however, was that point mutations in the template occasionally transfer to the product, implicating some involvement of RNA-guided DNA repair at recombination junctions [19].

4 Assembly Graphs and Recombination Strategies

4.1 *Toward an Abstract Model*

In this section, we define spatial graphs to model DNA structures in space during the recombination processes. Graphs were also used to model such recombination processes in [7].

First, we justify our approach from the point of view of our proposed model of recombination described in Sect. 2.

Consider the situation where two segments of DNA molecules have aligned pointers and are ready for recombination as presented in Fig. 3(E). Assume that X and Y are parts of the precursor DNA molecule from a germline gene in a stichotrichous ciliate. Suppose further that α is the last portion of the ith MDS sequence, β denotes the sequence of the $(i+1)$st pointer, and γ is the first portion of the $(i+1)$st MDS sequence. Sequences ϵ and δ are the last and first portions of IES sequences, respectively.

Schematically, this situation is depicted in Fig. 4. The stage at which the pointers align, and just before homologous recombination takes place, is depicted by an intersection of the two molecules at the pointer region (shown in Fig. 4, left). The resulting product of homologous recombination is depicted on the right of Fig. 4. After recombination, one of the resulting molecules contains two newly ordered MDSs, and the other contains the excised IES(s). As numerous templates can facilitate alignment of many pointers at once, a few, or even all of the recombination events may occur simultaneously. Therefore, we consider spacial graphs with 4-valent vertices modeling the case when multiple pointer sequences are aligned.

Consider the example in Fig. 1. We associate a graph to the micronuclear sequence depicted in Fig. 1 in the following way. For each pair of pointers that occurs in the micronuclear segment, we place a 4-valent vertex in the plane. Label the vertices with the corresponding number. There are 8 such vertices in this example, labeled 2 through 9 (see Fig. 5(A)). Next, we pick a point in the plane, called the

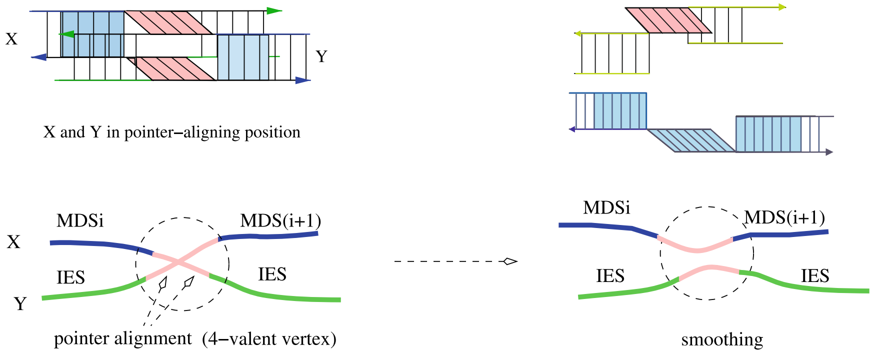

X
Y
X and Y in pointer–aligning position
MDSi
MDS(i+1)
X
IES
IES
Y
pointer alignment (4–valent vertex)
MDSi
MDS(i+1)
IES
IES
smoothing

Fig. 4 Schematic representation of the pointer alignment shown as a 4-valent vertex and the recombination result shown as smoothing of the vertex

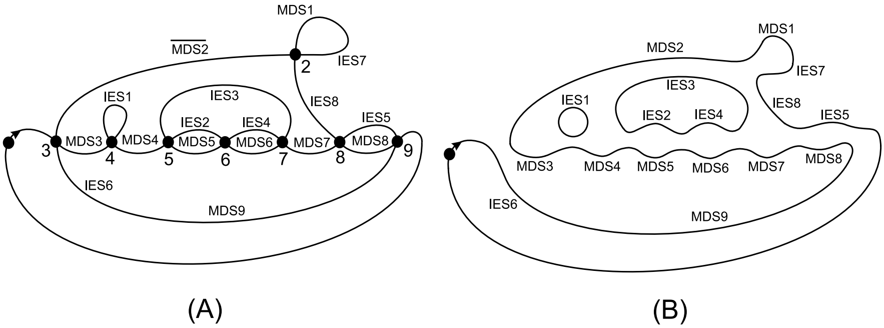


Fig. 5 Schematic representation of the simultaneous braiding and recombination process

base point, and as we read the string of pointers in the micronuclear gene (3-4-4-5-6-7-5-6-7-8-9-3-2-2-8-9), following a chosen direction, we start connecting the vertices: vertex 3 connects to vertex 4, loop back to vertex 4, connect to vertex 5, etc. When all vertices (pointers in the string) are exhausted, we connect the path back to the base point. In this way, we obtain a graph with 4-valent vertices. Here, a 4-valent vertex means a vertex that has 4 incident edges. Note that in the process of connecting the vertices, there may be instances when we have to intersect the arcs that have already been drawn. These intersections are depicted with over- and under-information as in knot diagrams (see, for example, [14]). The way the crossing information is depicted depends on the way the molecule is situated in space and needs investigation. The resulting "4-valent" graph is called a *spatial* graph, indicating that the graph is considered embedded in space, and its topological position in space is a question for consideration. Denote such obtained graph by Γ.

Next, label each edge connecting a pair of vertices in Γ by MDSi or IESj, such that starting at the base point, if one travels Γ following its orientation, the edge labels follow the appearance of the MDSs and IESs as they appear in the scrambled micronuclear gene. For the example shown in Fig. 1, after visiting vertex 3 the label of the next edge is MDS3, followed by IES1, then by MDS4 and so on. This labeling is shown in Fig. 5(A).

Note that the representation of the Actin I micronuclear gene from Fig. 1 as a spatial graph in Fig. 5(A) completely captures its MDS-IES structure such that each pair of pointers that occur in the gene is represented by a vertex in Γ. The DNA rearrangement can appear at every alignment of the pointers. As shown in Fig. 4, this is represented by smoothing of the vertices of Γ. Such smoothing can be performed simultaneously at each vertex of Γ. For the example of Actin I gene from Fig. 1, the result of the simultaneous smoothing is shown in the diagram depicted in Fig. 5(B). As shown, this result is composed of three connected components. Two of them are labeled only with IESs which indicates the IES excision. One of the components contains MDS1−MDS2−MDS3 − $\cdots$ −MDS9, which represents the assembled macronuclear gene in correct MDS order. Following this example, any MDS-IES micronuclear gene structure can be modeled by a spatial graph, and its assembly into a macronuclear gene can be viewed as a simultaneous smoothing of each vertex. We showed in [2] that simultaneous smoothing always leads to one component having all MDSs in a correct "macronuclear order." However, simultaneous rearrangement at all pointer sequences may not be realistic. Probably some rearrangements appear simultaneously, or in a close time period to be considered simultaneous, and identifying the sets of pointers that could undergo simultaneous rearrangement, as well as their possible order of appearance is the topic of the following sections.

In order to precisely formulate and study the MDS-IES micronuclear gene structures and their recombinations, we introduce the notion of assembly graphs as a special type of spatial graphs.

4.2 Assembly Graphs

A rigid vertex in a graph is a vertex with preassigned cyclic order of its incident edges such that each edge e incident to a vertex v has a predetermined "predecessor" edge, and a predetermined "successor" edge with respect to v which constitute the *neighbors* of e.

An *assembly graph* is a finite connected graph, where all vertices are rigid vertices of valency 1 or 4. A vertex of valency 1 is called an *end point*. Three examples of assembly graphs are depicted in Fig. 6. The one in Fig. 6(A) has two endpoints, and those in Fig. 6(B) and (C) have no endpoints. In Fig. 6(B), a nonplanar graph is depicted with crossing with an "over-arc" and the "under-arc" information provided which is not a vertex of the graph. The depicted graph is a projection of the spatial graph.

Write $|\Gamma|$ to denote the number of 4-valent vertices in Γ. The assembly graph is called trivial if $|\Gamma| = 0$. Note that the definition of an assembly graph implies that the number of end points is always even.

Different molecules from the micronucleus may be involved in the recombination process. They are represented with different "cyclic" or "linear" components in the assembly graph. In order to identify the molecules involved in the assembly process, i.e., the components in the assembly graph, we define transverse paths. A *transverse path* in an assembly graph Γ is a path of maximal length forming a sequence of vertices and nonrepeating edges in Γ such that any two consecutive edges incident

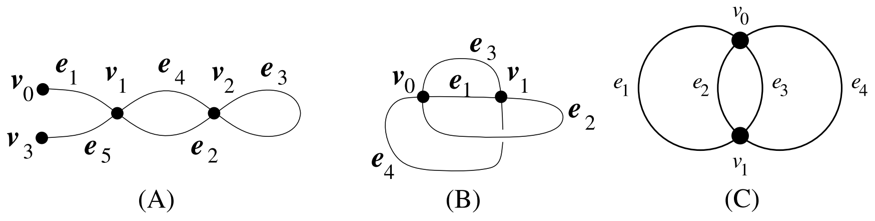


Fig. 6 Examples of assembly graphs. Simple assembly graphs (**A**) and (**B**), and a nonsimple assembly graph (**C**)

to the same vertex are not neighbors. In Fig. 6(A) and (B), examples for two types of transverse paths are depicted. A transverse path for each example is given by

$$\begin{aligned}
&\text{(A)} \quad (v_0, e_1, v_1, e_2, v_2, e_3, v_2, e_4, v_1, e_5, v_3),\\
&\text{(B)} \quad (v_0, e_1, v_1, e_2, v_0, e_3, v_1, e_4),\\
&\text{(C)} \quad (v_0, e_2, v_1, e_4),
\end{aligned}$$

respectively.

The case when the recombination appears intramolecular (i.e., only a single micronuclear molecule contains all MDSs of the macronuclear gene) is represented as an assembly graph with only one transverse path visiting all edges exactly once. We identify this case as a simple assembly graph. An assembly graph Γ is called *simple* if there is a transverse *Eulerian* path in Γ, meaning there is a transverse path γ that contains every edge from Γ exactly once.

In Fig. 6, graphs (A) and (B) are simple, but the graph (C) is not. Note that in a simple assembly graph, an Eulerian transverse path visits every nonend-point vertex twice.

As described in Sect. 4.1, the vertices of an assembly graph represent pairs of aligned pointers (see Fig. 4) and each of the edges is a representation of either an MDS or an IES sequence. Each pointer sequence is flanked on one side with an MDS sequence and on the other side with an IES sequence. Therefore, at each vertex, two edges representing MDS sequences meet. In order to track the sequence of MDSs if a path follows an edge with an MDS label, arrives at a vertex, the path follows with another edge having an MDS label only if the next edge is one of the neighboring edges. Using such paths in the assembly graph, we can represent precisely the correct ordering of the MDSs in the macronuclear gene, in fact, the macronuclear gene itself. As a macronuclear gene has no pointer sequences at the two ending MDSs, we are interested in paths that do not necessarily start at a vertex, called below, open paths. We define the notions more precise as follows.

Let Γ be an assembly graph. An *open path* in Γ is a homeomorphic image of the open interval (0, 1) in Γ.

An open path can be represented also by a sequence:

$$(e_1 \setminus v_0), v_1, e_2, v_2, e_3, \ldots, v_{m-1}, e_m, v_m, (e_{m+1} \setminus v_{m+1}),$$

where v_i's are vertices in Γ such that $v_i \neq v_j$ when $i \neq j$, e_i's are edges in Γ such that the initial vertex of e_1 and the terminal vertex of e_{m+1} are not included.

Two open paths are *disjoint* if they do not have a vertex in common.

A set of pairwise disjoint open paths $\{\gamma_1, \ldots, \gamma_k\}$ for a positive integer k is called *Hamiltonian* if their union contains all 4-valent vertices of Γ. An open path γ is called *Hamiltonian* if the set $\{\gamma\}$ is Hamiltonian. Finally, a *polygonal* path is an open path γ:

$$(e_1 \setminus v_0), v_1, e_2, \ldots, v_{m-1}, e_m, v_m, (e_{m+1} \setminus v_{m+1}),$$

such that e_i and e_{i+1} are neighbors for every $i \in \{1, 2, \ldots, m\}$.

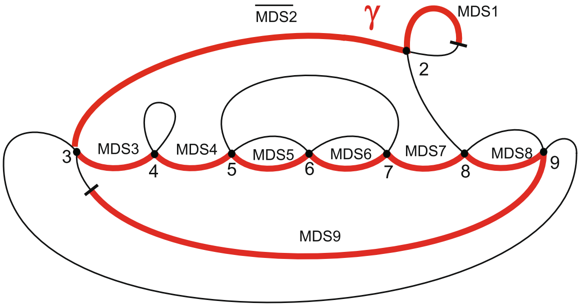


Fig. 7 Hamiltonian polygonal path γ in a simple assembly graph

Hamiltonian polygonal paths are of special interest, since they trace the correct order of the MDSs in the macronucleus. In Fig. 7, the assembly graph that models the micronuclear Actin I gene is given, with the Hamiltonian polygonal path γ that represents the macronuclear (after assembly) Actin I gene.

5 Successive Smoothings and Assembly Strategies

5.1 Smoothing

As mentioned in Sect. 4.1, it is assumed that DNA recombination appears soon after pointer sequences are aligned. The recombination as a result of splicing is represented as a smoothing of the vertex (see Fig. 4). If the pointer sequences are aligned in parallel, i.e., the pointer sequences appear in the same direction within the micronuclear sequence, then the smoothing of the vertex follows the predetermined direction of the graph, which we call parallel smoothing. If the alignment is antiparallel, i.e., the pointer sequences appear in opposite direction within the micronuclear sequence, then the smoothing of the vertex is performed opposite the direction of the graph, called nonparallel smoothing below. In this section, we define smoothing of assembly graphs determined by a polygonal path. More precisely, a *smoothing* of a 4-valent vertex in an assembly graph can be viewed as a removal of the vertex and connecting two parallel arcs as depicted in Fig. 8. There are two types of smoothings as depicted. To distinguish them, we have to fix an orientation of an assembly graph. An orientation of an assembly graph is a fixed direction of each transverse arc or cyclic component of the graph. These orientations are defined abstractly, independent from the $5' - 3'$ orientation of DNA. An assembly graph with a fixed orientation is called an *oriented* assembly graph. For an oriented assembly graph, each smoothing of a vertex is either orientation preserving (*parallel smoothing*, or p-smoothing, as in Fig. 8 left) or nonpreserving (*nonparallel smoothing*, or n-smoothing as in Fig. 8 right). We note that for a simple assembly graph, the type (p or n) of smoothing does not depend on a choice of the single orientation, as reversing it will reverse orientations of both arcs at every vertex.

The sets of Hamiltonian paths can be related to sets of smoothings as follows. Let Γ be an assembly graph. Let $\gamma = \{\gamma_1, \dots, \gamma_k\}$ be a Hamiltonian set of polygonal

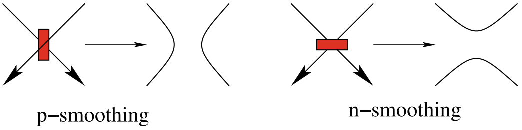


Fig. 8 Two types of smoothings, parallel (p-) smoothing (*left*) and nonparallel (n-) smoothing (*right*)

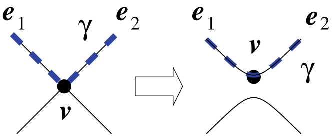


Fig. 9 Smoothing at a vertex v with respect to a polygonal path γ

paths for Γ. A smoothing of a 4-valent vertex v in Γ with respect to a polygonal path γ is a smoothing such that the arcs on which the path lies are connected as depicted in Fig. 9.

5.2 Strategies for Simultaneous Assemblies

As mentioned in the previous section, a polygonal path in an assembly graph can be seen as a unique representation of a macronuclear gene. A few of the pointer recombinations may occur simultaneously or within a narrow enough time window to be considered simultaneous. We assume that the recombinations that occur simultaneously do not disturb the natural MDS order in the macronuclear gene. In our model of an assembly graph, a set of simultaneous recombinations translates into a set of simultaneous smoothings, and the consistent order of the MDSs during the recombination process becomes a requirement that the corresponding polygonal path remains connected after these smoothings. In this section, we show a way to obtain smoothing strategies (i.e., recombination strategies) for a given simple assembly graph.

In this section, we work with simple assembly graphs with a single Hamiltonian polygonal path. In other words, we consider the case of a single micronuclear molecule containing a set of scrambled MDSs which assemble in a single macronuclear gene. We call such graphs *realizable*. A more detailed treatment of realizable graphs is included in [1].

Let Γ be a realizable assembly graph, and S a subset of vertices in Γ. An *S-partial smoothing of Γ with respect to a Hamiltonian polygonal path* γ, is an assembly graph with a set of 4-valent vertices $V(\Gamma)\setminus S$, denoted by $\tilde{\Gamma}_{(\gamma,S)}$, obtained by smoothing of all vertices in S with respect to γ.

We note that after a set of recombinations have been performed, the resulting molecule must contain all other MDSs in the inherited order, i.e., the remaining vertices must belong to a single polygonal path. Therefore, a subset $S\subset V(\Gamma)$ is called *linear* with respect to γ if the S-partial smoothing of Γ with respect to γ is a simple assembly graph.

Lacking further experimental evidence about the intermediate steps in the recombinations process leading from micronuclear molecules to macronuclear genes, we

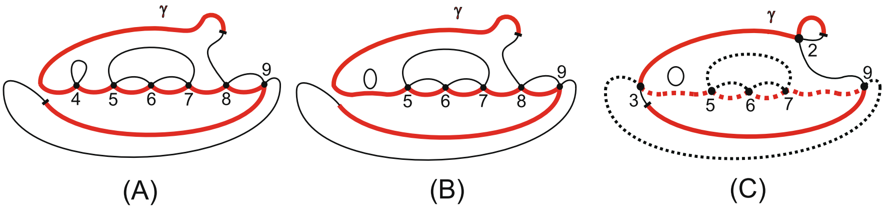


Fig. 10 Three choices of partial smoothings. (**A**) Example of a linear set: $\{2, 3\}$-smoothing of graph Γ from Fig. 7 results in a single transverse component containing γ. (**B**) Example of a successful but not linear set: $\{2, 3, 4\}$-smoothing of graph Γ from Fig. 7 results in two transverse components one of which contains γ. (**C**) Example of a nonlinear nonsuccessful set: $\{4, 8\}$-smoothing of graph Γ from Fig. 7 results in three transverse components two of which contain γ

also assume the following. If all MDSs of a macronuclear gene are part of a single micronuclear molecule, then after recombination at some part of the pointers, the MDSs remain as part of a single molecule. Hence, we further specify the sets of vertices for successful simultaneous smoothing. Let Γ be a realizable assembly graph, and γ be a Hamiltonian polygonal path. A subset $S \subset V(\Gamma)$ is called *successful with respect to* γ if the S-partial smoothing of Γ with respect to γ is an assembly graph that has a transverse arc component containing γ. (Here, we regard that γ remains in tact after the smoothing.)

A successful S-smoothing for a set $S \subset V(\Gamma)$ can be seen as a performed recombination at the pointers represented by the vertices in S, such that this recombination does not separate the MDSs (the edges from γ) that are expected to be part of the assembled gene. Hence, we regard a successful subset S of $V(\Gamma)$ as one step in the process of assembly such that the pointers in S are recombined at the same time. It is proven in [2] that for any simple assembly graph Γ, if $S = V(\Gamma)$, then S is successful. Note that every linear subset is successful.

Example 1 Let Γ be the assembly graph depicted in Fig. 7, where a polygonal Hamiltonian path γ is indicated by a thick line. Consider three sets of vertices $S' = \{2, 3\}$, $S'' = \{2, 3, 4\}$, and $S''' = \{4, 8\}$.

The assembly graphs $\tilde{\Gamma}_{(\gamma,S')}$, $\tilde{\Gamma}_{(\gamma,S'')}$, and $\tilde{\Gamma}_{(\gamma,S''')}$ depicted in Fig. 10(A), Fig. 10(B), and Fig. 10(C), respectively, are obtained from Γ by applying S'-, S''- and S'''-partial smoothings with respect to γ. We observe that S' is linear, since $\tilde{\Gamma}_{(\gamma,S_1)}$ is a simple assembly graph with a Hamiltonian polygonal path indicated with thick line. Therefore, S' is also successful.

On the other hand, S'' is successful, since $\tilde{\Gamma}_{(\gamma,S'')}$ has a transverse path (i.e., component) that contains γ. However, S'' is not linear, because $\tilde{\Gamma}_{(\gamma,S'')}$ has two transverse components: one containing γ and the other being a separate cyclic component. The set S''' is neither linear nor successful, since $\tilde{\Gamma}_{(\gamma,S''')}$ has three transverse components: one indicated with a dotted line, another indicated with a solid line containing parts of γ, and the third being a small cyclic part which does not contain any part of γ. Since there are two components that contain edges from γ, S''' is neither linear, nor successful.

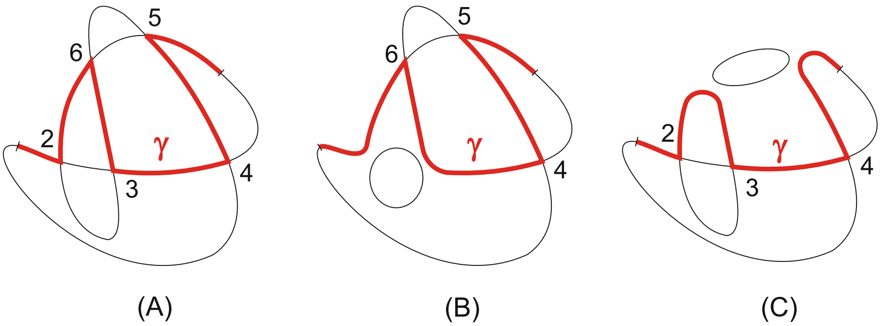


Fig. 11 (**A**) An example of an assembly graph and the corresponding Hamiltonian polygonal path. (**B**) S_1-smoothing of the graph in (A), for the successful set $S_1 = \{2, 3\}$. (**C**) S_2-smoothing of the graph in (A), for the set $S_2 = \{4, 5, 6\}$. S_2 is not successful and, therefore, cannot be the first smoothing set in a smoothing strategy

The observations obtained from this example about linear and successful sets of vertices can be formalized into a precise characterization of successful sets. Because this characterization is rather technical, the reader is referred to [1] for details. Here, we end with following result describing the "successful strategies" (sequences of sets) for assembling a macronuclear gene.

A macronuclear gene is completely assembled after recombination at all pairs of pointers. Therefore, we want to determine an assembly strategy for an assembly graph through partial successful smoothings such that the resulting graph has no 4-valent vertices.

Given a simple assembly graph Γ and a Hamiltonian polygonal path γ in Γ, a *successful smoothing strategy* for Γ with respect to γ is a sequence of pairwise disjoint subsets $(S_1, \dots, S_k)$ of $V(\Gamma)$ such that their union is $V(\Gamma)$ and S_i is successful in $\tilde{\Gamma}_{(\gamma, S'_{i-1})}$, where $S'_j = \bigcup_{h=1}^{j} S_h$.

Proposition 1 *There is one-to-one correspondence between the set of successful smoothing strategies and the set of all nested sequences*: $P_1 \subset P_2 \subset \cdots \subset P_k = V(\Gamma)$, *where* P_i *is successful for* Γ *for every* $i = 1, \dots, k$.

Example 2 For the simple assembly graph Γ and the polygonal Hamiltonian path γ given in Fig. 11(A), consider the sets $S_1 = \{2, 3\}$ and $S_2 = \{4, 5, 6\}$. The smoothing $\tilde{\Gamma}_{(\gamma, S_1)}$ is depicted in Fig. 11(B), and since γ belongs to a single transverse component of $\tilde{\Gamma}_{(\gamma, S_1)}$, S_1 is successful. The set $S_2 = \{4, 5, 6\}$ is not successful as seen in $\tilde{\Gamma}_{(\gamma, S_2)}$ depicted in Fig. 11(C) since portions of γ belong to two transverse components. Furthermore, $S_1 \cup S_2 = V(\Gamma)$. Therefore, the sequence (S_1, S_2) is a successful smoothing strategy, but (S_2, S_1) is not a successful smoothing strategy for Γ with respect to γ.

6 Concluding Remarks

In this paper, we presented a survey of the proposed model for RNA-guided DNA recombination and a theoretical model for spatial graphs that represent such recombination. Both models initiate new research, both experimentally and theoretically. Although recent experiments [19] support our proposal for RNA templates, further details about the recombination process are missing. For example, at what stage of the branch migration process are the strands cut, what type of enzymes are involved, and what properties of the pointers allow simultaneous recommendation to occur, etc. Further knowledge of the recombination events may allow precise (theoretical) description or characterization of the types of assembly graphs that are representations of naturally occurring recombination processes. Theoretically, we have only considered simultaneous smoothing of a single component (single micronuclear molecule) and performing such investigations for multicomponent graphs remains important in understanding unscrambling of a genome full of thousands of scrambled genes. Furthermore, the notion of smoothing along a polygonal path is new in mathematics and its study will provide new research pathways for knot and graph theory.

Acknowledgements N.J. was supported in part by NSF Grant CCF #0523928 and CCF #0726396. M.S. was supported in part by NSF Grant DMS #0603876. L.F.L. was supported in part by NSF Grant CCF #0622112.

References

1. Angeleska A, Jonoska N Saito M (2009) Discrete Appl Math (accepted)
2. Angeleska A, Jonoska N, Saito M, Landweber LF (2007) J Theor Biol 248(4):706–720
3. Brijder R, Hoogeboom HJ, Rozenberg G (2007) From micro to macro: how the overlap graph determines the reduction graph in ciliates. In: Lecture notes in computer science, vol 4639. Springer, Berlin, pp 149–160
4. Cavalcanti ARO, Landweber LF (2006) Insights into a biological computer: detangling scrambled genes in ciliates. In: Chen J, Jonoska N, Rozenberg G (eds) Nanotechnology: science and computation. Springer, Berlin
5. Chang WJ, Kuo S, Landweber LF (2006) Gene 368:72–77
6. Ehrenfeucht A, Harju T, Petre I, Prescott DM, Rozenberg G (2005) Computing in living cells. Springer, Berlin
7. Ehrenfeucht A, Harju T, Rozenberg G (2002) Theor Comput Sci 281:325–349
8. Garnier O, Serrano V, Duharcourt S, Meyer E (2004) Mol Cell Biol 24:7370–7379
9. Girard A, Sachidanandam R, Hannon GJ, Carmell MA (2006) Nature 442:199–202
10. Head T (1987) Bull Math Biol 49:737–759
11. Jonoska N, Saito M (2004) Algebraic and topological models for DNA recombinant processes. In: Calude CS, Calude E, Dinneen MJ (eds) Developments in language theory. Lecture notes in computer science, vol 3340. Springer, Berlin, pp 49–62
12. Juranek SA, Rupprecht S, Postberg J, Lipps HJ (2005) Eukaryot Cell 4:1934–1941
13. Kari L, Landweber LF (1999) Computational power of gene rearrangement. In: Winfree E, Gifford DK (eds) DNA based computers. AMS, Reading, pp 207–216
14. Kauffman LH (1991) Knots and physics. Series on knots and everything, vol 1. World Scientific, Singapore

15. Landweber LF (2007) Science 318:405–406
16. Kuo S, Chang W-J, Landweber LF (2006) Mol Biol Evol 23(1):4–6
17. Mochizuki K, Fine NA, Fujisawa T, Gorovsky MA (2002) Cell 110:689–699
18. Mollenbeck M, Zhou Y, Cavalcanti ARO, Jonsson F, Higgins BP, Chang WJ, Juranek S, Doak TG, Rozenberg G, Lipps HJ, Landweber LF (2008) PLoS ONE 3(6):e2330. http://www.plosone.org/article/info:doi/10.1371/journal.pone.0002330
19. Nowacki M, Vijayan V, Zhou Y, Schotanus K, Doak TG, Landweber LF (2008) Nature 451:153–158
20. Prescott DM, Greslin AF (1992) Dev Genet 13(1):66–74
21. Prescott DM, Ehrenfeucht A, Rozenberg G (2003) J Theor Biol 222:323–330
22. Seeman NC (2003) Nature 421:427–431
23. Sumners DW (1995) Lifting the curtain: using topology to probe the hidden action of enzymes. Not 42(5):528–537

Reality-and-Desire in Ciliates

Robert Brijder and Hendrik Jan Hoogeboom

Abstract The theory of gene assembly in ciliates has a number of similarities with the theory of sorting by reversal. Both theories model processes that are based on splicing, and have a fixed begin and end product. The main difference is the type of splicing operations used to obtain the end product from the begin product. In this overview paper, we show how the concept of breakpoint graph, known from the theory of sorting by reversal, can be used in the theory of gene assembly. Our aim is to present the material in an intuitive and informal manner to allow for an efficient introduction into the subject.

1 Introduction

Ciliates are single cell organisms that have two functionally different nuclei, one called micronucleus and the other called macronucleus. During sexual reproduction, a micronucleus is transformed into a macronucleus in a process called *gene assembly*. This is the most involved DNA processing in living organisms known today: The genome of the micronucleus is both functionally and physically very different from the genome of the macronucleus. Formal models for gene assembly has been developed; see, e.g., [7].

Another research area concerned with DNA processing is *sorting by reversal*; see, e.g., [1, 11, 13]. Two different species can have several segments in their genomes that are very similar, although their relative order (and orientation) may differ in both genomes. In the theory of sorting by reversal one tries to determine the number of reversal operations needed to reorder such a series of genomic 'blocks' from one species into that of another. An essential tool is the *breakpoint graph* (or reality and desire diagram) which is used to capture both the present situation, the genome of the first species, and the desired situation, the genome of the second species.

Motivated by the breakpoint graph, the notion of *reduction graph* was introduced in [4] into the theory of gene assembly. The intuition of 'reality and desire' remains in place: instead of two different species, we deal with two different nuclei—the reality is a gene in its (original) micronuclear form, and desire is the same gene but in its (final) macronuclear form.

R. Brijder (✉)
Leiden Institute of Advanced Computer Science, Universiteit Leiden, Niels Bohrweg 1,
2333 CA Leiden, The Netherlands
e-mail: rbrijder@liacs.nl

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_7, © Springer-Verlag Berlin Heidelberg 2009

In contrast to sorting by reversal, in gene assembly, the operations that splice the DNA are irreversible and can only be applied on a fixed pair (or two pairs) of positions in the DNA, called *pointers*. Therefore, although the intuition is the same, the definition of reduction graph is notably different from the definition of breakpoint graph (in the theory of sorting by reversal). Surprisingly, where the breakpoint graph is mostly useful to determine the number of needed operations, the reduction graph has different uses in the theory of gene assembly [2–4]. In this chapter, we provide a brief introduction in both sorting by reversal and gene assembly, we give an intuitive foundation of the reduction graph, and we summarize its uses. As this is an informal account of the material, formal definitions are sometimes and proofs are always omitted.

2 Sorting by Reversal

During nature's evolution, the genomes of the species change. One such change is due to inversion in which two pieces of a chromosome break and recombine in a different way; see Fig. 1. The end result is that the segment y between the two breakpoints is inverted; this is indicated by $\bar{y}$ in the figure. In this way, two different species can have several contiguous segments in their genomes that are very similar, although their relative order (and orientation) may differ in both genomes. For example, consider the two chromosomes in Fig. 2. Both chromosomes have 9 segments in common, however, their relative order and orientation differs. The breakpoints of a chromosome are the borders of each two consecutive segments. Figure 3 shows the application of such an inversion, called reversal, on the breakpoint between segments 0 and $\bar{2}$ (2 inverted) and the breakpoint between segments $\bar{1}$ and 6. These two breakpoints are indicated by two small arrows in the figure.

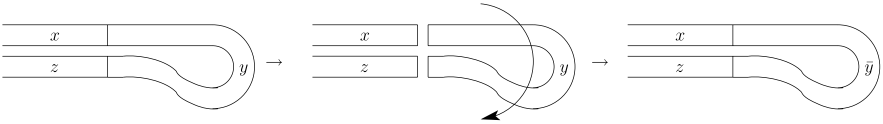
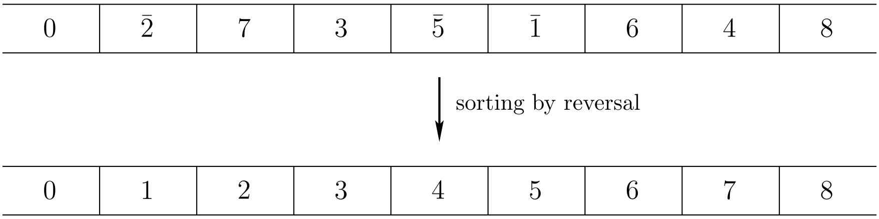


Fig. 1 Inversion within a chromosome



0	$\bar{2}$	7	3	$\bar{5}$	$\bar{1}$	6	4	8

0	1	2	3	4	5	6	7	8

Fig. 2 Two chromosomes of different species and their common contiguous segments

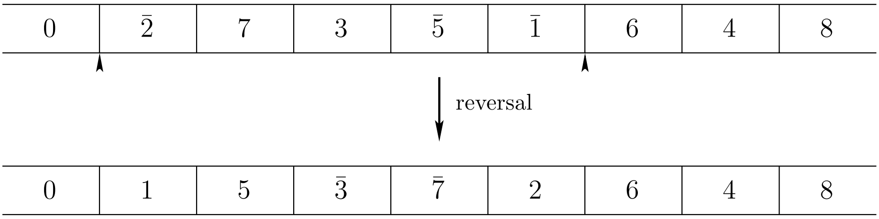

0 $\bar{2}$ 7 3 $\bar{5}$ $\bar{1}$ 6 4 8
reversal
0 1 5 $\bar{3}$ $\bar{7}$ 2 6 4 8

Fig. 3 Applying a reversal on the chromosome

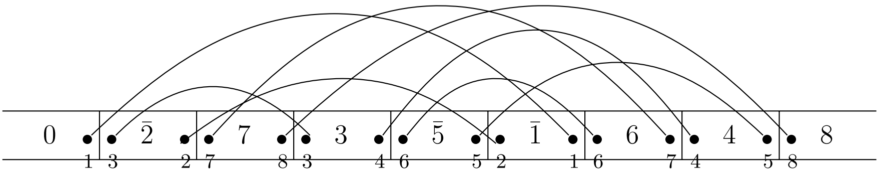


Fig. 4 The breakpoint graph of the given chromosome

In the theory of sorting by reversal, initiated by Hannenhalli and Pevzner in [8], one tries to determine the minimal number of reversals needed to convert such a series of genomic segments from one species into that of the other. The smaller this number, the more likely it is that their common ancestor in evolution is relatively young. Thus, this number can aid in constructing an ancestor tree of species, called a phylogenetical tree.

An essential tool in the theory of sorting by reversal is the breakpoint graph (or reality and desire diagram) which is used to capture both the present situation, the genome of the first species, and the desired situation, the genome of the second species. In this graph, we assign two vertices for each breakpoint, representing both sides of that breakpoint. These vertices are labeled such that segment i has a vertices labeled by i and $i + 1$. Then i represents the left-hand side and $i + 1$ the right-hand side of segment i. If i appears inverted in the genome, then w.r.t. the chromosome, i appears on the right-hand side and $i + 1$ on the left-hand side. Moreover, there are edges, called desire edges that connect vertices with the same label. In this way, each desire edge connects two sides of two segment that should be next to each other in the genome of the second species. In Fig. 4, these vertices and edges are depicted for our example.[1]

In addition to the desire edges, the breakpoint graph has a second set of edges, called reality edges. These edges connect each two vertices belonging to the same breakpoint. Thus, in Fig. 4, the left-most two vertices labeled by 1 and 3 are connected by a reality edge, and similarly for the next two vertices labeled by 2 and 7,

[1] It is customary for breakpoints graphs to instead let $2i - 1$ represent the left-hand side and $2i$ the right-hand side of segment i—in this way eliminating the need for labels. We choose this notation to make comparison with reduction graphs defined in the next chapter easier.

etc. The linear order of the vertices in the figure is therefore partially captured by the reality edges. However, the complete linear order of the vertices remains important, and hence the breakpoint graph should not be seen as a graph, but as a diagram where the vertices are drawn in this linear order. Consequently, reality and desire *diagram* is arguably a more appropriate name for this concept. One could extend the breakpoint graph with a third set of edges, for example called segment edges, connecting each two consecutive vertices belonging to the same segment. Then in Fig. 4, e.g. the two vertices labeled by 3 and 2 of segment $\bar{2}$ are connected by such a segment edge. In this way, we obtain a graph which retains the linear order of the vertices, and hence need not be seen as a diagram. We will later introduce these additional sets of edges in the context of gene assembly. Given only the breakpoint graph, it is possible to deduce, in a computationally efficient way, the minimal number of reversals needed to convert the genome from one species into that of the other [11].

3 Gene Assembly

Ciliates, a group of one-cellular organisms, differ from other organisms in that they have two nuclei that are radically different, both functionally and physically—this holds in particular for the stichotrichs group of ciliates. The two nuclei (which both can occur in arbitrary multiplicity) are called micronucleus (MIC) and macronucleus (MAC). Their names are due to their relative sizes within the ciliate.

All the genes occur in both the MIC and the MAC, but in very different forms. For each gene, however, one can distinguish a number of segments $M_1, \ldots, M_\kappa$, called MDSs (macronuclear destined segments), appearing in both the MIC and MAC form of that gene. In the MAC form, the MDSs appear as in Fig. 5: Each two consecutive MDSs overlap in the MAC gene. The gray areas in the figure where the MDSs overlap are called pointers. In the MIC form the MDSs appear scrambled and inverted with non-coding segments, called IESs (internal eliminated segments), in between. As an example, Fig. 6 shows the MIC form of the gene that encodes for the actin protein in a ciliate called sterkiella nova (see [5, 12], and [7]). Notice that the gene consists of nine segments, and that MDS M_2 occurs inverted, i.e. rotated 180 degrees, in the gene. It is important to realize that the number of MDSs, the specific permutation of the MDSs and the possible inversion in the gene of individual MDSs is fixed for a given gene in MIC form (and given species), but can be very different for different genes.

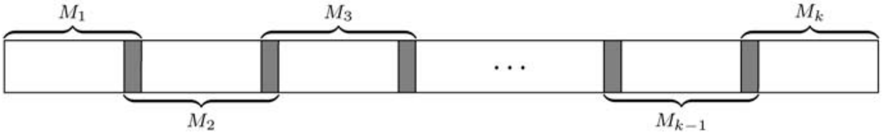


Fig. 5 The structure of a MAC gene consisting of κ MDSs

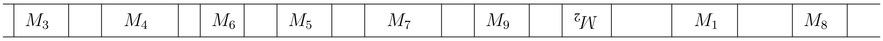


Fig. 6 The structure of the MIC gene encoding for the actin protein in sterkiella nova

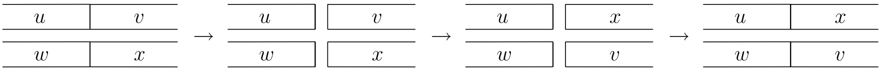


Fig. 7 Recombination operation

The process of gene assembly transforms the MIC into the MAC. This process occurs during sexual reproduction of two ciliates where first a MIC is formed holding half of the genetic information of each parent, and then a MAC is constructed from this newly formed MIC. During gene assembly, each gene in MIC form is transformed into the corresponding gene in MAC form. Since there are very many genes, this process is highly parallel. The transformation of a single gene from MIC form to MAC form is complex: All MDSs must be 'sorted' in the right order and must have the right orientation, and all IESs must be spliced out from between the MDSs. Therefore, quite a number of 'cutting and gluing', called recombination, is necessary for each such transformation. The process of recombination is illustrated in Fig. 7. Here, two segments, which may be part of the same sequence, are aligned, cut, and glued back together in a different way such that x and v are interchanged. Thus, recombination occurs on two locations within one sequence or between two sequences. We refer to [7] for an in-depth treatment of the biology of gene assembly.

4 Reality-and-Desire

In this section, we show how the concept of breakpoint graph can be used for gene assembly. Consider now a small example gene in MIC form given in Fig. 8. Recall that all MDSs are recombined to obtain the sequence given in Fig. 5. Now, we assign vertices to each side of each segment (MDSs or IESs) and draw edges connecting the MDSs in the order dictated by Fig. 5; see Fig. 9. Since gene assembly is accomplished using recombination, it is also known how the segments between the MDSs, the IESs, are to be connected. Therefore, also these segments are connected through edges. The edges now added are called *desire edges* due to the obvious similarities with those in the theory of sorting by reversal.

Consider now Figs. 10 and 11 where we have only drawn the pointers of the MDSs of Figs. 6 and 8, respectively. The pointer on the left-hand (right-hand, resp.) side of MDS M_i is denoted by i ($i+1$, resp.), except that M_1 (M_κ, resp.) has no pointer on the left-hand (right-hand, resp.) side. As usual, if MDS M_i appears inverted, then w.r.t. the MIC form of the gene, $\bar{i}$ appears on the right-hand side and $\overline{i+1}$ on the left-hand side.

We argue that the desire edges given in Fig. 9 can be constructed given only the sequence of pointers given in Fig. 11. Indeed, pointers indicate the pairs of locations

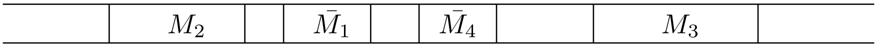


Fig. 8 An example sequence of MDSs

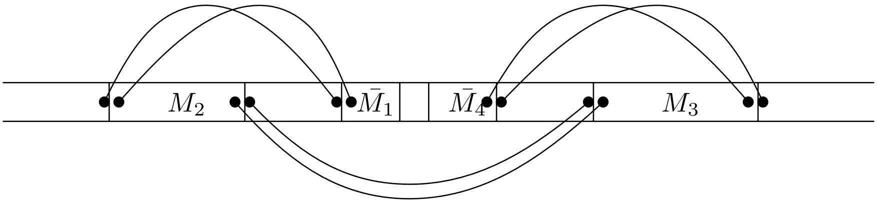


Fig. 9 An example sequence of MDSs with desire edges

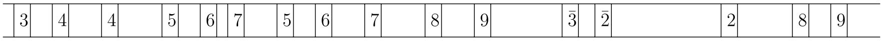


Fig. 10 The pointers of the MIC gene of Fig. 6

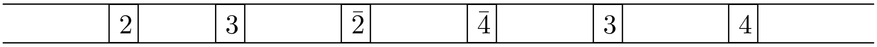


Fig. 11 An example sequence of pointers

of the MIC form of a gene on which recombination takes place. Each two locations corresponding to the same pointer are used together in a recombination operation. The segments between the pointers remain intact during gene assembly. Thus, for each pointer occurrence in the sequence, we assign two vertices representing the left and the right side of the pointer. If v_1 and v_2 are vertices belonging to the first occurrence of a pointer p and v_3 and v_4 belong to the other occurrence of p, then we connect *either* v_1 with v_3 and v_2 with v_4 *or* v_1 with v_4 and v_2 with v_3—this choice depends on the relative orientation of the two occurrences of pointers. In Fig. 11, the orientation of pointers 2 and 4 are different, called positive, and the orientation of pointer 3 is the same, called negative. If a pointer is positive, then the corresponding desire edges are drawn in such a way they 'cross' and if the pointer is negative, then they are drawn parallel to each other; see Fig. 12 where we have augmented Fig. 11 with desire edges. Notice that the desire edges are identical to that of Fig. 9. Therefore, in the remaining, we will abstract from the notions of MDSs and IESs and only consider sequences of pointers. Now, if we add a second set of edges, connecting each pair of vertices of a segment (thereby representing the segments), then we have obtained a graph that represents the end result after all recombination has taken place. These edges, which are called *reality edges*, are represented as 'double edges' and are depicted in Fig. 13. We have added special vertices s and t representing both ends of the gene in MIC form. Since each vertex (except s and t) belongs to a pointer, we label each vertex by the pointer (without possibly its bar) it belongs to. The obtained graph, called *reduction graph*, for our

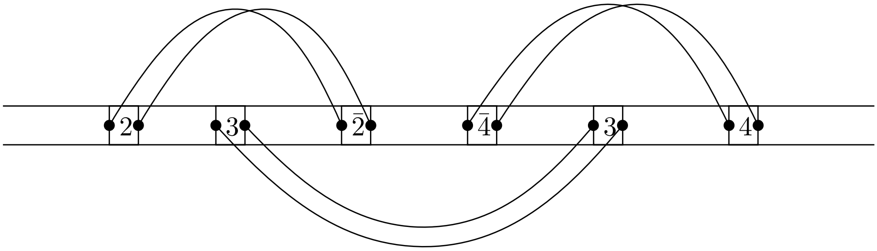


Fig. 12 An example sequence of pointers with desire edges

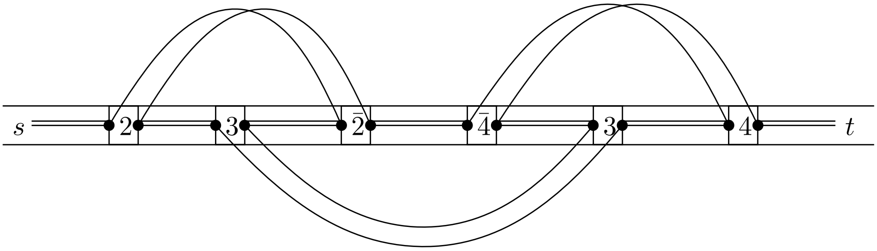


Fig. 13 An example sequence of pointers and its reduction graph

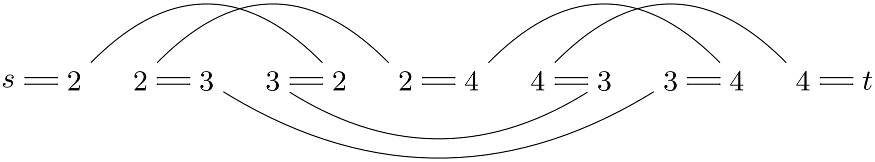


Fig. 14 The reduction graph of the example

example is given in Fig. 14. The same graph is depicted in Fig. 15—we have only rearranged the vertices.

Recall that this graph represents the end result after gene assembly has been performed: The reality edges represent the segments and the desire edges connect the segments as they appear in the end result. We notice in Fig. 15 one linear connected component and one cyclic connected component. Therefore, during gene assembly, a circular molecule must have been excised for this gene. It can be easily verified that a reduction graph always has exactly one linear connected component and zero or more cyclic connected components.

We now formally define the notions, including reduction graph, mentioned above. The sequences of pointers in Figs. 10 and 11 and will be represented as the strings $34456756789\bar{3}\bar{2}289$ and $23\bar{2}\bar{4}23$, respectively. Each such string is called a *legal string* since it is a string over $\Pi = \{2, 3, \ldots, \kappa\} \cup \{\bar{2}, \bar{3}, \ldots, \bar{\kappa}\}$, for some fixed κ, and for each $p \in \Pi$ that occurs in u, u contains exactly two occurrences

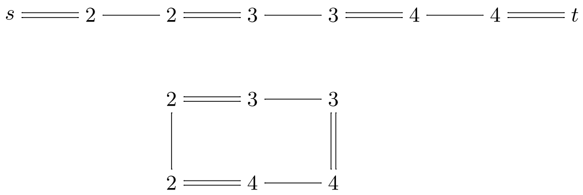


Fig. 15 The reduction graph of Fig. 14

from $\{p, \bar{p}\}$. We will use the convention that $\bar{\bar{p}} = p$ for $p \in \Pi$. We also define $\Delta = \{2, 3, \dots, \kappa\}$.

Although each gene in MIC form can be represented by a legal string, there are legal strings, e.g. $2324\bar{3}4$, that do not correspond to a sequence of MDSs, and hence cannot correspond to a gene in MIC form. Hence, for those strings, a figure similar to Fig. 9 cannot, but a figure similar to Fig. 13 *can* be given. In fact, it can be shown that renumbering the symbols of $2324\bar{3}4$ in a way that respects the orientation of the pointers (positive or negative) cannot lead to a legal string corresponding to a sequence of MDSs; see Chapter 8 in [7]. From now on, we will work with legal strings, and consequently the notions and results are given in greater generality: Fundamentally, we deal with arbitrary recombination using pointers.

For $p \in \Pi$, we define $\mathbf{p} = p$ if $p \in \Delta$ and $\mathbf{p} = \bar{p}$ if $p \in \bar{\Delta}$, i.e. $\mathbf{p}$ is the 'unbarred' variant of p. The *domain* of a string $v \in \Pi^*$ is $\mathrm{dom}(v) = \{\mathbf{p} \mid p \text{ occurs in } v\}$. For a pointer p and a legal string u, if both p and $\bar{p}$ occur in u, then we say that both p and $\bar{p}$ are *positive* in u; if on the other hand only p or only $\bar{p}$ occur in u, then both p and $\bar{p}$ are *negative* in u. So, every pointer occurring in a legal string is either positive or negative in it.

A *2-edge colored graph* is a 6-tuple $G = (V, E_1, E_2, f, s, t)$, where V is a finite set of *vertices*, $s, t \in V$ are called the *source* and *target*, and $f : V\backslash\{s, t\} \rightarrow \Gamma$ is a vertex labeling function. There a two (not necessary disjoint) sets of undirected edges $E_1, E_2 \subseteq \{\{x, y\} \mid x, y \in V, x \neq y\}$. The range of f of 2-edge colored graph G is denoted by $\mathrm{dom}(G)$. Also, 2-edge colored graphs G and G' are considered *isomorphic*, denoted $G \approx G'$, when they are equal modulo the identity of the vertices. However, the labels of the identified vertices must be equal. Also, the source (target, resp.) of G needs to be identified with the source (target, resp.) of G'.

Definition 1 Let $u = p_1 p_2 \cdots p_n$ with $p_1, \dots, p_n \in \Pi$ be a legal string. The *reduction graph of* u, denoted by $\mathcal{R}_u$, is a 2-edge colored graph (V, E_1, E_2, f, s, t), where

$$V = \{I_1, I_2, \dots, I_n\} \cup \{I'_1, I'_2, \dots, I'_n\} \cup \{s, t\},$$

$$E_1 = \{e_0, e_1, \dots, e_n\} \quad \text{with}$$

$$e_i = \{I'_i, I_{i+1}\} \text{ for } 0 < i < n,\ e_0 = \{s, I_1\},\ e_n = \{I'_n, t\},$$

$$E_2 = \big\{\{I'_i, I_j\}, \{I_i, I'_j\} \mid i, j \in \{1, 2, \dots, n\} \text{ with } i \neq j \text{ and } p_i = p_j\big\}$$

$$\cup \big\{\{I_i, I_j\}, \{I'_i, I'_j\} \mid i, j \in \{1, 2, \dots, n\} \text{ and } p_i = \bar{p}_j\big\}, \quad \text{and}$$

$$f(I_i) = f(I'_i) = \mathbf{p}_i \quad \text{for } 1 \leq i \leq n.$$

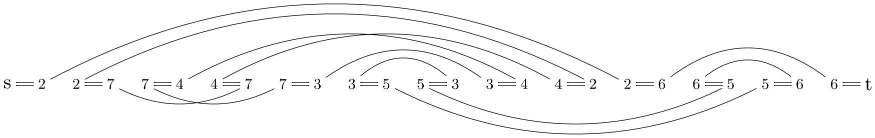
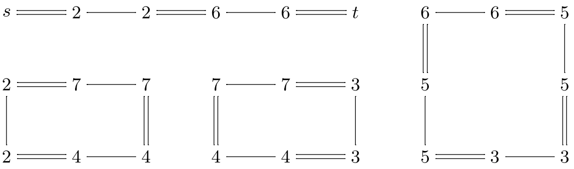


Fig. 16 The reduction graph $\mathcal{R}_u$ of $u = 2\bar{7}47353\bar{4}2656$

Fig. 17 The reduction graph $\mathcal{R}_u$ of Fig. 16



The exact identities of the vertices in the definition of reduction graph are not essential and, therefore, in graphical depictions, we represent the vertices (except for s and t) by their labels (as we did in Fig. 14). Note that the reduction graph is defined for the general concept of legal strings. Therefore, the reduction graph represents the end product after recombination of arbitrary sequences of pointers (which by definition come in pairs)—not only those that correspond to sequences of MDSs.

As another example, Figs. 16 and 17 show the reduction graph $\mathcal{R}_u$ of legal string $u = 2\bar{7}47353\bar{4}2656$. Although we will sometimes use other legal strings as well, we will use this legal string as our running example.

5 The Form of Reduction Graphs

Since legal strings represent the begin product (gene in MIC form) and the corresponding reduction graph the end result (the same gene in MAC form and its excised products), it is natural to study the possible forms of reduction graphs. Formally, we characterize in this section the (2-edge colored) graphs that are (isomorphic to) reduction graphs.

Certainly, a graph G isomorphic to a reduction graph must be a 2-edge colored graph (V, E_1, E_2, f, s, t) such that each label is a (unbarred) pointer, and each label must occur exactly four times. Each vertex must be connected to exactly one (reality) edge from E_1, and each vertex, except s and t, must be connected to exactly one (desire) edge from E_2. Finally, edges from E_2 must connect vertices with a common label. Let us call these graphs *abstract reduction graphs*, and let ARG be the set of all abstract reduction graphs. It turns out that there are graphs in ARG that are not (isomorphic to) reduction graphs; one such example is given in Fig. 18.

To obtain a characterization, we need one more property of reduction graphs: the possibility to linearly order the vertices as done in Fig. 16. To make this linear order of vertices explicit, we introduce *merge edges* to the reduction graph as done in Fig. 19.

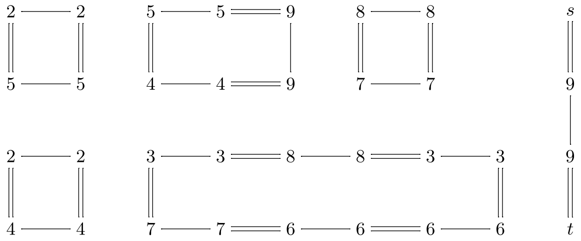


Fig. 18 An abstract reduction graph

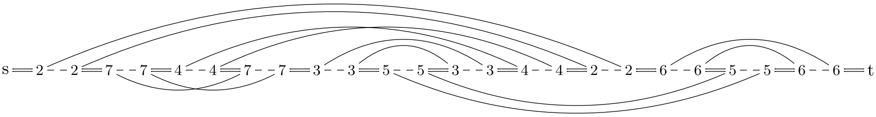


Fig. 19 The reduction graph $\mathcal{R}_u$ of Fig. 16 augmented with merge edges

Remark 1 In the breakpoint graph, the reality edges are actually the merge edges in the reduction graph. Thus, perhaps it would have been more appropriate to call merge edges reality edges, and reality edges, e.g. ‘segment edges’ (since they represent segments of DNA) in the reduction graph. The notion of segment edges is not introduced in breakpoint graphs but it is implicitly present since the graph is drawn as a circular diagram: the reality and desire diagram.

Now, when is a set of edges M for $G \in \mathrm{ARG}$ a set of merge edges? Like desire edges they have the properties that (1) the edges connect vertices with a common label and (2) each vertex except s and t is connected to exactly one merge edge. Moreover, M and the set E_2 are disjoint—no desire edge is parallel to a merge edge. Finally, the reality edges and merge edges must allow for a path from s to t passing each vertex once. This last requirement is equivalent to the fact that the reality and merge edges induce a connected graph.

If it is possible to add a set of merge edges to the graph, then it is not difficult to see that the graph is isomorphic to a reduction graph $\mathcal{R}_u$. Indeed, we can identify such a u for this reduction graph by simply considering the path from s to t alternating over the reality and merge edges. The orientation (whether it is positive or negative) of each pointer is determined by the crossing or non-crossing of the desire edges (exactly as in the definition of reduction graph). Thus, e.g. based on Fig. 19, we see that legal string $u = 2\bar{7}47353\bar{4}2656$ corresponds to this graph. However, also, e.g. legal string $2\bar{7}47\bar{3}5\bar{3}\bar{4}2656$ corresponds to this graph. We call these legal strings equivalent. Formally, we say that legal strings u and v are *equivalent*, denoted by $u \approx v$, if there is homomorphism $\varphi : \Pi^* \to \Pi^*$ with $\varphi(p) \in \{p, \bar{p}\}$ and $\varphi(\bar{p}) = \overline{\varphi(p)}$ for all $p \in \Pi$ such that $\varphi(u) = v$. Thus, the set of legal strings corresponding with an reduction graph with merge edges is an equivalence class w.r.t. $\approx$.

It is shown in [2] that surprisingly, $G \in \mathrm{ARG}$ has a set of merge edges precisely when the pointer-component graph $\mathcal{PC}_G$, defined below, is a connected graph. In other words, G is (isomorphic to) a reduction graph iff $G \in \mathrm{ARG}$ and $\mathcal{PC}_G$ is a connected graph.

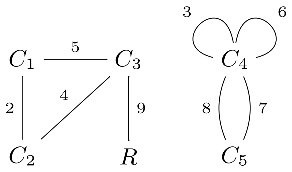


Fig. 20 The pointer-component graph of the abstract reduction graph from Fig. 18

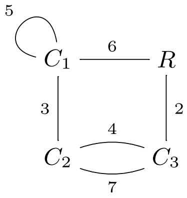


Fig. 21 The pointer-component graph of the reduction graph from Fig. 17

The pointer-component graph represents how the labels of graphs in ARG (and in particular reduction graphs) are distributed among its connected components. For $G \in$ ARG, the *pointer-component graph of* G, denoted by $\mathcal{PC}_G$, is a multi-graph (we allow loops and parallel edges), where the vertices are the connected components of G, the edges are the elements of $\mathrm{dom}(G)$, and an edge $p \in \mathrm{dom}(G)$ connects two different connected components C_1 and C_2 if both contain a vertex labeled by p. If there is only one connected component C with vertices labeled by p, then edge p is a loop on C. Note that for each label p there are exactly two desire edges with vertices labeled by p. Hence, each label is present in at most two connected components, and $\mathcal{PC}_G$ is well defined.

The pointer-component graph of the graph of Fig. 18 is given in Fig. 20. The linear connected component of Fig. 18 is denoted by R in the figure. Since the graph in Fig. 20 is not a connected graph, the graph of Fig. 18 is *not* isomorphic to a reduction graph. On the other hand, the pointer-component graph of the abstract reduction graph from Fig. 17, given in Fig. 21, is a connected graph—confirming that this graph is isomorphic to a reduction graph.

6 Different Strings, Same Graph

In this section, to simplify terminology, we consider equivalent legal strings as being identical. Let us consider now (two different) legal strings $u = pq\bar{p}q$ and $v = pqpq$. It turns out that they have the same reduction graph (up to isomorphism), thus $\mathcal{R}_u \approx \mathcal{R}_v$, and this graph is given in Fig. 22. The reason different legal strings may have the same reduction graph is that a reduction graph may have more than one set of merge edges—each one corresponding to a different legal string, cf. Sect. 5. Thus, there can be many MIC forms of genes (i.e. legal strings) obtaining the same MAC structure (i.e. reduction graph). In [2], it is shown how for a given legal string u we can obtain precisely the set of all legal strings having the same reduction graph (up to isomorphism). In fact, it turns out that this set is exactly the set of all legal strings

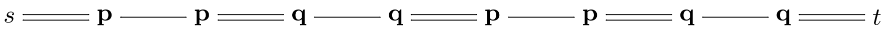


Fig. 22 The reduction graph of $pq\bar{p}q$ (and $pqpq$)

obtained by applying compositions of the following string rewriting rules. Hence, each 'orbit' of legal strings under these string rewriting rules is an equivalence class w.r.t. graph isomorphism of its reduction graphs.

First, we define for a string $u = x_1 x_2 \cdots x_n$ with $x_i \in \Pi$, its inversion $\bar{u} = \bar{x}_n \bar{x}_{n-1} \cdots \bar{x}_1$. For all $p, q \in \Pi$ with $\mathbf{p} \neq \mathbf{q}$,

- the *dual string positive rule* for p is defined by $\mathbf{dspr}_p(u_1 p u_2 p u_3) = u_1 p \bar{u}_2 p u_3$,
- the *dual string double rule* for p, q is defined by $\mathbf{dsdr}_{p,q}(u_1 p u_2 q u_3 \bar{p} u_4 \bar{q} u_5) = u_1 p u_4 q u_3 \bar{p} u_2 \bar{q} u_5$,

where $u_1, u_2, \ldots, u_5$ are arbitrary (possibly empty) strings over Π. The names of the two rules are due to their similarities with the string positive rule and string double rule defined later.

Let us take $u = pq\bar{p}q$ and $v = pqpq$ given earlier. Clearly, $\mathbf{dspr}_q(u) = v$, thus these legal strings indeed have a common reduction graph. If we consider again legal string $u = 2\bar{7}4735 3\bar{4}2656$ with its reduction graph given in Fig. 17, then we can apply $\mathbf{dsdr}_{4,\bar{5}}$ $\mathbf{dspr}_3$ to u and obtain legal string $2\bar{7}426\bar{5}3\bar{4}7356$. Hence, the reduction graph of this legal string is also given in Fig. 17.

7 Intermediate Gene Patterns

We have shown that the reduction graph is a representation of the MAC form of a gene (including the IESs) given a gene in its MIC form (formally and more generally, a legal string u). Here, we show that we can generalize the notion of reduction graph to allow for representations of any intermediate product. In such an intermediate product some pointers, represented as a subset D of $\mathrm{dom}(u)$, have not yet been used in recombination operations, while the other pointers, in $\mathrm{dom}(u) \backslash D$, have already been used in recombination operations. A reduction graph of u w.r.t. this set D, denoted by $\mathcal{R}_{u,D}$, represents such intermediate product. Hence, the reduction graph of u w.r.t. $\varnothing$ is (equivalent to) the reduction graph of u in Definition 1.

We informally define this graph through an example, and refer to [4] for a formal definition. The reduction graph $\mathcal{R}_{u,D}$ of our running example $u = 2\bar{7}47353\bar{4}2656$ w.r.t. $D = \{2, 4\}$ is given in Fig. 23.

We build the reduction graph as before, but simply ignore the pointers in D—they are put as strings on the reality edges which are now directed edges. For example, the string $\bar{4}2$ between (occurrences of) pointers 3 and 6 in u can be found as a label on an edge between vertices labeled by 3 and 6. The reverse edge is labeled by its inversion $\bar{2}4$. By rearranging the vertices, we obtain the graph in Fig. 24. It is assumed that a recombination operation during gene assembly cannot be undone. Thus, when a pointer is used in recombination, it is not considered a pointer anymore. Therefore, the intermediate product of our example is the linear sequence of

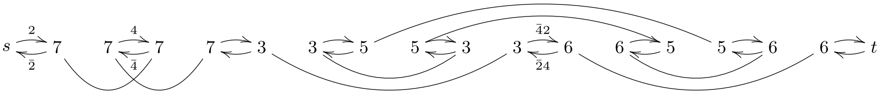


Fig. 23 The reduction graph of u w.r.t. $D = \{2, 4\}$

Fig. 24 The reduction graph of u w.r.t. $D = \{2, 4\}$

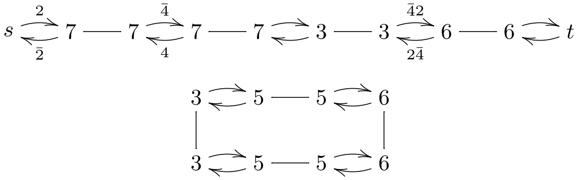
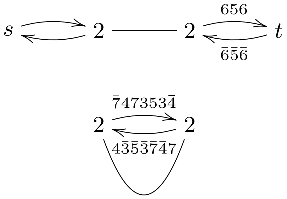


Fig. 25 The reduction graph of u w.r.t. $D = \{3, 4, 5, 6, 7\}$



pointers (legal string) $2\bar{4}\bar{4}2$, the label of the alternating path from s to t which we will denote by red(u, D), and a circular molecule. We moreover assume that gene assembly is intramolecular,[2] meaning that all recombination takes place on the linear DNA molecule. Thus, the generated circular molecules must not contain any pointers. Since the cyclic connected component in Fig. 24 has only empty strings as edge labels, this figure represents a valid intermediate product w.r.t. the required intramolecular nature of the process. It is easy to obtain an 'invalid intermediate product'; take, for example $D = \{3, 4, 5, 6, 7\}$ with its reduction graph given in Fig. 25. Hence, it is not possible to first recombine pointer 2, followed by recombination of the remaining pointers.

8 Gene Assembly Operations

Since we assume the process is intramolecular, there are restrictions on the use of recombination operations, as illustrated in the previous section by the existence of 'invalid' intermediate products. In this section, we recall constraints on the use of recombination through three types of operations to enforce that only valid intermediate products can be created; see also [7]. Recombination on a positive pointer is shown in Fig. 26—notice its similarities with the reversal operation in Fig. 1. Since

[2] The study of intermolecular models of gene assembly, which we do not consider here, has been initiated in [10]. There, it is possible for different components/molecules to recombine.

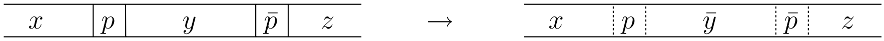


Fig. 26 Hairpin recombination

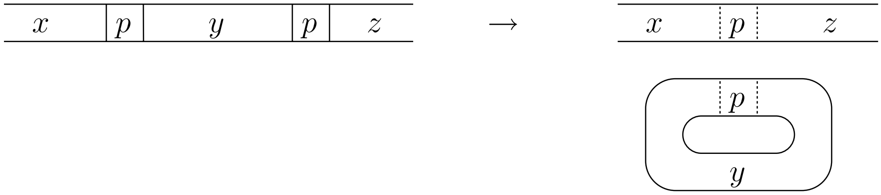


Fig. 27 Loop recombination

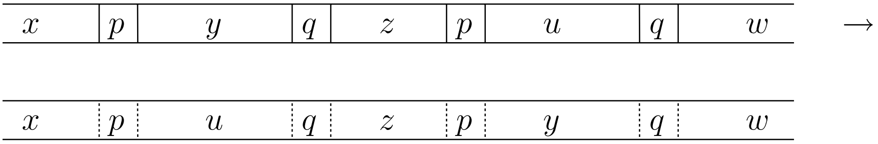


Fig. 28 Double-loop recombination

this operation is intramolecular, we allow recombination on positive pointers: We call this *hairpin recombination*. Recombination on a negative pointer is shown in Fig. 27. Since we require an intramolecular process, we allow this operation only if segment y in the figure is not relevant. That is, if y does not contain any pointers. We call such operation *loop recombination*. Finally, we may also allow recombination on a negative pointer p if it is done in parallel with recombination on another negative pointer q *overlapping* with p—meaning that the sequence is of the form $xpyqzpuqw$ for arbitrary sequences of pointers x, y, z, u, and w. This operation is given in Fig. 28 and is called *double-loop recombination*. Notice that this operation interchanges segments u and y. When a pointer is used in recombination, it is not considered a pointer anymore. Hence, in the figures, the 'pointers' after recombination are indicated by dotted lines.

We can formally model this process as follows. As usual, the sequences of pointers are described by legal strings. The three types of recombination operations can be defined now as three types of string rewriting rules, called *string pointer rules*, operating on legal strings. For all $p, q \in \Pi$ with $\mathbf{p} \neq \mathbf{q}$:

- the *string negative rule* for p is defined by $\mathbf{snr}_p(u_1 ppu_2) = u_1u_2$,
- the *string positive rule* for p is defined by $\mathbf{spr}_p(u_1 pu_2\bar{p}u_3) = u_1\bar{u}_2u_3$,
- the *string double rule* for p, q is defined by $\mathbf{sdr}_{p,q}(u_1 pu_2 qu_3 pu_4 qu_5) = u_1u_4u_3u_2u_5$,

where $u_1, u_2, \ldots, u_5$ are arbitrary strings over Π. Note that each of these rules is defined only on legal strings that satisfy the given form. For example, $\mathbf{snr}_2$ is not defined on legal string 2323. Notice the surprising similarities of the string positive and double rules with the dual string pointer rules defined in Sect. 6.

The string negative (positive, double, resp.) rule corresponds to the loop (hairpin, double-loop, resp.) recombination operation. Note that the fact that the molecular operations remove pointers is explicit in the string pointer rules. The model constituting the notions of legal string and the three types of string pointer rules is called string pointer reduction system (**SPRS**, for short) and was introduced in [6]. Also, monograph [7] describes this model in detail.

Example 1 We consider again our running example $u = 2\bar{7}473534\bar{4}2656$. We cannot apply $\mathbf{spr}_{\bar{5}}$ to u, but we can apply $\mathbf{spr}_4$ to u. We have $\mathbf{spr}_4(u) = 2\bar{7}\bar{3}\bar{5}\bar{3}\bar{7}2656$. Now, we can apply $\mathbf{spr}_{\bar{5}}$ to the obtained legal string.

For a given gene in its micronuclear form, a sequence of these molecular operations is *successful* if it transforms the pattern into its macronuclear form. Note that the macronuclear form of a gene corresponds to the empty string as then all pointers have been used in recombination operations. We now formally define the notion of successfulness. Let u and v be legal strings. A composition φ of reduction rules is called a *reduction of* u, if φ is applicable to (defined on) u. A *successful reduction* φ *of* u is a reduction of u such that $\varphi(u) = \lambda$ (the empty string is denoted by λ). Thus, e.g. $\mathbf{snr}_4\ \mathbf{spr}_{\bar{3}}\ \mathbf{spr}_2$ is a successful reduction of $23\bar{2}\bar{4}34$.

It is important to realize that for every non-empty legal string there is at least one reduction rule applicable. Indeed, every legal string for which no string positive rule and no string double rule is applicable must have only negative pointers and no overlapping pointers. Thus, all pointers are negative and nested and, therefore, there is an applicable string negative rule. Consequently, every legal string has a successful reduction (cf. Theorem 9.1 in [7]).

We define $\mathrm{dom}(\rho)$ for a reduction rule ρ as the set of pointers used by ρ, so $\mathrm{dom}(\mathbf{snr}_p) = \mathrm{dom}(\mathbf{spr}_p) = \{\mathbf{p}\}$ and $\mathrm{dom}(\mathbf{sdr}_{p,q}) = \{\mathbf{p}, \mathbf{q}\}$ for $p, q \in \Pi$. For a composition $\varphi = \rho_n \cdots \rho_2\ \rho_1$ of reduction rules $\rho_1, \rho_2, \ldots, \rho_n$, we define $\mathrm{dom}(\varphi) = \mathrm{dom}(\rho_1) \cup \mathrm{dom}(\rho_2) \cup \cdots \cup \mathrm{dom}(\rho_n)$. Moreover, we define $\mathrm{snrdom}(\varphi) = \mathrm{dom}(\rho_{i_1}) \cup \cdots \cup \mathrm{dom}(\rho_{i_k})$, where $\rho_{i_1}, \ldots, \rho_{i_k}$ are (all) the **snr** rules of φ.

9 Cyclic Components

Recall from Sect. 4 that the cyclic connected components of the reduction graph represent the cyclic molecules that are excised from the DNA molecule. A quick look at the three types of recombination operations of the previous section shows that each such molecule must have been created due to loop recombination. Since loop recombination is formally described by string negative rules, we have the following result.

Theorem 1 *Let N be the number of cyclic connected components in the reduction graph of legal string u. Then every successful reduction of u has exactly N string negative rules.*

Example 2 Since $\mathcal{R}_u$ in Fig. 17 has three cyclic components, by Theorem 1, every successful reduction φ of u has exactly three string negative rules. For example, $\varphi = \mathbf{snr}_2\ \mathbf{snr}_{\bar{4}}\ \mathbf{spr}_{\bar{7}}\ \mathbf{snr}_6\ \mathbf{sdr}_{3,5}$ is a successful reduction of u. Indeed, φ has exactly three string negative rules. Alternatively, $\mathbf{snr}_6\ \mathbf{snr}_3\ \mathbf{snr}_7\ \mathbf{spr}_2\ \mathbf{spr}_{\bar{5}}\ \mathbf{spr}_4$ is also a successful reduction of u, with a different number of (**spr** and **sdr**) operations.

It turns out that the reduction graph also allows for determining *on which (sets of) pointers* the string negative rules can be applied—formally these are the sets $\mathrm{snrdom}(\varphi)$ for all successful reductions φ of u. Surprisingly, the pointer-component graph is of use again. In fact, it was originally defined for this purpose; see [3].

Let us first denote $\mathcal{PC}_u|_D$ as the graph obtained from $\mathcal{PC}_u$ by removing the edges outside D.

Theorem 2 *Let u be a legal string, and let $D \subseteq \mathrm{dom}(u)$. There is a successful reduction φ of u with $\mathrm{snrdom}(\varphi) = D$ iff $\mathcal{PC}_u|_D$ is a tree.*

Example 3 In our running example, we see that $D = \{2, 3, 6\}$ induces a (spanning) tree of $\mathcal{PC}_u$ given in Fig. 21. Therefore, there is a successful reduction φ of u with $\mathrm{snrdom}(\varphi) = D$. Indeed, we have $\varphi' = \mathbf{sdr}_{\bar{4},5}\ \mathbf{spr}_{\bar{7}}(u) = 226336$. It is clear that we can extend φ' to a successful reduction which applies string negative rules on 2, 3, and 6. Notice that here $\mathbf{snr}_3$ must be applied *before* $\mathbf{snr}_6$.

In [2], it is shown that the possible orders in which the string negative rules can be applied is also deducible from $\mathcal{PC}_u$ by considering rooted trees.

We conclude this section with a result concerning intermediate gene patterns and the generalized notion of reduction graph of Sect. 7. In this section, we have already seen that for reductions φ of u with $D = \mathrm{dom}(u)\backslash\mathrm{dom}(\varphi)$, we have $\varphi(u) = \mathrm{red}(u, D)$ (the label of the alternating path from s to t in $\mathcal{R}_{u,D}$). As a consequence, reductions φ_1 and φ_2 of a legal string u with the same domain have the same effect: $\varphi_1(u) = \varphi_2(u)$. This puts Theorem 4.8 of [9] into a new perspective: the fact that a reordering of operations yields the same result is independent of the parallelism of the operations. The next result shows for which $D \subseteq \mathrm{dom}(u)$ there exists a reduction φ with $D = \mathrm{dom}(u)\backslash\mathrm{dom}(\varphi)$.

Theorem 3 *Let u be a legal string, and let $D \subseteq \mathrm{dom}(u)$. There is a reduction φ of u with $\mathrm{dom}(\varphi) = \mathrm{dom}(u)\backslash D$ iff $\mathrm{red}(u, D)$ is legal with domain D (or equivalently, the label of every reality edge of a cyclic connected component is λ).*

It should be stressed that the results of this section depend mostly on the following two properties of our string rewriting system **SPRS** under consideration: (1) only one type of rule 'creates' cyclic components, and (2) every legal string has a successful reduction. Most results carry over to other string rewriting systems having these two properties.

10 Discussion

The concept of breakpoint graph from the theory of sorting by reversal has many uses for gene assembly in ciliates. While a legal string is a representation of the micronuclear form of a gene (the begin product of gene assembly), the reduction graph of that legal string is a representation of the macronuclear form of that same gene (the end product of gene assembly). The possible forms of reduction graphs are characterized, as well as the sets of legal strings that obtain a common reduction graph. Also, loop recombination is well explained by the reduction graph, just as the reversal operation (similar to hairpin recombination) is explained by the breakpoint graph. It remains an open problem to characterize hairpin and double-loop recombination.

References

1. Bergeron A, Mixtacki J, Stoye J (2005) On sorting by translocations. In: Miyano S et al (eds) RECOMB. Lecture notes in computer science, vol 3500. Springer, Berlin, pp 615–629
2. Brijder R, Hoogeboom HJ (2008) The fibers and range of reduction graphs in ciliates. Acta Inform 45:383–402. doi:10.1007/s00236-008-0074-3
3. Brijder R, Hoogeboom HJ, Muskulus M (2008) Strategies of loop recombination in ciliates. Discrete Appl Math 156:1736–1753. doi:10.1016/j.dam.2007.08.032
4. Brijder R, Hoogeboom HJ, Rozenberg G (2006) Reducibility of gene patterns in ciliates using the breakpoint graph. Theor Comput Sci 356:26–45
5. Cavalcanti ARO, Clarke TH, Landweber LF (2005) MDS_IES_DB: a database of macronuclear and micronuclear genes in spirotrichous ciliates. Nucleic Acids Res 33:D396–D398
6. Ehrenfeucht A, Harju T, Petre I, Prescott DM, Rozenberg G (2003) Formal systems for gene assembly in ciliates. Theor Comput Sci 292:199–219
7. Ehrenfeucht A, Harju T, Petre I, Prescott DM, Rozenberg G (2004) Computation in living cells—gene assembly in ciliates. Springer, Berlin
8. Hannenhalli S, Pevzner PA (1999) Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. J ACM 46(1):1–27
9. Harju T, Li C, Petre I, Rozenberg G (2006) Parallelism in gene assembly. Nat Comput 5(2):203–223
10. Landweber LF, Kari L (1999) The evolution of cellular computing: nature's solution to a computational problem. Biosystems 52:2–13
11. Pevzner PA (2000) Computational molecular biology: an algorithmic approach. MIT Press, Cambridge
12. Prescott DM, DuBois M (1996) Internal eliminated segments (IESs) of oxytrichidae. J Euk Microbiol 43:432–441
13. Setubal JC, Meidanis J (1997) Introduction to computational molecular biology. PWS, Boston

Template-Guided Recombination: From Theory to Laboratory

Mark Daley and Michael Domaratzki

Abstract Template-guided recombination (TGR) is a model for the rearrangement of genomic DNA that takes place in some ciliated protozoa. Originally proposed as a formal model, TGR has been investigated both as a realistic model for genome rearrangement in ciliates and, due to interest in the potential of ciliates as "in vivo computers", in terms of its computational power. TGR was put forward as a biological hypothesis that certain types of DNA rearrangements in ciliates are primarily controlled by a process of template-matching, where new genes are generated by using old genes as templates. Most significantly, it has recently been experimentally established that gene rearrangement in the stichotrichous ciliate *Oxytricha trifallax* (*Sterkiella histriomuscorum*) proceeds in a template-guided fashion. This survey describes recent work on TGR as a biological process and the computational properties of the formal model of TGR.

1 Introduction

Template-guided recombination (TGR) is a model for the rearrangement of genomic DNA that takes place in some ciliated protozoa. Originally proposed as a formal model, TGR has been investigated both as a realistic model for genome rearrangement in ciliates and, due to interest in the potential of ciliates as "in vivo computers", in terms of its computational power.

Originally introduced by Prescott et al. [16], template-guided recombination was put forward as a biological hypothesis that DNA rearrangement in ciliates is primarily controlled by a process of template-matching, viz., new genes are descrambled by using old, already descrambled, genes as templates. Most significantly, it has recently been experimentally established by Nowacki et al. [15] that gene descrambling in the stichotrichous ciliate *Oxytricha trifallax* (*Sterkiella histriomuscorum*) proceeds in a template-guided fashion.

M. Daley (✉)
Departments of Biology and Computer Science, University of Western Ontario, London, ON, N6A 5B7, Canada
e-mail: daley@csd.uwo.ca

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_8, © Springer-Verlag Berlin Heidelberg 2009

1.1 Relationship to Natural Computing

Ciliate DNA rearrangement and its computational properties are of particular interest to researchers in natural computing. Natural computing studies the use of natural media, such as molecules, particles (in the context of quantum computation), or light to perform computation. In this field, the evolution of a mechanism in ciliates which performs rearrangement, a basic form of computing, has received much attention in the literature. For instance, the complexity of solving particular problems in certain formal models of DNA rearrangement in ciliates has recently been examined by Alhazov et al. [1]. We note in passing that such investigation for particular computational problems (e.g., NP-complete problems) has not yet been undertaken in the TGR model.

1.2 Outline

In this survey, we examine the computational and experimental research on TGR.

The remainder of this survey is structured as follows. In Sect. 2, we briefly describe the process of conjugation in ciliates and the role of DNA rearrangement in that process.

In Sect. 3, we present the formal biological model for template-guided recombination. In Sect. 4, we review the basics of formal language theory and present the formal language theoretic formulation of TGR. In Sect. 5, we note the relationship of TGR to splicing systems, in Sect. 6, we investigate the computational power of TGR, Sect. 7 addresses the problem of deciding the equivalence of template sets, and Sect. 8 introduces the notions of covers and scaffolds. We finish with Sect. 9 describing recent experiments verifying the template-guided model.

2 Conjugation in Ciliates

The ciliated protozoa are an ancient group of single-celled eukaryotes which possess a unique feature: almost all ciliates have two types of nuclei. Typical eukaryotes (e.g., humans) have a single nucleus in each cell that performs somatic day-to-day "genetic housekeeping" functions, such as transcribing genes, as well as serving as the germline repository of genetic information to pass on to the next generation. Ciliates, however, have developed a unique division of labor: a macronucleus (MAC) serves as the somatic nucleus while a much smaller, functionally inert, micronucleus (MIC) serves as germline storage.

While ciliates reproduce through asexual fission, they also engage in sexual activity known as conjugation. During conjugation no offspring are produced; instead, two ciliates exchange genetic material and each then leaves the conjugation process

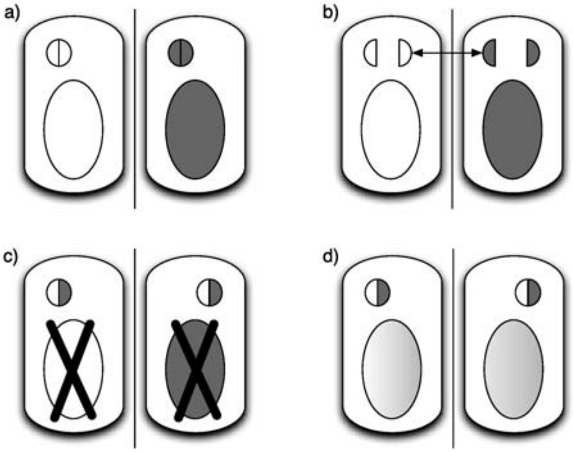


Fig. 1 A simplified view of conjugation. Two ciliates (**a**) meiotically generate and exchange haploid MICs (**b**), form a new MIC and destroy the old MAC (**c**) and finally generate a new MAC (**d**)

as a genetically "new" organism. The precise details of conjugation vary significantly across ciliate species, but the basics of the process, outlined here and summarized in Fig. 1, are highly conserved.

After initiating conjugation, the MICs of the two paired ciliates undergo the process of meiosis, creating haploid[1] micronuclei. One of these haploid MICs, the stationary MIC, remains in the parent cell, while the other "migratory" haploid MIC is transferred to the conjugate partner. Upon receiving the partner's migratory haploid MIC, each cell combines the migratory and stationary haploid MICs to form a new, diploid MIC.

At this point, the cells then begin the process of destroying their old MACs and generating new MACs from their new MICs. The process of macro-nuclear development is highly variable in different species, but in the stichotrichous ciliates it involves extreme rearrangements of the micro-nuclear DNA. Up to 95% of the micro-nuclear DNA is eliminated from the macronucleus and the remaining DNA is then heavily amplified. Further complexity is introduced by the fact that the macro-nuclear form of genes may be significantly different than the micro-nuclear form.

When comparing a MIC gene to the functional version of that gene in the MAC, one notices that two types of transformation have taken place. Many segments of the MIC gene, called Internal Eliminated Sequences (or IESs) have been completely deleted from the macro-nuclear version of the gene. This process of IES elimination has been observed in all ciliate species which have been subject to genetic study.

The stichotrichous ciliates have the even more remarkable property that some genes have had large sections completely re-ordered between the MIC and MAC versions of the gene. That is, if a gene has the form 1–2–3–4–5 in the MAC, the associated MIC version may have the form: 3–5–1–2–4; see Fig. 2 for a schematic depiction. Each MIC gene can be divided into three types of regions: IESs, macro-nuclear destined sequences (or MDSs) and so-called pointer sequences which flank

[1] The ciliate MIC is diploid—it possess exactly two copies of each chromosome, as is the case, e.g., in humans.

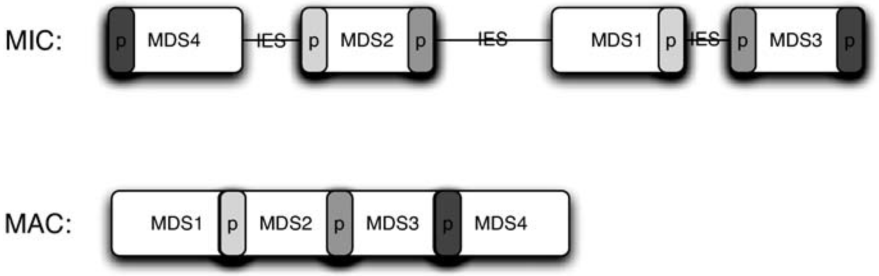


Fig. 2 A schematic representation of the MIC and MAC versions of a hypothetical stichotrich gene

each MDS. The pointer sequence at the far right of MDS n is identical to the pointer sequence at the left of MDS $n+1$, when MDSs are labeled according to the order in which they occur in the MAC.

While the "pointer" sequences are necessary to enable recombination between MDSs which must be made adjacent, they are not "pointers" in the sense that they are able to guide descrambling of MDSs. Indeed, pointer sequences are far too short to uniquely guide descrambling. Template-guided recombination was thus proposed as a model to explain how "DNA descramblings" of this apparent complexity might take place in the cell. The central hypothesis is that fragments of descrambled genes from the old macronucleus serve as templates to guide the reconstruction of a new macro-nuclear gene from the micro-nuclear version of the gene.

3 Formal Model

We now describe the formal model for TGR proposed by Prescott et al. [16]. The model of TGR involves three DNA segments, which we denote X, Y, and T. The segments X and Y represent portions of ciliate DNA which are being rearranged, while T represents the template. The role of the template is to arrange the pointers. As noted in Sect. 2, Prescott et al. [16] hypothesized that the origin of the templates for rearrangement is genetic material from the maternal micronucleus.

For TGR to occur, we require the presence of specific regions, including a pointer region, in X, Y, and T. From $3'$ to $5'$, we require the presence of blocks of DNA of the following form (here, $\overline{a}$ represents the reverse complement of a):

$$\begin{aligned} X&: \quad \alpha\beta_1\beta_2\delta, \\ T&: \quad \overline{\gamma}\overline{\beta_2}\,\overline{\beta_1}\overline{\alpha}, \\ Y&: \quad \varepsilon\beta_1\beta_2\gamma. \end{aligned}$$

Further, we require that the following constraints be satisfied:

(a) δ is not complementary to $\overline{\gamma}$ and ε is not complementary to $\overline{\alpha}$.

(b) For some constants C, D, and E, $|\alpha|, |\gamma| \geq C$ and $D \leq |\beta_1\beta_2| \leq E$. (Here, $|\alpha|$ is the length of α.)

No other constraints, for example, on the length of δ or ε are made. The region $\beta_1\beta_2$ represents the pointer regions used to align the proper MDSs, while α and γ represent additional material required, in the model, to uniquely resolve the pointers. Condition (b) above is motivated by the observed lengths of the pointer sequences in ciliates. Typically, the length of $\beta_1\beta_2$ is between two and twenty nucleotides (see, e.g., Chang et al. [3]).

With these requirements met, TGR occurs with the alignment of X,Y, and T. This alignment of the molecules prior to the recombination is depicted schematically in Fig. 3.

Next, recombination occurs by the destruction of hydrogen bonds in the $\alpha\beta_1$ block of X, the $\alpha\beta_1\beta_2\gamma$ block of T and the $\beta_2\gamma$ block of Y. The formation of hydrogen bonds occurs between the shaded regions depicted in Fig. 4; note that hydrogen bonds form between, e.g., both strands of the double helix of X and a corresponding strand of the double helix of T in the $\alpha\beta_1$ block.

After the formation of the hydrogen bonds, the model stipulates that eight cuts are made to the phosphate backbones of X, T, and Y. These are also depicted in Fig. 4: the pair of scissors represent the cuts to either the upper or lower stands in the double helix. These cuts result in four separate molecules, depicted in Fig. 5. There are six nicks that need to be repaired in the resulting molecules.

A crucial part of the model is that after completion the template can be repaired by ligation of three nicks. Thus, the operation can be repeated using the same template molecule, and a large supply of templates it not necessary.

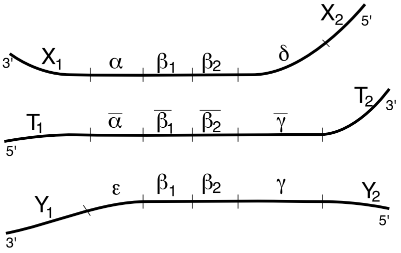


Fig. 3 Depiction of the preconditions for TGR. Each strand is represented schematically as a *single line* (i.e., the complementary strands are not shown). *Tick marks* indicate divisions between the labeled blocks

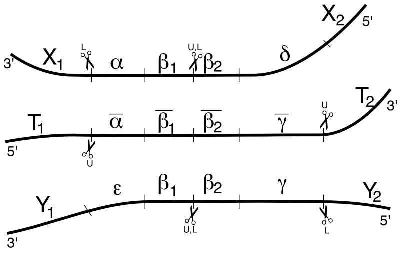


Fig. 4 Formation of new hydrogen bonds and cuts to the phosphate backbones during TGR. At each cut point (indicated by a *pair of scissors*), U and L indicate whether the upper or lower strand is cut

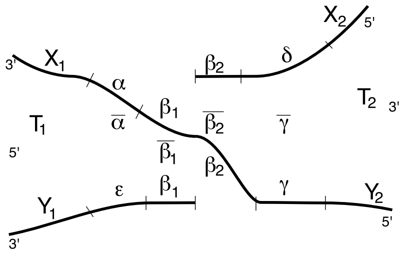


Fig. 5 Four molecules resulting from the eight phosphate backbone cuts. The template strand is depicted in *grey* for easier viewing of the molecules at the intersection

Another important observation of the formal model is its irreversibility. None of the four molecular products can again serve as a template for a TGR involving any pair of the resultant molecules.

3.1 Origin of IESs

The TGR model is a guiding assumption in the recent, related development of a hypothesis for the advent of IESs and scrambled micro-nuclear DNA in ciliates. The proposed model, described by Ehrenfeucht et al. [9], is motivated in part by experimental observations that ciliates are highly resistant to exposure to radiation and desiccation.

The model describes how repairs to broken DNA strands could give rise to pointers, IESs and scrambled MIC genes. In the model, duplicated regions (the pointers) as well as additional material (the IESs) are introduced during the repair process. The major requirement in the repair model is the presence of two undamaged copies of the DNA to repair one damaged copy.

With this hypothesis, multiple breaks in coiled DNA can result in the types of scrambling and duplicated pointers and intervening IESs observed in ciliates [9]. In particular, if the multiple breaks in a coiled are rejoined in the correct ways by the repair process, the only change is the addition of IESs and an orthodox form of the micro-nuclear gene is obtained. If multiple breaks in a coil are rejoined to make new connections, then we obtain scrambled micro-nuclear genes.

We note that the proposed repair mechanism acts as an inverse to TGR by duplicating pointer regions and introducing IESs. This is similar to the concept of the inverse of an operation in language theoretic terms, introduced by Kari [13] (see Sect. 4.1 for an introduction to formal language theory). The concept of an inverse to TGR in an intra-molecular setting has been examined in the context of language equations by Daley et al. [4].

3.2 RNA Template Model

Recently, Angeleska et al. [2] have reconsidered the TGR model proposed by Prescott et al. The main difference between the two models is the substitution of

RNA templates (either double-stranded or single-stranded) for DNA templates in the original model. The authors note several distinctions between the DNA template model and the RNA template model [2]. In particular, the RNA model requires only four cuts to the sugar-phosphate backbone of the micro-nuclear RNA, as opposed to a total of eight for the DNA template model, including cuts to both the micro-nuclear DNA and the DNA template.

The RNA template model is also feasible as it does not introduce molecules containing a mixture of RNA and DNA on a single backbone and all of the recombined material consists of micro-nuclear DNA only [2]. We note (see Fig. 4) that the DNA template models, while not creating DNA-RNA hybrids, does introduce a portion of the template into the rearranged DNA.

Another observation [2] is that the RNA model produces circular strands of DNA as a byproduct of the TGR operation. This is consistent with experimental research showing circular strands of DNA (consisting of IESs) present after the recombination in some ciliates (see, e.g., Jaraczewski and Jahn [12]).

While the operation has several differences from the model of Prescott et al., as the authors note it has no effect on the inter-molecular TGR operation as a formal language theory operation (described below in Sect. 4.2). We examine the experimental evidence for proposing RNA templates in Sect. 9.

4 Formal Language Model

In this section, we describe the formal language theoretic model for TGR. We first review the necessary formal language theory.

4.1 Formal Language Theory Basics

For additional background on formal languages, see Rozenberg and Salomaa [18]. Let Σ be a finite set of symbols, called *letters*; we call Σ an *alphabet*. Then Σ^* is the set of all finite sequences of letters from Σ, which are called *words*. The empty word ε is the empty sequence of letters. We denote by Σ^+ the set of non-empty words over Σ, i.e., $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. The length of a word w is denoted by $|w|$. A *language* L is any subset of Σ^*.

Let Δ, Σ are alphabets. A function $h : \Delta \to \Sigma^*$ is called a *morphism*. It is extended to $h : \Delta^* \to \Sigma^*$ by the rule $h(xy) = h(x)h(y)$ for all $x, y \in \Delta^*$. Given a language $L \subseteq \Delta^*$, $h(L) = \{h(x) : x \in L\}$. Further, if $L' \subseteq \Sigma^*$, then $h^{-1}(L') = \{x : h(x) \in L'\}$.

We assume the reader is familiar with the classes of finite, regular, context-free recursive, and recursively enumerable (r.e.) languages. In particular, a language is *regular* (resp., *context-free*, *recursive*, *recursively enumerable*) if it is accepted (resp., generated, accepted, accepted) by a deterministic finite automaton (resp., context-free grammar, Turing machine which halts on all inputs, Turing machine).

The classes of finite, regular, context-free, recursive, and r.e. languages form a strict hierarchy of inclusions. We denote the class of finite, regular, context-free, recursive, and recursively enumerable languages, respectively, by FIN,REG,CF,REC,RE.

Recall that a class of languages $\mathcal{L}$ is a cone if it is closed under morphism, inverse morphism, and intersection with regular languages. If $\mathcal{L}$ is also closed under concatenation and Kleene closure, then $\mathcal{L}$ is a *full abstract family of languages* (full AFL). For any class of languages $\mathcal{L}$, we denote by $\mathcal{L}_0$ the class of ε-free languages in $\mathcal{L}$, i.e., $\mathcal{L}_0 = \{L \in \mathcal{L} : \varepsilon \notin \mathcal{L}\}$. In this respect, we will mostly be interested in REG_0, the class of ε-free regular languages.

4.2 Formal Language Model for TGR

We now give the formal language theoretic definition of TGR, which was proposed by Prescott et al. [16] and first studied as a formal operation by Daley and McQuillan [5]. If $n_1, n_2 \geq 1$ and $x, y, z, t \in \Sigma^*$ are words, we denote by $(x, y) \vdash_{t,n_1,n_2} z$ the fact that we can write

$$\begin{aligned} x &= u_1 \alpha \beta v_1, \\ y &= v_2 \beta \gamma u_2, \\ z &= u_1 \alpha \beta \gamma u_2, \\ t &= \alpha \beta \gamma, \end{aligned}$$

with $\alpha, \beta, \gamma, u_1, u_2, v_1, v_2 \in \Sigma^*$, $|\alpha|, |\gamma| \geq n_1$ and $|\beta| = n_2$. The word t is the template.

The requirements for applying $\vdash_{t,n_1,n_2}$ are depicted in Fig. 6. If these requirements are met, then the constructed word is $u_1 \alpha \beta \gamma u_2$.

If $T, L \subseteq \Sigma^*$ are languages, then $\pitchfork_{T,n_1,n_2}(L)$ is defined by

$$\pitchfork_{T,n_1,n_2}(L) = \big\{ z : \exists x, y \in L,\ t \in T \text{ such that } (x, y) \vdash_{t,n_1,n_2} z \big\}.$$

We use the notation $\pitchfork_T(L)$ if n_1, n_2 are understood or unimportant. The language T is called the *set of templates*.

We note here the main differences between the formal model presented in Sect. 3 and the relation $\vdash_{t,n_1,n_2}$.

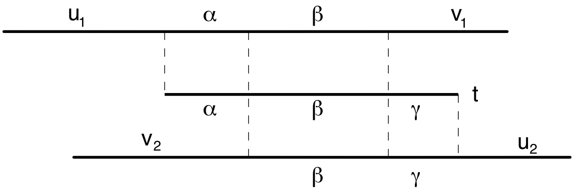


Fig. 6 Requirements for an application of TGR

First, the original model allows for the region $\alpha\beta\gamma$ to be a sub-word of longer templates (i.e., in Fig. 3, T_1 and T_2 can be non-empty). However, as most classes of languages are closed under the sub-word operation, this restriction typically does not affect the complexity of the results obtained.

Another difference is that the formal language operation is only concerned with the product $u_1\alpha\beta\gamma u_2$ of TGR. The products $\beta_1 v_1$ and $\beta_2 v_2$ (for an appropriate factorization of β as $\beta = \beta_1\beta_2$) are ignored. This agrees with our focus on the re-arrangement process—the alignment of pointers and the elimination of IESs—rather than all molecular products. In particular, we ignore the segmented IESs in the formal language model.

We also note as well the difference between the length requirements of the model and the language operations. In particular, the formal language operation insists that $|\beta| = n_1$ for some constant n_1. However, as shown by Daley and McQuillan [5], this is equivalent to the condition that $|\beta| \geq n_1$ for the same constant n_1.

Finally, we note that as a simplification, the formal language theoretic model removes the condition on non-complementarity between, e.g., v_1 and γ.

4.3 Iterated TGR

Daley and McQuillan [5] have also introduced and extensively examined the iterated version of TGR. As a model of DNA rearrangement in ciliates, iterated TGR is much more realistic than a single application of TGR. Cellular biochemistry is an inherently stochastic process involving the interaction of enzymes, substrates, and catalysts in solution and it is thus natural to consider iterated operations in which non-determinism is used to model the underlying stochastic system.

Iterated TGR is defined in the natural way: let $\pitchfork^0_{T,n_1,n_2}(L) = L$ and for all $i \geq 1$, let

$$\pitchfork^i_{T,n_1,n_2}(L) = \pitchfork^{i-1}_{T,n_1,n_2}(L) \cup \pitchfork_{T,n_1,n_2}\left(\pitchfork^{i-1}_{T,n_1,n_2}(L)\right).$$

Then we also define $\pitchfork^*_{T,n_1,n_2}(L)$ as

$$\pitchfork^*_{T,n_1,n_2}(L) = \bigcup_{i\geq 0} \pitchfork^i_{T,n_1,n_2}(L).$$

4.4 Classes of Languages Defined by TGR

In order to discuss the closure properties of classes of languages, we introduce some additional notation. Let $\mathcal{L}, \mathcal{T}$ be classes of languages and $n_1, n_2 \geq 1$.

$$\pitchfork_{\mathcal{T},n_1,n_2}(\mathcal{L}) = \left\{\pitchfork_{T,n_1,n_2}(L) : T \in \mathcal{T},\ L \in \mathcal{L}\right\},$$
$$\pitchfork^*_{\mathcal{T},n_1,n_2}(\mathcal{L}) = \left\{\pitchfork^*_{T,n_1,n_2}(L) : T \in \mathcal{T},\ L \in \mathcal{L}\right\},$$

$$\pitchfork_{\mathcal{T}}(\mathcal{L}) = \left\{\pitchfork_{T,n_1,n_2}(L) : T \in \mathcal{T},\ L \in \mathcal{L},\ n_1, n_2 \geq 1\right\},$$
$$\pitchfork^*_{\mathcal{T}}(\mathcal{L}) = \left\{\pitchfork^*_{T,n_1,n_2}(L) : T \in \mathcal{T},\ L \in \mathcal{L},\ n_1, n_2 \geq 1\right\}.$$

4.5 Deletion Contexts in TGR

We note that Daley and McQuillan [6] have investigated the computational power of TGR in which templates specify additional constraints on the deleted regions: In particular, if $t \in \Sigma^*\#\Sigma^*\#\Sigma^*$, let $\vdash^{(d)}_{t,n_1,n_2}$ be the binary relation defined by $(x, y) \vdash^{(d)}_{t,n_1,n_2} z$ if we can write

$$x = u_1\alpha\beta v_1 v'_1,$$
$$y = v'_2 v_2\beta\gamma u_2,$$
$$z = u_1\alpha\beta\gamma u_2,$$
$$t = v_2\#\alpha\beta\gamma\# v_1$$

with $\alpha, \beta, \gamma, u_1, u_2, v'_1, v'_2, v_1, v_2 \in \Sigma^*$, $|\alpha|, |\gamma| \geq n_1$ and $|\beta| = n_2$.

This operation is motivated by the investigation of the necessary modifications to the TGR operation in order to significantly modify its computational power, and not by observed biological properties. Daley and McQuillan [6] investigate the closure properties of the operation defined by TGR with deletion contexts, and languages which are closed under TGR with deletion contexts.

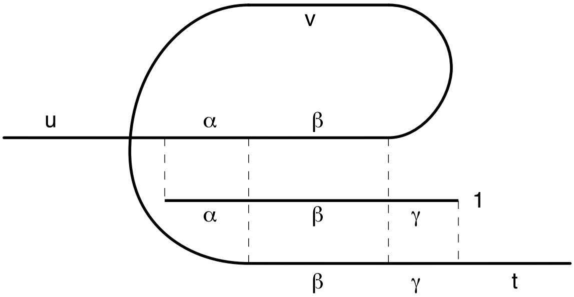
4.6 Intra-molecular TGR

As shown in Fig. 7 and as noted in the original paper of Prescott et al. [16], TGR does not need to be applied to two distinct strands (or words in the formal language theoretic model). Indeed, it can be applied to two regions of the same linear strand. The resulting operation can be seen as a templated version of the *ld* (loop delete) operation, which is an earlier formal model of ciliate DNA rearrangement. Please see, e.g., Ehrenfeucht et al. [11] for a description of the *ld* operation, and the associated *hi* (hairpin inversion) and *dlad* (double loop and delete) operations.

This leads to the following definition of intra-molecular TGR as a formal language theoretic operation [4]. Let $x, t, y \in \Sigma^*$ and $n_1, n_2 \geq 1$. We say that $x \vdash^{(\ell)}_{t,n_1,n_2} y$ if there exist $u, v, w, \alpha, \beta, \gamma \in \Sigma^*$ with $|\alpha|, |\gamma| \geq n_1$ and $|\beta| = n_2$ such that we can write

$$x = u\alpha\beta w\beta\gamma v,$$
$$y = u\alpha\beta\gamma v,$$
$$t = \alpha\beta\gamma.$$

Fig. 7 Intra-molecular TGR



If $T, L \subseteq \Sigma^*$ are languages, then $\pitchfork^{(\ell)}_{T,n_1,n_2}(L)$ is defined by

$$\pitchfork^{(\ell)}_{T,n_1,n_2}(L) = \left\{z : \exists x \in L,\ t \in T \text{ such that } x \vdash^{(\ell)}_{t,n_1,n_2} z\right\}.$$

We use the notation $\pitchfork^{(\ell)}_{T}(L)$ if n_1, n_2 are understood or unimportant.

Angeleska et al. [2] have independently described a templated version of the ld operation, as well as the hi and $dlad$ operations, using their refined TGR model. Their description of hi is especially instructive about the differences between the Prescott et al. [16] model and their model: in the Prescott et al. [16] model, the hairpin language operation involves a loss of some materials in general (see Daley et al. [4] for a description of this operation). However, no loss of material occurs in the hi language operation described using the model of Angeleska et al.

5 Relation to Splicing Systems

Daley and McQuillan [5] have examined the relationship between TGR and splicing systems, which we briefly recall here. We refer the reader to Păun et al. [17] for further results on splicing systems.

An *H scheme* is a pair $\sigma = (\Sigma, R)$ where Σ is an alphabet and $R \subseteq \Sigma^*\#\Sigma^*\$\Sigma^*\#\Sigma^*$ is a set of splicing rules where \$, # are not elements of Σ.

For a rule $r \in R$, we define the relation $(x, y) \models_r z$ if

$$r = u_1\#u_2\$u_3\#u_4,$$

$$x = x_1u_1u_2x_2,$$

$$y = y_1u_3u_4y_2,$$

$$z = x_1u_1u_4y_2,$$

for some $u_1, u_2, u_3, u_4, y_1, y_2, x_1, x_2 \in \Sigma^*$.

For a language $L \subseteq \Sigma^*$ and an H scheme $\sigma = (\Sigma, R)$, we define

$$\sigma(L) = \left\{z \in \Sigma^* : \exists x, y \in L,\ r \in R \text{ such that } (x, y) \models_r z\right\}.$$

Further, for classes of languages $\mathcal{L}, \mathcal{R}$, let

$$S(\mathcal{L}, \mathcal{R}) = \big\{\sigma(L) : L \in \mathcal{L},\ \sigma = (\Sigma, R) \text{ and } R \in \mathcal{R}\big\}.$$

Let $\sigma = (\Sigma, R)$ be an H scheme. For iterated splicing, let $\sigma^0(L) = L$ and $\sigma^i(L)$ be defined by

$$\sigma^i(L) = \sigma^{i-1}(L) \cup \sigma\big(\sigma^{i-1}(L)\big)$$

for all $i \geq 1$. Finally, as expected,

$$\sigma^*(L) = \bigcup_{i \geq 0} \sigma^i(L).$$

For classes of languages $\mathcal{L}, \mathcal{R}$, let

$$H(\mathcal{L}, \mathcal{R}) = \big\{\sigma^*(L) : L \in \mathcal{L},\ \sigma = (\Sigma, R) \text{ and } R \in \mathcal{R}\big\}.$$

The following results are due to Daley and McQuillan [5].

Lemma 1 *Let $\mathcal{L}$ be a trio or $\mathcal{L} = \text{FIN}$. Then for all $T \in \mathcal{L}$ and all $n_1, n_2 \geq 1$, there exists an H scheme $\sigma = (\Sigma, R)$ where $R \in \mathcal{L}$ such that for all $L \subseteq \Sigma^*$, $\pitchfork_{T,n_1,n_2}(L) = \sigma(L)$.*

The proof of Lemma 1 shows that in order to simulate a finite set of templates T, the resulting splicing system requires $\sum_{t \in T}(|t| - 2n_1 - n_2 + 1)$ rules. Daley and McQuillan show that this bound is tight for single-application TGR, provided that the size of the alphabet is linear in $|T|$. The authors note that proving the same bound is tight for fixed sized alphabets is still open, as is the problem for iterated TGR.

The following corollary is immediate from Lemma 1.

Corollary 1 *For all $n_1, n_2 \geq 1$, all classes of languages $\mathcal{L}$ and all classes $\mathcal{T}$ such that $\mathcal{T}$ is a trio (or $\mathcal{T} = \text{FIN}$), the following inclusions hold:*

$$\pitchfork_{\mathcal{T},n_1,n_2}(\mathcal{L}) \subseteq S(\mathcal{L}, \mathcal{T}),$$
$$\pitchfork^*_{\mathcal{T},n_1,n_2}(\mathcal{L}) \subseteq H(\mathcal{L}, \mathcal{T}).$$

In some cases, the inclusions in Corollary 1 are actually equalities.

Lemma 2 *For all full AFLs $\mathcal{L}$ and all $n_1, n_2 \geq 1$ the following equalities hold:*

$$\mathcal{L} = \pitchfork^*_{\text{FIN},n_1,n_2}(\mathcal{L}) = H(\mathcal{L}, \text{FIN}).$$

However, some inclusions are strict; in what follows, $\subset$ denotes strict inclusion.

Lemma 3 *Let $\mathcal{L}$ be a class of languages such that* $\text{FIN} \subseteq \mathcal{L} \subset \text{RE}$ *and $\mathcal{L}$ is closed under intersection with regular languages. Let $\mathcal{R}$ be a class of languages such that* $\text{REG}_0 \subseteq \mathcal{R}$. *Then*

$$\pitchfork^*_{\text{RE}}(\mathcal{L}) \subset H(\mathcal{L}, \mathcal{R}) = \text{RE}.$$

For finite languages and finite sets of templates, the following strict inequalities are known.

Lemma 4 *For all $n_1, n_2 \geq 1$,*

$$\text{FIN} \subset \pitchfork^*_{\text{FIN},n_1,n_2}(\text{FIN}) \subset H(\text{FIN}, \text{FIN}) \subset \text{REG}.$$

6 Computational Power

6.1 Closure Properties

Many results on the computation power of TGR are inherited from Sect. 5. In this section, we describe some additional closure properties for TGR.

Despite Lemma 4, every regular language can be expressed as the coding of the language resulting from iterating TGR on a finite language with a finite set of templates. Recall that a *coding* (or letter-to-letter morphism) is a morphism such that the image of each letter is again a letter. Let $C(\mathcal{L})$ be the coding closure of a class of languages $\mathcal{L}$.

Lemma 5 *The following equality holds*:

$$\text{REG} = C\big(\pitchfork^*_{\text{FIN}}(\text{FIN})\big).$$

Daley and McQuillan [7] also present several results on the closure properties of full AFLs under iterated TGR. They rely in part on the concept of usefulness for templates. We say that a template $t \in T$ is *useful* on L, n_1, n_2 if there exists $u_1\alpha\beta v_1, v_2\beta\gamma u_2 \in \pitchfork^*_{T,n_1,n_2}(L)$ with $|\alpha|, |\gamma| \geq n_1$, $|\beta| = n_2$, $u_1, u_2, v_1, v_2 \in \Sigma^*$ and $t = \alpha\beta\gamma$.

That is, a template t is useful with respect to a given language L if t can be used to combine two words in the iterated TGR of L. Note that the requirements do not insist that the recombination which can occur when t is useful—that is, the assembly of $u_1\alpha\beta\gamma u_2$ in our notation—is distinct from $u_1\alpha\beta v_1$ and $v_2\beta\gamma u_2$.

If every template $t \in T$ is useful on L, n_1, n_2, then we say that T is useful on L, n_1, n_2.

Theorem 1 *Let $n_1, n_2 \geq 1$, $\mathcal{L}$ be a full AFL and $L, T \in \mathcal{L}$. If T is useful on L, n_1, n_2, then $\pitchfork^*_{T,n_1,n_2}(L) \in \mathcal{L}$.*

Thus, provided that we restrict ourselves to sets of templates in which every template can be used at some point in an operation of $\pitchfork^*_{T,n_1,n_2}$, then full AFLs are closed under iterated TGR. Daley and McQuillan [7] show that if T is a regular set of templates, and L is any language, the subset of T corresponding to all useful templates on L is also regular. However, the construction provided by Daley and McQuillan is not effective. Thus, as the regular languages are a full AFL, we have the following corollary.

Corollary 2 *For all full AFLs $\mathcal{L}$, $\pitchfork^*_{\mathrm{REG}}(\mathcal{L}) = \mathcal{L}$.*

However, this result is not effective. McQuillan et al. [14] have given effective closure properties of iterated TGR. In particular, they show the following results.

Theorem 2 *Let $n_1, n_2 \geq 1$, L be a regular (resp., context-free) language and T be a regular set of templates. If L and T are effectively given, then $\pitchfork^*_{T,n_1,n_2}(L)$ is an effective regular (resp., context-free) language.*

6.2 Closure Properties of Intra-molecular TGR

As mentioned in Sect. 4.6, we can view TGR as occurring on a single strand of DNA. The result is a unary word operation. Daley et al. [4] have investigated the formal language theoretic properties of the operation $\pitchfork^{(\ell)}_T$; the results below are phrased for $n_1 = n_2 = 1$, but analogues for other values of n_1, n_2 are not difficult to obtain. In what follows, we let $\pitchfork^{(\ell)}_T$ denote the intra-molecular TGR operation with $n_1 = n_2 = 1$.

Daley et al. [4] note that $\pitchfork^{(\ell)}_T(L) \subseteq ld(L)$ for all L and all T. However, even for $T = \Sigma^*$, there exist languages such that $\pitchfork^{(\ell)}_T(L) \neq ld(L)$. That is, because of the requirements for α and β to be non-empty, the intra-molecular TGR cannot directly simulate ld.

Theorem 3 *Let $\mathcal{L}$ be a cone. Let $L, T \subseteq \Sigma^*$ such that one is from $\mathcal{L}$ and the other is regular. Then $\pitchfork^{(\ell)}_T(L) \in \mathcal{L}$. In particular,*

(a) *If T, L are regular, then $\pitchfork^{(\ell)}_T(L)$ is a regular language.*
(b) *If T is regular and L is a CFL, then $\pitchfork^{(\ell)}_T(L)$ is a CFL.*
(c) *If T is a CFL and L is regular, then $\pitchfork^{(\ell)}_T(L)$ is a CFL.*

We contrast the above result with two non-closure properties [4].

Theorem 4 *The following non-closure properties hold:*

(a) *There exist CFLs T, L such that $\pitchfork^{(\ell)}_T(L)$ is not a CFL.*
(b) *There exist a context-sensitive language $L \subseteq \Sigma^*$ and a finite set of templates $T \subseteq \Sigma^*$ such that $\pitchfork^{(\ell)}_T(L)$ is not recursive.*

We note that, to date, no computational study of the templated hi or $dlad$ operations under the RNA template model of Angeleska et al. [2] has been undertaken. Further, the iterated version of the intra-molecular TGR operation presented here has not been investigated.

6.3 Closure Property Dependencies

Daley and McQuillan [6] have shown dependencies in closure properties between TGR and related operations. For instance, the following result gives conditions under which closure under TGR implies closure under intersection.

Lemma 6 *Let $\mathcal{L}$ be a class of languages closed under left and right concatenation with a single symbol, quotient with a single symbol and union with singletons. Let $\mathcal{T}$ be a class of languages closed under left and right concatenation with a single symbol. If $\pitchfork_{\mathcal{T},n_1,n_2}(\mathcal{L}) \subseteq \mathcal{L}$ for any $n_1, n_2 \geq 2$, then $\mathcal{L}$ is closed under intersection with languages from $\mathcal{T}$.*

A result of this form can also be given for closure under concatenation.

Lemma 7 *Let $\mathcal{L}$ be a class of languages closed under limited erasing homomorphism, union, left and right concatenation by a symbol. Let $\mathcal{T}$ be any class of languages containing the singleton languages. If $\pitchfork_{\mathcal{T},n_1,n_2}(\mathcal{L}) \subseteq \mathcal{L}$ for any $n_1, n_2 \geq 1$, then $\mathcal{L}$ is closed under concatenation.*

We can also consider results which examine conditions necessary to guarantee closure under TGR. In the following lemma, we use the notation $\mathcal{L}_1 \wedge \mathcal{L}_2$ to mean $\mathcal{L}_1 \wedge \mathcal{L}_2 = \{L_1 \cap L_2 : L_1 \in \mathcal{L}_1, L_2 \in \mathcal{L}_2\}$.

Lemma 8 *If $\mathcal{L}$ be a full trio closed under concatenation and let $\mathcal{T}$ be a trio or $\mathcal{T} \subseteq$ REG. If $\mathcal{L} \wedge \mathcal{T} \subseteq \mathcal{L}$, then $\pitchfork_{\mathcal{T}}(\mathcal{L}) \subseteq \mathcal{L}$.*

As a corollary, we have the following result.

Corollary 3 *Let $\mathcal{L}$ be a intersection-closed full semi-AFL, $\pitchfork_{\mathcal{L}}(\mathcal{L}) \subseteq \mathcal{L}$.*

Daley and McQuillan also show that the context-sensitive languages are not closed under TGR with a finite set of templates for any $n_1, n_2 \geq 1$ [6].

For intra-molecular TGR, the following results have been obtained [4].

Lemma 9 *Let $\mathcal{L}$ be a class of languages closed under concatenation and quotient with a single symbol, and under $\pitchfork^{(\ell)}$. Then $\mathcal{L}$ is closed under intersection.*

Theorem 5 *Let $L \subseteq \Sigma^*$ be a r.e. language. Then there exist an alphabet Δ, linear CFLs $L_1, L_2 \subseteq \Delta^*$ and a morphism $h : \Delta^* \to \Sigma^*$ such that the following equality holds*:

$$L = h\big(\pitchfork_{L_2}^{(\ell)} (L_1)\big).$$

7 Equivalence

Beyond the closure properties of TGR, the equivalence problem for sets of templates has recently been considered [8]. By the equivalence problem, we mean questions of the form "Given sets of templates T_1, T_2, are $\pitchfork_{T_1}$ and $\pitchfork_{T_2}$ identical operations?" Such questions are important in the consideration of TGR as a natural computing operation, since they help identify the necessary modifications to a set of templates in order to change the resulting rearrangement process in a meaningful way.

We require some additional formal language theoretic terminology. A word $x \in \Sigma^*$ is a *prefix* of a word $y \in \Sigma^*$ if there exists $w \in \Sigma^*$ such that $y = xw$. Similarly, x is a *suffix* of y if there exists $u \in \Sigma^*$ such that $y = ux$. If $x \in \Sigma^*$, then $\mathrm{pref}(x)$ (resp., $\mathrm{suff}(x)$) is the set of all prefixes (resp., suffixes) of x.

Let $n_1, n_2 \geq 1$. For $T_1, T_2 \subseteq \Sigma^*$, we say that T_1 and T_2 are (n_1, n_2)*-equivalent*, denoted by $T_1 \equiv_{n_1,n_2} T_2$, if $\pitchfork_{T_1,n_1,n_2} (L) = \pitchfork_{T_2,n_1,n_2} (L)$ for all $L \subseteq \Sigma^*$. By $T_1 \sqsubseteq_{n_1,n_2} T_2$, we mean $\pitchfork_{T_1,n_1,n_2} (L) \subseteq \pitchfork_{T_2,n_1,n_2} (L)$ for all languages $L \subseteq \Sigma^*$. It is easy to see that $T_1 \equiv_{n_1,n_2} T_2$ if and only if $T_1 \sqsubseteq_{n_1,n_2} T_2$ and $T_2 \sqsubseteq_{n_1,n_2} T_1$.

Example 1 Let $n_1 = n_2 = 1$, $T_1 = \{aaa, aab, abc, aaaabc\}$ and $T_2 = \{aaa, aab, abc\}$. Then we can verify that $T_1 \equiv_{n_1,n_2} T_2$. In particular, any use of $aaaabc$ in T_1 can be simulated by one of the three templates in T_2, depending on the decomposition of $aaaabc$ into $\alpha\beta\gamma$ with $|\beta| = 1$.

Let (C1) be the following condition:

$$\begin{aligned} &\forall t, t_1, t_2 \in \Sigma^* \text{ with } |t| = 2n_1 + n_2, \text{ if } t_1 t t_2 \in T_1 \\ &\qquad \text{then } \exists t_1' \in \mathrm{suff}(t_1), t_2' \in \mathrm{pref}(t_2)\ (t_1' t t_2' \in T_2). \end{aligned} \tag{C1}$$

Condition (C1) is illustrated in Fig. 8: for every sub-word t of length $2n_1 + n_2$ in a template in T_1, there must be an extension of t in T_2 which agrees with the template in T_1 on the sub-words flanking t.

Then we can give an exact characterization of equivalence [8]:

Theorem 6 *Let Σ be an alphabet with $|\Sigma| \geq 3$, $n_1, n_2 \geq 1$ and $T_1, T_2 \subseteq \Sigma^*$. The condition* (C1) *holds if and only if $T_1 \sqsubseteq_{n_1,n_2} T_2$.*

Thus, we can characterize when two sets of templates define the same TGR operation in formal language theoretic terms. This gives us the tool to determine which changes can be made to a set of templates in order to affect the rearrangement process.

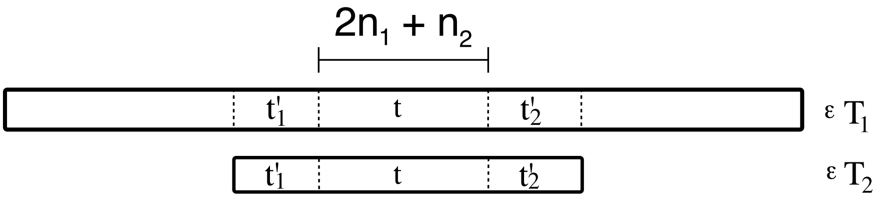


Fig. 8 Illustration of condition (C1)

Lemma 10 *Let $n_1, n_2 \geq 1$ and $T_1, T_2 \subseteq \Sigma^*$ ($|\Sigma| \geq 3$) be regular sets of templates. Then it is decidable whether $T_1 \equiv_{n_1,n_2} T_2$.*

However, the above result does not give an efficient algorithm. It is open to determine if efficient algorithms exist for deciding whether two sets of templates are equivalent. In contrast to Lemma 10, as would be expected, there exists a fixed regular set of templates T_0 such that the problem "Given a context-free set of templates T, is $T \equiv_{n_1,n_2} T_0$?" is undecidable [8]. However, $T_0 \neq \Sigma^*$: it is possible to test equivalence to Σ^* for recursive sets of templates.

It can also be shown that if $\pitchfork_T$ is replaced by the intra-molecular operation $\pitchfork_T^{(\ell)}$, the characterization in (C1) still holds. However, it is open whether the same characterization also holds for equivalence of iterated TGR. Further, from a formal language point of view, it is interesting to ask whether the condition that $|\Sigma| \geq 3$ can be substituted with $|\Sigma| \geq 2$ in Theorem 6; this problem is also currently open.

8 Covering and Scaffolding

We briefly describe another line of research prompted by the TGR model—that of covering from templates and scaffolding. Ehrenfeucht and Rozenberg [10] recently introduced the concepts of covers from templates. Their results deal with covering of a word x by a collection of words from a given language L. However, as the authors note, unlike previous work in this area, the authors here are concerned with the specific positions of x which are covered by words from L. In other words, a sub-word x' of x which occurs in two different positions in x is not necessarily covered in both positions if $x' \in L$.

In particular, if $n, m \in \mathbb{N}$ with $n \leq m$, let $[n, m]$ denote the set $\{n, n+1, \ldots, m\}$. Then for an alphabet Σ, a segment is a function $f : [n, m] \to A$. Thus, a segment is a word which does not necessarily begin at the index 1, but which does occupy some range of indices.

Note that under this definition, a word is any segment on an interval of the form $[1, n]$ for some $n \geq 0$. The set of all segments over an alphabet Σ is denoted by $\mathcal{S}_\Sigma$.

Given a set $C \subseteq \mathcal{S}_\Sigma$ and a segment $f \in S$, we say that C is a *cover* of f if $f = \bigcup_{z \in C} z$. This cover of f is said to be *tight* if for all $z \in C$, $C - \{z\}$ is not a cover of f. A cover C of f is said to be *small* (with respect to some set F with

$C \subseteq F \subseteq \mathcal{S}_\Sigma$) if C is minimal with respect to cardinality among all covers $Z \subseteq F$ of f. Denote by $\mathrm{SC}_F(f)$ the set of all small covers of f with respect to F. The *small index* of f (with respect to F) is the cardinality of the small covers of f with respect to F; this concept is well defined since all small covers of f with respect to F have the same cardinality. For any small cover C of f with respect to F, we let $C = \{C(1), C(2), \ldots, C(m)\}$, where the positions are determined by increasing left-most elements of the elements $C(i)$; as noted by Ehrenfeucht and Rozenberg; this is also well defined.

Example 2 Let $\Sigma = \{a, b, c\}$ and $f \in \mathcal{S}_\Sigma$ be defined by

$$f = \{(2, a), (3, b), (4, a), (5, b), (6, c)\}.$$

This is abbreviated as $f = (2, ababc)$. The set $C = \{(2, ab), (4, ab), (5, bc)\}$ is a cover of f. However, with respect to the set $F = \{(2, ab), (4, ab), (5, bc), (4, abc)\}$, C is not small, as $C' = \{(2, ab), (4, abc)\}$ is a cover of f with smaller cardinality.

These definitions of covers are motivated by viewing f as the assembled macro-nuclear gene of the ciliate and F as the set of segments available for assembly from the scrambled micro-nuclear gene.

Let $f \in \mathcal{S}_\Sigma$, $F \subseteq \mathcal{S}_\Sigma$ and let m be the small index of f with respect to F. Let $1 \le i \le m$. The ith kernel of f with respect to F (denoted $\ker_{i,F}(f)$) is defined by

$$\ker_{i,F}(f) = \left(\bigcap_{C \in \mathrm{SC}_F(f)} C(i) \right) - \bigcup_{C' \in \mathrm{SC}_F(f)} \bigcup_{j \ne i} C'(j).$$

The set of kernels of f is called the scaffold of f (with respect to F). Ehrenfeucht and Rozenberg show the following result on the kernels of f.

Theorem 7 *Let $f \in \mathcal{S}_\Sigma$ and $F \subseteq \mathcal{S}_\Sigma$. For all $1 \le i \le m$, the set $\ker_{i,F}(f)$ is a non-empty segment.*

Example 3 Let $f = (1, abbabaaab)$ and $F = \{g_i\}_{i=1}^{6}$ where $g_1 = (1, ab)$, $g_2 = (1, aab)$, $g_3 = (2, bba)$,$g_4 = (4, abaa)$, $g_5 = (3, babaaa)$, $g_6 = (7, aab)$, and $g_7 = (9, b)$. Then it is not hard to see that $m = 3$ and that $\mathrm{SC}_F(f)$ consists of the five covers $\{g_2, g_4, g_6\}$, $\{g_2, g_5, g_6\}$, $\{g_2, g_5, g_7\}$, $\{g_1, g_5, g_6\}$, and $\{g_1, g_5, g_7\}$.

With this, we can calculate that

$$\ker_{1,F}(f) = g_1,$$
$$\ker_{2,F}(f) = (4, aba),$$
$$\ker_{3,F}(f) = g_7.$$

The relationship between the scaffold of f and certain covers of f requires us to introduce the notion of long segments. We say that a segment $g \in P_F(f)$ is *long* (with respect to F) if it is not properly included in any other segment in F; the set of all long segments of f (with respect to F) is denoted $\mathrm{LP}_F(f)$. For example, g_5 in Example 3 is a long segment, while g_4 is not, since g_4 is properly contained in g_5. A cover is said to be long if it consists only of long segments. A long small cover is a cover that is both long and small.

Example 4 Continuing from Example 3, segments g_1, g_4, and g_7 are not long. Therefore, the only long small cover of f is $\{g_2, g_5, g_6\}$.

For each $1 \le i \le m$, let $\mathrm{LP}_F(f, i) = \{y \in \mathrm{LP}_F(f) : \mathrm{ker}_{i,F}$ is contained in $y\}$. That is, we categorize the long segments of f with respect to F according to containment of kernels. It is not hard to see that $\mathrm{LP}_F(f, i)$ is an ordered set. Thus, we let $rt_F(f, i)$ (resp., $lt_F(f, i)$) be the maximal (resp., minimal) element of $\mathrm{LP}_F(f, i)$. With these notations, we can state the following result.

Theorem 8 *The sets* $\{rt_F(f, i)\}_{i=1}^m$ *and* $\{lt_F(f, i)\}_{i=1}^m$ *are long small covers of* f *with respect to* F.

Example 5 Continuing from Example 4, we get that

$$\mathrm{LP}_F(f, 1) = \{g_2\}, \qquad \mathrm{LP}_F(f, 2) = \{g_5\}, \qquad \mathrm{LP}_F(f, 3) = \{g_6\}.$$

Thus, we have that $rt(f, i)$ and $lt_F(f, i)$ are the unique elements of $\mathrm{LP}_F(f, i)$ for $1 \le i \le 3$.

As noted by Ehrenfeucht and Rozenberg, we have that both $rt_F(f, i)$ and $lt_F(f, i + 1)$ are adjacent to (i.e., contact but do not overlap with) $\mathrm{ker}_{i,F}(f)$ for all i with $2 \le i \le m$. On the boundaries, $\mathrm{ker}_{1,F}(f)$ is adjacent to $lt_F(f, i)$ and $\mathrm{ker}_{m,F}(f)$ is adjacent to $rt_F(f, m - 1)$. We can verify this in Example 5.

Ehrenfeucht and Rozenberg also examine the structure of the scaffold and bridges, which link the kernels of f. The relationships between long segments and the kernels and bridges of f are also considered [10].

9 Experimental Results

The hypothesis that the gene descrambling process in stichotrichous ciliates involves template-guided recombination was recently tested in the ciliate *Oxytricha trifallax* by Nowacki et al. [15]. Initially, experiments using RNA interference (RNAi) were undertaken. RNAi is an experimental technique that allows the selective degradation of RNA molecules in a cell which contain a chosen target sequence. When RNAi was performed against putative template sequences for a particular gene, it was observed that this gene was incorrectly descrambled. This demonstrated that *Oxytricha* is not able to descramble scrambled genes in the absence of RNA molecules containing template sequences; thus, templates are a *necessary* component of the descrambling mechanism.

To test if TGR is, in some sense, *sufficient* to account for gene descrambling, a series of more ambitious experiments were performed: DNA and RNA were directly micro-injected into the macronuclei of conjugating *Oxytricha* cells. In the first set of experiments, artificial chromosomes (double-stranded DNA), coding for incorrectly descrambled versions of the genes *TEBPα* and *TEBPβ* were micro-injected. It was observed in the progeny of the micro-injected cells that corresponding incorrectly-descrambled versions of *TEBPα* and *TEBPβ* were present. This result supports the hypothesis that descrambling is guided by templates made from the chromosomal DNA of the old macronucleus but leaves open the question of whether the templates are DNA or RNA.

The second series of experiments involved micro-injection of RNA sequences designed to intentionally mis-descramble *TEBPβ* if actually used by the cell as templates. It was observed that the progeny of RNA micro-injected cells contained correspondingly incorrectly-descrambled *TEBPβ* genes, thus providing support for the RNA template model.

10 Concluding Remarks

We have provided a brief introduction to conjugation and gene descrambling during macro-nuclear development in ciliated protozoa, along with an exposition of two theoretical models for descrambling: the original template-guided recombination model of Prescott et al. [16] and the RNA-template model of Angeleska et al. [2].

Following this, we reviewed the formalization of this process to a ternary word operation, TGR, which combines words in a splicing-like manner, controlled by a template word; we noted that this formalization is consistent with both the models of Prescott et al. [16] and Angeleska et al. [2]. Variants of TGR were described, including iterated TGR which is a more realistic model for ongoing biological processes, TGR with added deletion contexts and intramolecular TGR.

It was shown that under some parametric restrictions, one could construct a splicing system equivalent to a single-application TGR system although, in general, TGR and iterated TGR are strictly weaker operations than the equivalent splicing operations. Despite this limitation, iterated TGR on a finite base language, with a finite template set, is able to generate a coding of an arbitrary regular language. More generally, all full AFLs are closed under iterated TGR in the case where one is able to guarantee that the template set is "useful". For the intramolecular case of TGR, regular languages are closed while if one of the template, or base language, is regular, and the other context-free, the result of intramolecular TGR will be context-free.

For single-application and intramolecular TGR, it is decidable if two regular template sets are equivalent (i.e., they will define the same operation). For the case of context-free templates, the same question becomes undecidable. The decidability status of template equivalence remains open in the iterated case.

A related line of research recently introduced by Ehrenfeucht and Rozenberg [10], describing coverings and scaffolds, was also reviewed. Motivated by the notion of micro-nuclear gene fragments "covering" a macro-nuclear gene, this research centres on the investigation of how a given word may be "covered" by words from a fixed language.

Finally, we described the recent experimental results of Nowacki et al. [15] which verify the biological validity of the RNA template-guided recombination model. The fact that such strong experimental support now exists for the correctness of the template model casts theoretical results in an exciting light, and motivates further study, as we now know that we are reasoning about the properties of an actual "natural computer".

References

1. Alhazov A, Petre I, Rogojin V (2008) Solutions to computational problems through gene assembly. In: Garzon M, Yan H (eds) DNA computing. Lecture notes in computer science, vol 4848. Springer, Berlin, pp 36–45
2. Angeleska A, Jonoska N, Saito M, Landweber LF (2007) RNA-guided DNA assembly. J Theor Biol 248:706–720
3. Chang W, Kuo S, Landweber LF (2006) A new scrambled gene in the ciliate uroleptus. Gene 368:72–77
4. Daley M, Domaratzki M, Morris A (2007) Intra-molecular template-guided recombination. Int J Found Comput Sci 18:1177–1186
5. Daley M, McQuillan I (2005) Template-guided DNA recombination. Theor Comput Sci 330:237–250
6. Daley M, McQuillan I (2006) On computational properties of template-guided DNA recombination in ciliates. In: Carbone A, Pierce N (eds) DNA computing. Lecture notes in computer science, vol 3892. Springer, Berlin, pp 27–37
7. Daley M, McQuillan I (2006) Useful templates and iterated template-guided DNA recombination in ciliates. Theory Comput Syst 39:619–633
8. Domaratzki M (2008) Equivalence in template-guided recombination. Nat Comput 7(3):439–449
9. Ehrenfeucht A, Prescott D, Rozenberg G (2007) A model for the origin of internal eliminated segments (IESs) and gene rearrangement in stichotrichous ciliates. J Theor Biol 244:108–114
10. Ehrenfeucht A, Rozenberg G (2006) Covers from templates. Int J Found Comput Sci 17:475–488
11. Ehrenfeucht A, Harju T, Petre I, Prescott D, Rozenberg G (2004) Computation in living cells: gene assembly in ciliates. Springer, Berlin
12. Jaraczewski JW, Jahn CL (1992) Elimination of Tec elements involves a novel excision process. Genes Dev 7:95–105
13. Kari L (1994) On language equations with invertible operations. Theor Comput Sci 132:129–150
14. McQuillan I, Salomaa K, Daley M (2006) Iterated TGR languages: membership problem and effective closure properties. In: Chen D, Lee D (eds) Computing and combinatorics. Lecture notes in computer science, vol 4112. Springer, Berlin, pp 94–103
15. Nowacki M, Vijayan V, Zhou Y, Schotanus K, Doak T, Landweber LF (2008) RNA-mediated epigenetic programming of a genome-rearrangement pathway. Nature 451:153–159
16. Prescott D, Ehrenfeucht A, Rozenberg G (2003) Template-guided recombination for IES elimination and unscrambling of genes in stichotrichous ciliates. J Theor Biol 222:323–330
17. Păun G, Rozenberg G, Salomaa A (1998) DNA computing. Springer, Berlin
18. Rozenberg G, Salomaa A (eds) (1997) Handbook of formal languages. Springer, Berlin

Part IV Nanoconstructions and Self-assembly

DNA Cages with Icosahedral Symmetry in Bionanotechnology

Nataša Jonoska, Anne Taormina, and Reidun Twarock

Abstract Blueprints for polyhedral cages with icosahedral symmetry made of circular DNA molecules are provided. The basic rule is that every edge of the cage is met twice in opposite directions by the DNA strand(s), and vertex junctions are realized by a set of admissible junction types. As nanocontainers for cargo storage and delivery, the icosidodecahedral cages are of special interest because they have the largest volume per surface ratio of all cages discussed here.

1 Introduction

Recent advances in biotechnology provide the necessary tools to engineer cage structures from nucleic acids, and open novel avenues for applications in nanotechnology. Cages with crystallographic symmetry have already been realized experimentally in the shape of a cube [1], a tetrahedron [2], an octahedron [3], or a truncated octahedron [4], and one natural idea is to use such cages for cargo delivery or storage [5]. Moreover, models for two realizations of a cage with a non-crystallographic symmetry have recently been proposed from a theoretical point of view [6, 7]. These studies were motivated by the hope that such mathematical considerations on the organization of RNA or DNA in cages with icosahedral symmetry would aid the design of artificial cages inspired by nature.

We provide here a systematic comparative analysis of three polyhedral cages with icosahedral symmetry, the icosahedron, the dodecahedron [7], and the icosidodecahedron [6] (see Fig. 1), which are—from a mathematical point of view—distinguished because they are the three smallest vertex sets realizing icosahedral symmetry. Moreover, their edges are uniformly of the same length. We expect that these properties make them easier to be realized experimentally than other polyhedra with this symmetry, and we therefore focus on these three cases here.

An interesting feature of these polyhedra is the fact that they have vertices of different connectivity: The dodecahedron has trivalent vertices, and hence needs to be realized in terms of three-junctions, whilst the icosidodecahedron requires four-junctions and the icosahedron five-junctions. We investigate here possibilities of realizing these polyhedra with a single circular DNA molecule. Our analysis shows

N. Jonoska (✉)
Department of Mathematics, University of South Florida, Tampa, FL 33620, USA
e-mail: jonoska@math.usf.edu

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_9, © Springer-Verlag Berlin Heidelberg 2009

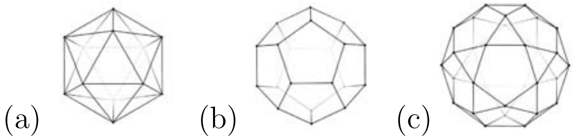


Fig. 1 The icosahedron (**a**), dodecahedron (**b**), and icosidodecahedron (**c**) corresponding to the three polyhedra with icosahedral symmetry with less than 60 vertices

that the complexity of this problem increases when the polyhedra considered have even degree vertices. The strategy implemented here is to enumerate all possible inequivalent realizations of these cages with a single strand of DNA molecule, given a set of rules on the junctions. This combinatorial approach is manageable without computer help for the icosahedral and the dodecahedral cages, but becomes cumbersome for the icosidodecahedron. Our treatment of the latter in this paper is therefore computer-aided, but a more elegant mathematical framework to solve this type of problem is being developed.

Our analysis shows that in all three cases at least two circular DNA strands are needed to form the cage. From that point of view, all three polyhedra lend themselves equally well for templates of DNA cages. However, the icosidodecahedron is the polyhedron among the three with the largest volume to surface ratio (at an edge length of 1, it is approximately 0.47, compared to 0.37 for the dodecahedron, 0.25 for the icosahedron[1]), making it perhaps the most interesting icosahedral cage for applications in nanotechnology.

We start by introducing our theoretical construction method in general terms for all polyhedra with icosahedral symmetry in Sect. 2, and then provide details for the three polyhedra in the subsequent sections: the icosahedral cage in Sect. 3, the dodecahedral cage in Sect. 4, and the icosidodecahedral cage in Sect. 5.

2 Construction of Cages with Icosahedral Symmetry: General Principles

The goal is to provide blueprints for polyhedral cages with icosahedral symmetry, made of a circular DNA molecule, with the basic rule that every edge of a given polyhedral cage is met twice in opposite directions by the strand. This requirement enables hybridization of the two portions of the strand running along an edge into a double helix structure.

Mathematically, a cage is a graph whose nodes are the vertices of the corresponding polyhedron, and the connectors are the edges. As explained in [8, 9], the

[1] Moreover, the volume to area ratios of all icosahedral cages considered here are larger than those of the crystallographic cages realized to date. For comparison, the value for the octahedron is approximately 0.14.

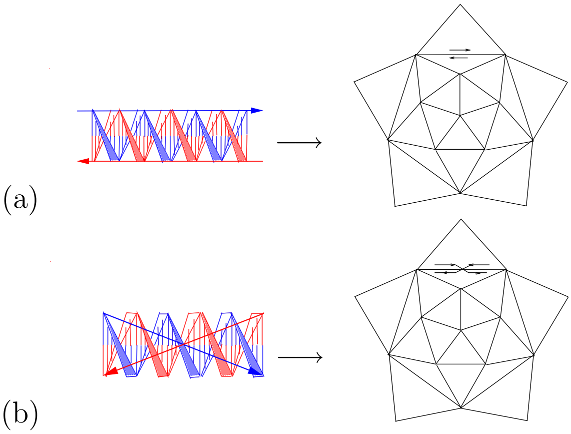


Fig. 2 (**a**) The DNA double helix is represented by *lines* (*blue* and *red*) that trace the backbone of the helices. (**b**) Depending on their lengths, additional half-turns may appear, that are represented by cross-overs on the planar representation of the graph

idea is to topologically embed graphs into orientable thickened graphs as deformation retracts. Such thickened graphs are compact orientable 2-dimensional surfaces constructed out of strips and thickened n-junctions glued together. They can be considered as models for DNA cages as follows: The boundary curves of the thickened graph represent single stranded DNA molecules in the cage, and provide a blueprint that specifies which types of junctions have to be used to realize this graph.

The design of such templates for DNA cages must take the following factors into account:

1. *Initial data*: Assume that the cages correspond to polyhedra with all edges of equal length λ. Then the number $\nu(\lambda)$ of half-turns in the duplex structure along each edge depends on λ. Configurations where $\nu(\lambda)$ is odd are modeled as cross-overs in the planar projective views of the polyhedral cages as shown in Fig. 2. Note that for DNA there are about 10.5 base pairs (bp) per helical turn.
2. *Thickened n-junctions*: Mechanical stress may be imposed on the overall configuration if the strands of the edges cross each other at the incident vertex junctions. For example, the thickened n-junction shown in Fig. 3(a) imposes no stress on the configuration (we name it 'type A_n'), whilst the thickened n-junctions appearing in Fig. 3(b) and (c) accommodate one or two cross-overs (we name them 'type B_{1n}' and 'type B_{2n}') and may impose stress on the overall configuration unless extra nucleotides are introduced along the corresponding edge that compensate for it. In general, a thickened n-junction with k cross-overs of the strands is of type $\mathrm{B}_{kn}, k \leq n$.

The number n of legs in the junctions depends on the type of cage considered. In subsequent sections, cages with thickened 5-junctions (icosahedra), 4-junctions[2] (icosidodecahedra) and 3-junctions (dodecahedra) are discussed.

[2] Stable four-junctions can be assembled; see, for example, [10].

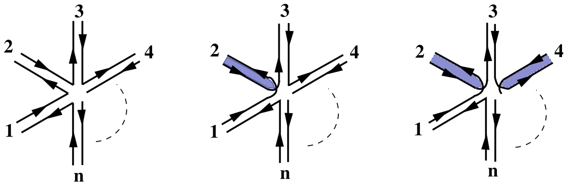


Fig. 3 (**a**) Energetically optimal thickened n-junction (type A_n); (**b**) higher energy (one strand "reaches" across to another) thickened n-junction (type B_{1n}); (**c**) higher energy (two strands cross-over their neighbors) thickened n-junction (type B_{2n}). The dotted line indicates that there may be more edges present

2.1 The Construction Procedure: Step I

The first step in the construction procedure is to identify *start configurations*, i.e., orientable thickened graphs with a maximum number of type A_n thickened junctions. Such graphs are usually made of several distinct circular strands. In order to determine the start configuration, first assume that every vertex of the polyhedron is represented by a junction of type A_n. This is always possible if the polyhedral edges have an even number of helical half-turns. The start configuration in this case is given by N separate circular strands (loops), where N is the number of polyhedral faces.

However, in the presence of cross-overs which take into account the odd number of half-turns along the edges, this distribution of type A_n junctions does not necessarily provide an orientable thickened graph. In particular, this is the case if faces with an odd number of edges occur in the polyhedron (as is the case for all three polyhedra considered here). In particular, if the cages have all edges of the same length as in the present analysis, and moreover, exhibit an odd number of half-turns in the double helix along each edge, the start configuration is obtained as follows: The two-dimensional surface, orientable or not, obtained by gluing the twisted strips representing the cross-overs to type A_n junctions, is called the *initial data configuration* (see Fig. 4 for the initial data configuration of the icosahedron; note that this configuration has 6 loops, but they do not all run in opposite directions). To this initial start configuration, we apply the *bead rule* to obtain a start configuration. The bead rule consists of placing beads on selected edges of the polyhedron to indicate that a twisted strip (cross-over) is glued to a twisted leg of a type B_{kn} thickened junction, as illustrated in Fig. 5 for a 4-coordinated polyhedron.

Note that by addition of a "cross-over" at one vertex the orientation of the boundary components change at all neighboring edges. For polyhedra with an odd number of half-turns on their edges, a start configuration is obtained if the following rules are satisfied:

- Each edge accommodates either a cross-over or a bead.
- Every face of the polyhedron in the start configuration must have an *even* number of cross-overs.
- The number of beads is minimal.

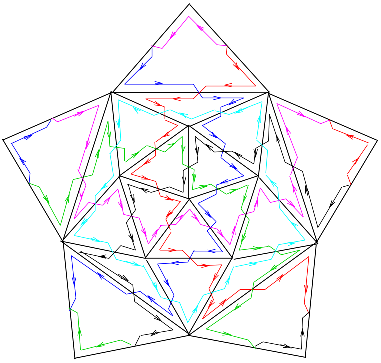


Fig. 4 Initial data configuration for an icosahedral cage when $\nu(\lambda)$ is odd

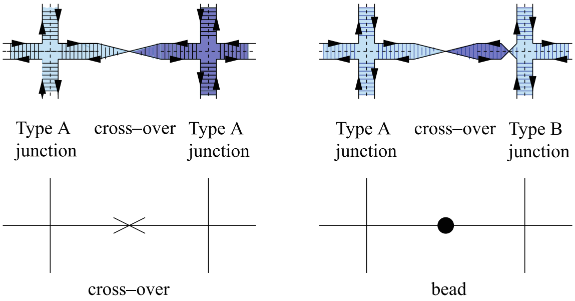


Fig. 5 Correspondence of a "cross-over" and a "bead" to the boundary orientation. An odd number of helical half-turns corresponds to a "cross-over" (*left*), and the addition of one half-turn corresponds to placing a "bead" on the edge (*right*)

After the bead rule has been applied, one must discard configurations which are equivalent under icosahedral symmetry. The symmetry-inequivalent bead configurations correspond to the possible start configurations. In such configurations some junctions are of type A_n, and others of type B_{kn}, the latter having been introduced to provide orientability of the two-dimensional surface representing the thickened graph obtained from the polyhedron.

2.2 The Construction Procedure: Step II

Start configurations are usually given in terms of more than one strand of DNA. Since focus is here on building cages out of a minimal number of strands, these need to be "merged." This can be achieved by replacing some of the junctions in the start configuration by junctions with different connectivity of the strands. Such replacements result in a change of the overall number of strands used to form the cage and, therefore, depend on the overall structure, which is different in all three cases considered here. These replacements are therefore discussed on a case by case basis in the following sections.

Note that these replacements result in a change of the number of half-turns of the DNA strands, because they add or remove a half-turn in the duplex structure [7]. If the edges of the polyhedral cage are of equal length and are rigid duplexes, i.e., have hybridization going up to the vertices, then there is little flexibility of the strands at the vertices to cross over from one edge to another, and we assume that such replacements of the vertex structure are difficult. The replacements of one type of junction with another are therefore carried out under the assumption that the strands at the vertices of the cage are not completely hybridized or the lengths of the edges may differ by a few nucleotides. In these cases, the changes in the connection of the strands introduced by these replacements do not interfere with the embedding of the double helical structure.

3 The Icosahedral Cage

The rules described above can be applied in a straightforward manner to the construction of icosahedral cages, and we start by considering the case where an extra twist occurs on each edge, i.e., the edges are of equal length with an odd number of half-turns. An icosahedron is a five-coordinated polyhedron with twenty triangular faces. According to the preceding section, orientability requires an odd number of beads per face, and the minimum number of 10 beads is achieved with exactly one bead per face.

Compiling an exhaustive list of start configurations in this case is effortless. For instance, the bead rule can be implemented by first considering one of the 12 five-coordinated vertices of the icosahedron, and asking how many allowed possibilities there are to place beads on the triangular faces forming a pentagon whose center is the vertex in question.

Direct inspection shows that there are three inequivalent ways to obtain a minimal bead distribution; all illustrated in Fig. 6. The 3-, 4-, and 5-bead configurations are labeled $(p, r) = (1, 2), (3, 1)$, and $(5, 0)$, respectively, where p is the number of beads on the pentagon perimeter, and r is the number of beads on the radii.

The following strategy provides a full set of start configurations (all letters used for vertices refer to the labeling convention shown in Fig. 7(a)):

1. Place a $(5, 0)$ pentagon configuration at any vertex (choose A, say, and label this configuration $(5, 0)_{\mathrm{A}}$). Due to the high symmetry of this bead distribution, the only bead distribution for pentagons with center vertices B, C, D, E, and F is the 3-bead one, as the corresponding vertices automatically have at least two beads on radii. There is no allowed bead distribution with more than two beads on radii, so $(1, 2)$ is the only possibility. A similar argument holds for the vertices G, H, I, J, and K, which must be of type $(1, 2)$. Finally, vertex L is automatically

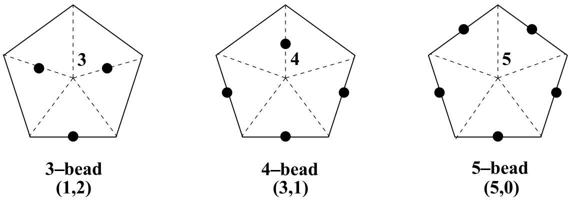


Fig. 6 Minimal bead configurations on a triangulated pentagon compatible with the bead rule

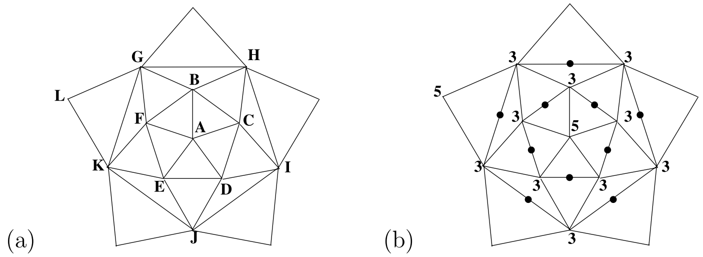
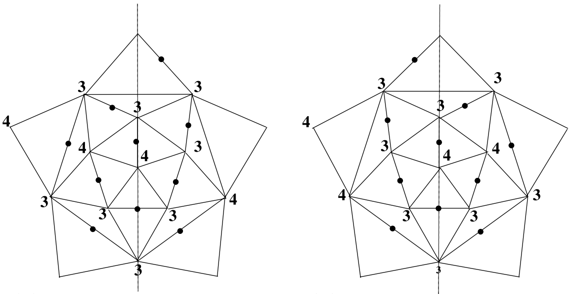


Fig. 7 (**a**) Choice of vertex labels on the icosahedron. (**b**) Start configuration when the bead distribution $(5, 0)$ is placed at vertex A (configuration 1 in Table 1). The number associated to each vertex counts the number of beads in the pentagon whose center is that vertex

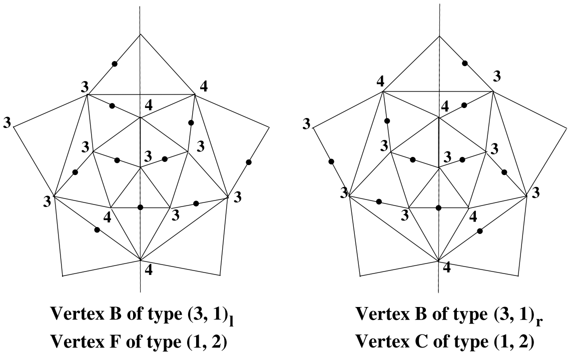
Fig. 8 Mirrored start configurations when the bead distribution $(3, 1)$ is placed at vertex A (configurations 3 and 4 in Table 1). The axis of mirror symmetry is the *dashed blue line* through vertices A, B, J, and L



(a) **Vertex B of type $(1, 2)_l$** (b) **Vertex B of type $(1, 2)_r$**

of type $(5, 0)$. The essential features of this icosahedral bead configuration are encoded in Fig. 7(b), where each vertex is labeled by the number of beads placed on its incident edges. This start configuration (without further replacements of 5-junctions) requires four DNA single strands.

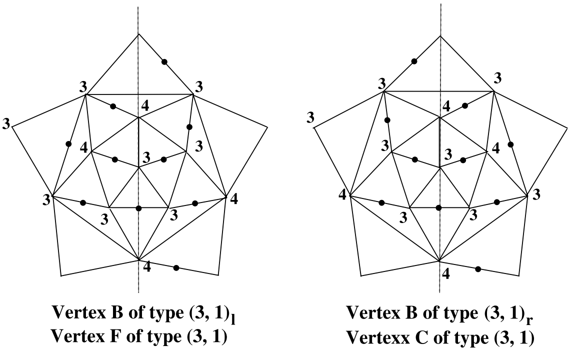
2. Place a $(3, 1)$ pentagon configuration at any vertex in any orientation. Choose vertex A, say, and place this $(3, 1)_{\mathrm{A}}$ configuration such that the edge linking vertices A and B hosts a bead. This fixes vertices D, E, and J to have $(1, 2)$ configurations. Vertex B can be in a $(3, 1)$ or a $(1, 2)$ configuration. The former yields that the minimal number bead distribution makes the corresponding thickened graph non-orientable, as one triangular face acquires two beads. The configuration $(1, 2)$ can be placed on vertex B in two orientations: the 'left' orientation $(1, 2)_{\ell,\mathrm{B}}$ shows a bead on the edge linking B and G, while the 'right' orientation $(1, 2)_{r,\mathrm{B}}$ shows a bead on the edge linking B and H. Note that B lies on the axis connecting A, B, and J and a bead on the edge connecting B and G is a reflection of the bead placed on the edge connecting B and H, therefore, $(1, 2)_{l,\mathrm{B}}$ and $(1, 2)_{r,\mathrm{B}}$ yield equivalent bead configurations (see Fig. 8).
3. Place a $(1, 2)$ pentagon configuration at any vertex in any orientation. Choose vertex A, say, and place this $(1, 2)_{\mathrm{A}}$ so that the edge linking vertices D and E hosts a bead. Then vertex B must be in a $(5, 0)$ or a $(3, 1)$ configuration since two edges on the corresponding pentagon perimeter host beads. If B is of type



Vertex B of type (3, 1)$_l$
Vertex F of type (1, 2)

Vertex B of type (3, 1)$_r$
Vertex C of type (1, 2)

Fig. 9 Mirrored start configurations when the bead distribution $(1, 2)$ is placed at vertex A, with either $(3, 1)_{\ell,\mathrm{B}}$ and $(1, 2)_{\mathrm{F}}$ (configuration 6 in Table 1), or with $(3, 1)_{r,\mathrm{B}}$ and $(1, 2)_{\mathrm{C}}$ (configuration 8 in Table 1). The axis of mirror symmetry is the *dashed blue line* through vertices A, B, J, and L



Vertex B of type (3, 1)$_l$
Vertex F of type (3, 1)

Vertex B of type (3, 1)$_r$
Vertexx C of type (3, 1)

Fig. 10 Mirrored start configurations when the bead distribution $(1, 3)$ is placed at vertex A, with either $(3, 1)_{\ell,\mathrm{B}}$ and $(3, 1)_{\mathrm{F}}$ (configuration 7 in Table 1), or with $(3, 1)_{r,\mathrm{B}}$ and $(3, 1)_{\mathrm{C}}$ (configuration 9 in Table 1). The axis of mirror symmetry is the *dashed blue line* through vertices A, B, J, and L

$(5, 0)$, one arrives at the same configuration as discussed in case 1 (Fig. 7 (b)). If B is $(3, 1)$, the configuration can be placed in two orientations: $(3, 1)_{\ell,\mathrm{B}}$ if the edge linking B and G hosts a bead, and $(3, 1)_{r,\mathrm{B}}$ if the edge linking B and H hosts a bead. Once the orientation is chosen, there is one more choice to make. In the 'left' configuration, vertices C and D are automatically in $(1, 2)$ configurations but vertex F can be of type $(1, 2)$ or $(3, 1)$. Each choice leads to a start configuration which requires two DNA strands. In the 'right' configuration, vertices E and F are automatically in $(1, 2)$ configurations but vertex C can be of type $(1, 2)$ or $(3, 1)$. Each choice once again leads to a start configuration which requires two DNA strands. These configurations are shown in Figs. 9 and 10. Note that two of these four start configurations are mirror images of the other two.

These considerations are summarized in Table 1.

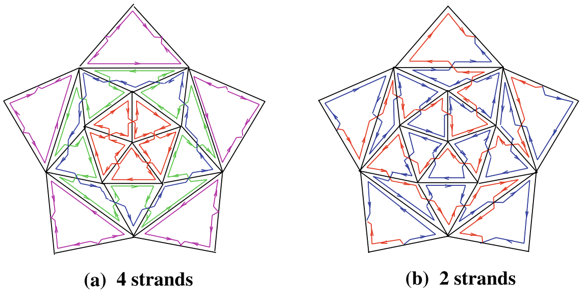
The result is that up to mirror symmetry, one has four inequivalent classes. One admits four strands (configurations 1 and 5 are equivalent under a rotation of the icosahedron), and the other three require two strands. Configurations 3 and 4 are mirror-symmetric, as are configurations 6 and 8, as well as 7 and 9.

The final step in the construction of such cages is to merge the different strands by using new types of junctions. The start configuration 1 corresponds to the four-strand cage of Fig. 11(a). The figure shows that the two vertices (A and L, compare with the labeling in Fig. 7(a)) host thickened junctions of type A_5, containing five

Table 1 Summary of possible start configurations for the icosahedral cages prior to the identification of the subset of symmetry-inequivalent configurations

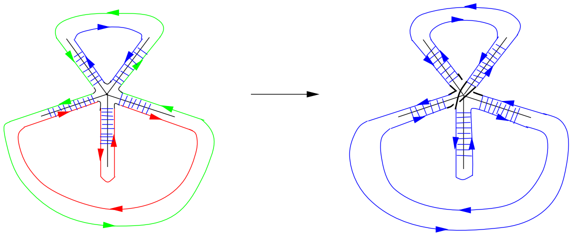
Systematics of minimal bead configurations								
$(5,0)_{\mathrm{A}}$		$(3,1)_{\mathrm{A}}$		$(1,2)_{\mathrm{A}}$				
$(1,2)_{\mathrm{B}}$		$(1,2)_{\mathrm{D}}$		$(5,0)_{\mathrm{B}}$	$(3,1)_{\ell,\mathrm{B}}$		$(3,1)_{r,\mathrm{B}}$	
$(1,2)_{\mathrm{C}}$		$(1,2)_{\mathrm{E}}$		$(1,2)_{\mathrm{C}}$	$(1,2)_{C}$		$(1,2)_{\mathrm{E}}$	
$(1,2)_{\mathrm{D}}$		$(1,2)_{\mathrm{J}}$		$(1,2)_{\mathrm{D}}$	$(1,2)_{\mathrm{D}}$		$(1,2)_{\mathrm{F}}$	
$(1,2)_{\mathrm{E}}$	$(3,1)_{\mathrm{B}}$	$(1,2)_{\ell,\mathrm{B}}$	$(1,2)_{r,\mathrm{B}}$	$(1,2)_{\mathrm{E}}$	$(1,2)_{\mathrm{F}}$	$(3,1)_{\mathrm{F}}$	$(1,2)_{\mathrm{L}}$	$(3,1)_{\mathrm{C}}$
$(1,2)_{\mathrm{F}}$	X	$(1,2)_{\mathrm{C}}$	$(3,1)_{\mathrm{C}}$	$(1,2)_{\mathrm{F}}$	$(3,1)_{\mathrm{E}}$	$(1,2)_{\mathrm{E}}$	$(3,1)_{\mathrm{D}}$	$(1,2)_{\mathrm{D}}$
$(1,2)_{\mathrm{G}}$	X	$(3,1)_{\mathrm{F}}$	$(1,2)_{\mathrm{F}}$	$(1,2)_{\mathrm{G}}$	$(1,2)_{\mathrm{G}}$	$(1,2)_{\mathrm{G}}$	$(3,1)_{\mathrm{G}}$	$(1,2)_{\mathrm{G}}$
$(1,2)_{\mathrm{H}}$	X	$(1,2)_{\mathrm{G}}$	$(1,2)_{\mathrm{G}}$	$(1,2)_{\mathrm{H}}$	$(3,1)_{\mathrm{H}}$	$(1,2)_{\mathrm{H}}$	$(1,2)_{\mathrm{H}}$	$(1,2)_{\mathrm{H}}$
$(1,2)_{\mathrm{I}}$	X	$(1,2)_{\mathrm{H}}$	$(1,2)_{\mathrm{H}}$	$(1,2)_{\mathrm{I}}$	$(1,2)_{\mathrm{I}}$	$(3,1)_{\mathrm{I}}$	$(1,2)_{\mathrm{I}}$	$(1,2)_{\mathrm{I}}$
$(1,2)_{\mathrm{J}}$	X	$(3,1)_{\mathrm{I}}$	$(1,2)_{\mathrm{I}}$	$(5,0)_{\mathrm{J}}$	$(3,1)_{\mathrm{J}}$	$(3,1)_{\mathrm{J}}$	$(3,1)_{\mathrm{J}}$	$(3,1)_{\mathrm{J}}$
$(1,2)_{\mathrm{K}}$	X	$(1,2)_{\mathrm{K}}$	$(3,1)_{\mathrm{K}}$	$(1,2)_{\mathrm{K}}$	$(1,2)_{\mathrm{K}}$	$(1,2)_{\mathrm{K}}$	$(1,2)_{\mathrm{K}}$	$(3,1)_{\mathrm{K}}$
$(5,0)_{\mathrm{L}}$	X	$(3,1)_{\mathrm{L}}$	$(3,1)_{\mathrm{L}}$	$(1,2)_{\mathrm{L}}$	$(1,2)_{\mathrm{L}}$	$(1,2)_{\mathrm{L}}$	$(1,2)_{\mathrm{L}}$	$(1,2)_{\mathrm{L}}$
1	(2)	3	4	5	6	7	8	9
4 strds		2 strds	2 strds	4 strds	2 strds	2 strds	2 strds	2 strds

Fig. 11 (**a**) Four-strand start configuration 1, (**b**) two-strand start configuration 6


(a) 4 strands

(b) 2 strands

Fig. 12 Replacement of a three-strand configuration by a one-strand configuration at a type $B_{2,5}$ junction



strands. All other vertices have thickened junctions of type $B_{2,5}$, i.e., 5-junctions with two cross-overs of the strands at the vertices. These junctions involve three different strands, and via a replacement of strand connections as depicted in Fig. 12 these strands can be merged into one. If such a replacement takes place at vertex

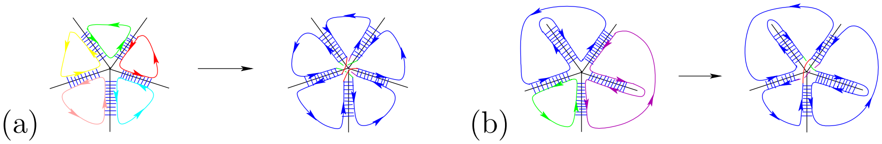


Fig. 13 The junctions used as replacements for A_5 vertices

B say (replacing the junction of type $B_{2,5}$), then the resulting cage is made of two distinct strands. The results in [8] show that the parity of the boundary components in every thickened graph of a given polyhedron is always the same. Therefore, two strands are the least number of strands that can be used in the construction of an icosahedron.

For the case when an icosahedron has an edge length with an even number of helical half-turns, a start configuration is given by 20 strands of DNA tracing the polyhedral faces. These can be reduced to two strands in a number of ways. One possibility is to use the replacements shown in Fig. 13, which result in (a) a reduction of five distinct strands meeting at a vertex to one, and (b) a reduction of three different loops to one. Via three replacements of type (a) and two of type (b), one obtains a two-strand realization of the cage.

4 The Dodecahedral Cage

Next, we consider the case of the dodecahedron, a three-coordinated polyhedron with 12 pentagonal faces. We start with the case where the edge length of the dodecahedral cage is such that the DNA duplex has an odd number of half-turns along each edge, and we encode this additional helical turn via a red cross-over on a dodecahedral cage in planar projection (see Fig. 14(a)). Since the resulting number of twists along a pentagonal face is odd, the surface corresponding to the ribbon representing the double helical structure of the DNA is non-orientable. We follow the procedure outlined in Sect. 2 to produce orientable embeddings that may serve as start configurations for our analysis. In particular, we replace A_3 junctions by $B_{1,3}$ junctions in order to compensate for the twist, and determine their locations via the bead rule.

As there are twelve faces in the dodecahedron, the minimal number of beads needed to provide an orientable embedding of the whole double helical structure is six. There are several non-equivalent ways to introduce six beads by tessellating the surface of the dodecahedron with the tile shown in Fig. 14(b). The inequivalent possibilities of placing beads on the edges of the dodecahedron such that precisely one edge per pentagonal face is decorated by a bead are obtained as follows. Without loss of generality, we place the first bead on any of the edges, because they are all equivalent by symmetry. We label this edge, and hence the bead on this edge, as 1; see Fig. 15. To keep the number of beads at a minimum, no other edge of the two pentagons labeled A and B can have a bead.

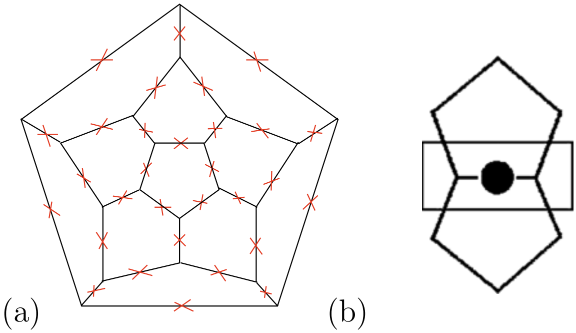


Fig. 14 (**a**) A dodecahedral cage in planar projection, with cross-over schematically representing the twists that occur along the edges as a result of an odd number of half-turns in the helical structure. (**b**) Tiles needed to make the structure orientable

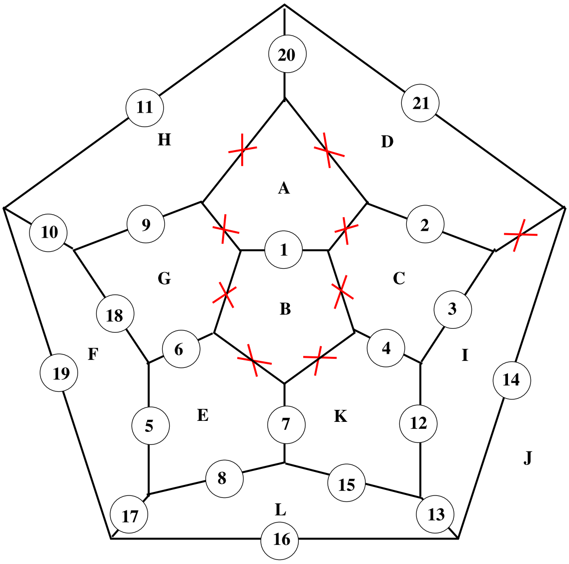


Fig. 15 Labeling system corresponding to Fig. 16. *Symbols* (letters, numbers) identifying the pentagons follow the first branch of the solution tree in Fig. 16

Since the order in which pentagons are considered does not matter, we can next concentrate on the pentagon labeled C. Edges labeled 2 and 4 are equivalent by symmetry, and hence lead to equivalent configurations. We therefore consider only the possibility of placing a bead on edge 2 or edge 3, and treat each case separately. A bead on edge 2 ‘covers’ pentagons C and D in Fig. 15. Since every pentagonal face has to have a bead on one of its edges, we can, without loss of generality, look at pentagon E next. Due to the beads already present there is no symmetry, and thus all edges 5 to 8 lead to different solutions. There are 10 solutions in total, denoted S1 to S10 in Fig. 16. We next concentrate on edge 3 instead of edge 2, hence pentagons A, B, C, and I contain an edge with a bead. Up to symmetry arguments, there are three different ways to complete this analysis based on the bead rule, labeled S11 to S13 in Fig. 16.

We investigate whether any of these 13 solutions can be symmetrically mapped onto another. To do so, we observe that different bead configurations can be classi-

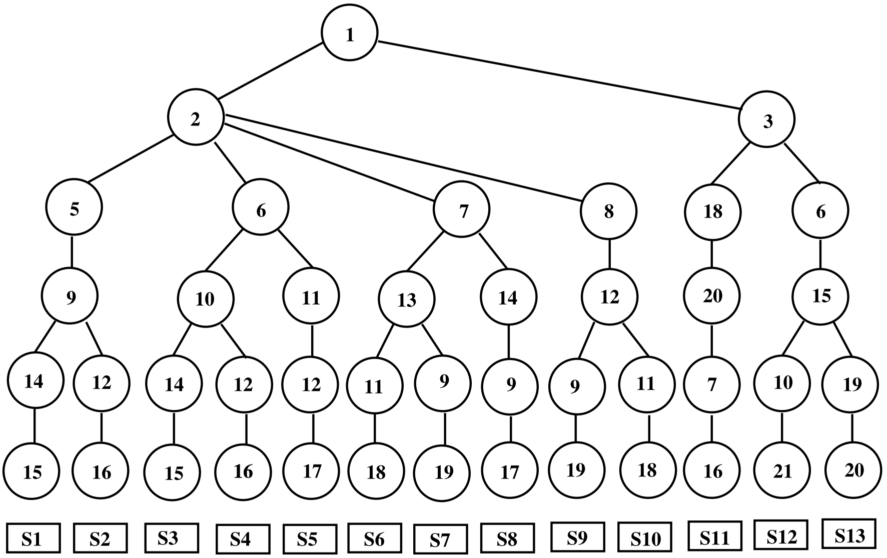


Fig. 16 The tree encoding the 13 solutions of placing six beads on the dodecahedral edges such that each pentagon has precisely one bead. Only solutions S1, S2, S3, S6, and S11 are inequivalent

Table 2 Summary of possible bead paths for the identification of the symmetry-inequivalent start configurations

	Systematics of bead paths
Type I	all beads lie on a closed (cyclic) bead path (corresponding to solutions S1, S2, S4, S9)
Type II	all beads lie on a non-cyclic (open) bead path (corresponding to solutions S3, S5, S7, S8, S10, S12)
Type III	all beads lie on three disjoint bead paths with two beads each (corresponding to solutions S6, S13)
Type IV	all beads lie on six disjoint bead paths one edge in length (corresponding to solution S11)

fied according to different *bead paths*: We call any path along edges of the decorated dodecahedron a bead path if, starting and ending on a beaded edge, it alternates along non-beaded and beaded edges. The four possible scenarios of bead paths are summarized in Table 2.

Bead placements corresponding to a given type (I–IV) are inequivalent to arrangements of any other type, because any symmetry would map a path onto another path, and a cycle onto another cycle. Hence, an equivalence may appear only within a given type. We start with type I. Solutions S1 and S9 are mirror images of each other (obtained by turning the sphere inside out, i.e., they have different helical structures but are otherwise identical). We therefore consider them as equivalent. Solutions S2 and S4 can be considered equivalent as they correspond to the closed path that divides the 12 pentagons of the dodecahedron into two identical sets of six. However, S1 and S2 are not related by symmetry, and there are hence two distinct solutions of type I, represented by S1 and S2. Solutions S3, S5, and S8 represent mutually symmetric open paths, while S7, S10, and S12 represent mutually symmetric paths, but of opposite orientation to S3, S5, and S8 (again obtained by turning the sphere inside out). Hence, we can consider all these options as equivalent, and so there is only one (up to symmetry) arrangement of beads producing a non-cyclic path, represented by S3. The two arrangements of type III corresponding to solutions S6 and S13 can be mapped one onto another. So, there is only one solution of this type, represented, say by S6. Finally, S11 is the only solution of type IV. It corresponds to an equidistant distribution of beads on edges and is in that sense the 'most symmetric' solution.

Every bead configuration translates into a start configuration. The five inequivalent start configurations are depicted in Fig. 17. Each of these DNA embeddings has 14 vertex configurations of type A_3 and 6 vertex configurations of type $B_{1,3}$. The DNA embeddings corresponding to solutions S2 and S11 result in six distinct strands (Fig. 17(d, e)), the embeddings corresponding to solutions S1 and S3 in four strands (Fig. 17(b, c)), while the embedding corresponding to solution S6 in two strands (Fig. 17(a)).

In order to realize the cage in terms of a minimal number of DNA strands, the configurations in Fig. 17(b–d) require further vertex replacements. Note that as in the case of the icosahedron, (by the results in [8]) two is the least number of strands that can be used to obtain the dodecahedral cage. Indeed, if a vertex configuration of type A_3 obtained by hybridizing three separate strands is replaced by a vertex configuration of a branch point of type $B_{1,3}$, then the number of strands meeting at the vertex reduces to one according to [8]. In the case of six separate strands (i.e., the cases of S2 and S11), we need to choose two vertices for replacement, while for the case of four separate strands (i.e., the cases of S1 and S3), only one replacement is needed. The reader is referred to [7] for details. In all cases, the minimal number of strands to assemble a dodecahedron using only junctions of type A_3 and $B_{1,3}$ is two. Figure 18(a) shows the two-strand configuration corresponding to solution S6.

We now consider the case of double-stranded DNA embeddings into dodecahedral cages with *an even number of half-turns per edge*. In these cases, there are no additional twists on the edges. A start configuration is therefore easily obtained by placing 12 separate strands into the 12 faces of the dodecahedron. Via $n = 5$ replacements, one again obtains a DNA embedding with two separate strands. According to the remark at the end of Sect. 2 (see also details in [7]), these replacements can be

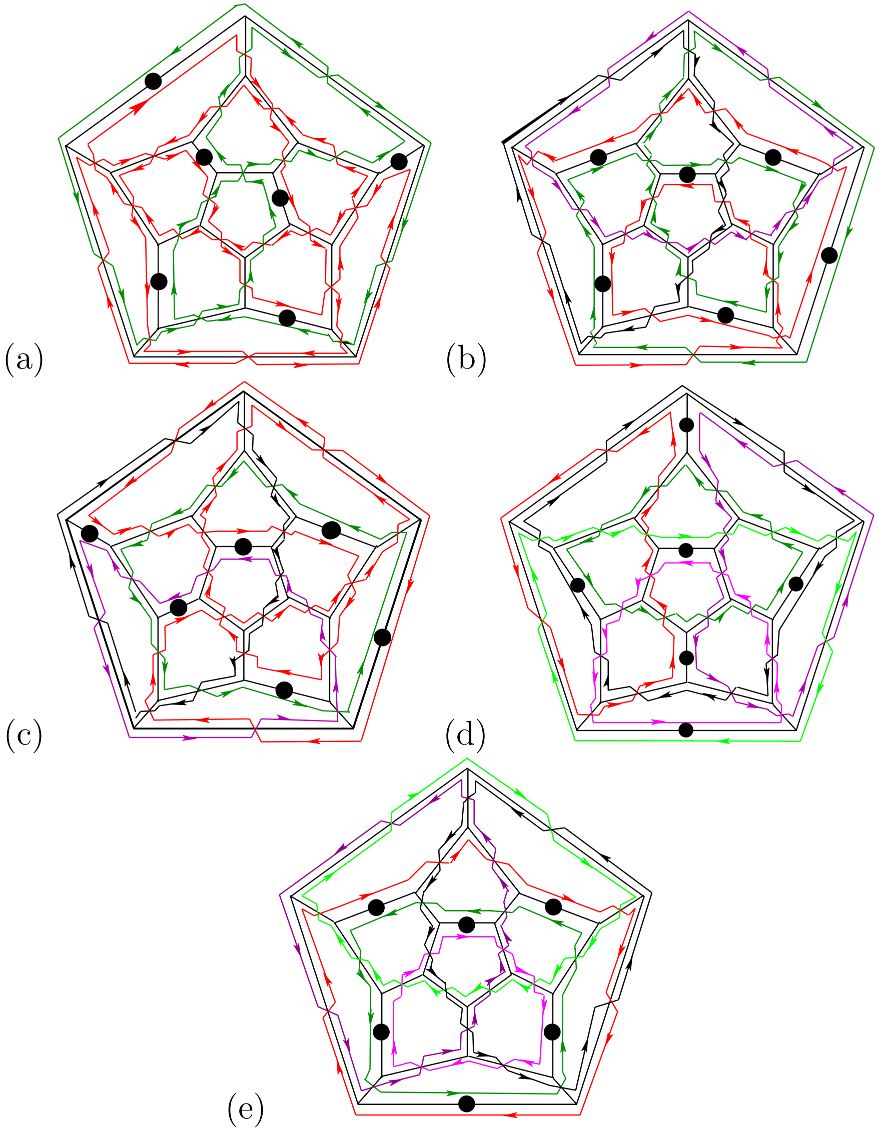


Fig. 17 Embeddings of the DNA duplex structures in a dodecahedral cage corresponding to the inequivalent bead configurations. The bullets indicate the placements of beads, i.e., the edge containing an even number of half-turns. The configurations correspond to (**a**) solution S6 with two separate strands; (**b**) solution S1 with four separate strands; (**c**) solution S3 with four separate strands; (**d**) solution S11 with six separate strands; and (**e**) solution S2 with six separate strands

freely chosen if flexible strand connections are used, and require particular positions in the non-flexible case.

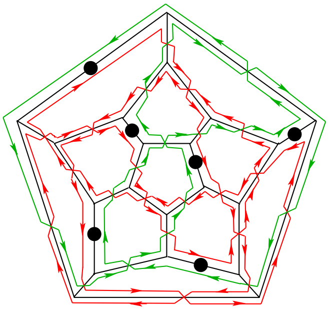


Fig. 18 Embedding of a duplex DNA structure corresponding to the solution S6 with two strands

5 The Icosidodecahedral Cage

We now consider the icosidodecahedron, a 32-faced polyhedron with quatro-valent vertices. We again start with the case where all edges accommodate an odd number of helical half-turns, which implies additional cross-overs on the edges in the initial configuration. According to Sect. 2, the bead rule has to be applied in order to obtain a start configuration. The minimal number of beads required is easily calculated. All faces of the icosidodecahedron have an odd number of edges: there are 12 pentagons and 20 triangles. However, each face must have an even number of cross-overs to keep the orientability. Therefore, each of the 20 triangles must receive at least one bead. However, by placing a bead on each triangle, one also places at least one bead on each pentagon. The distribution of this minimum number of 20 beads should be such that pentagons receive an odd number of beads. Let α be the number of pentagons receiving one bead, β be the number of pentagons receiving three beads and γ be the number of pentagons receiving five beads. Given that there are 12 pentagons in total, we must satisfy the two equations

$$\alpha + 3\beta + 5\gamma = 20,$$
$$\alpha + \beta + \gamma = 12,$$

with α, β, and γ positive integers. There are three solutions to the problem, namely

Case I	$\alpha = 8,$	$\beta = 4,$	$\gamma = 0,$
Case II	$\alpha = 9,$	$\beta = 2,$	$\gamma = 1,$
Case III	$\alpha = 10,$	$\beta = 0,$	$\gamma = 2.$

We start by considering case I. This tells us that the bead rule is fulfilled if there are four pentagons with three beads each, and if every triangle has precisely one bead. We therefore determine all symmetry-inequivalent start configurations with that property. Since this is a significant combinatorial task for the polyhedron under consideration, our analysis is computer-assisted.

In a first instance, we use the icosahedral symmetry to reduce the number of options to be considered. In particular, we determine all symmetry-inequivalent distributions of four pentagonal faces on the icosidodecahedron. Each of these four

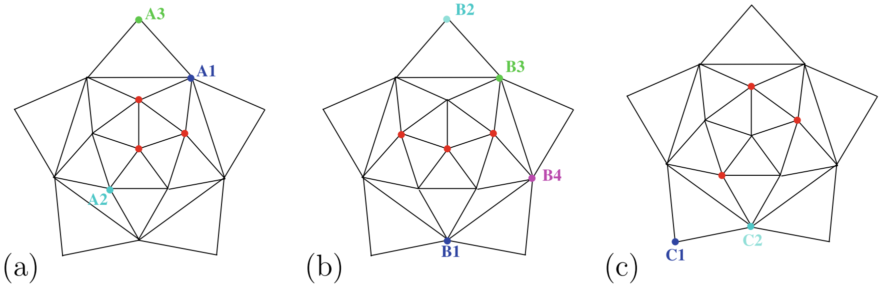


Fig. 19 Partial start configurations for case I, with a distribution of four out of twelve pentagons on the icosidodecahedron, represented here as distributions of four vertices on an icosahedron (**a**) configurations with three vertices being those of a *triangle* (*red*) and the fourth vertex being either A_1, A_2, or A_3, (**b**) configurations with three *red* vertices and the fourth one being either B_1, B_2, B_3, or B_4, (**c**) configurations with three red vertices and the fourth one being either C_1 or C_2

Table 3 The distribution of 10-, 12-, 14-, and 16-loop configurations among the three different cases

Loop number	Case I	Case II	Case III
10	11527	0	0
12	343	951	0
14	3	8	73
16	0	0	1

faces has three of its edges decorated by one bead, whilst all other pentagons and all triangles have only one bead on their perimeter. In order to determine all inequivalent configurations of four pentagons, we consider the equivalent problem of finding all different possibilities of coloring four of the 12 vertices of an icosahedron.

There are 9 inequivalent such configurations for case I, which we call the *partial start configurations*. We show them schematically in a projective view of the icosahedron in Fig. 19.

We next determine the inequivalent bead configurations for each partial start configuration. Each partial start configuration encodes several possible cage scenarios which correspond to all inequivalent ways of placing three beads on the edges of four distinguished pentagons, and one bead on one of the edges of all other faces (pentagonal or triagonal). We carry out this combinatorial task computationally via a computer program that tests, for each start configuration, which combinations of beads are possible, given the fact that the four distinguished pentagonal faces each have three beads, while all others have one. The results for all three cases are summarized in Table 3, which indicates the number of different configurations with a given number of loops in each of these cases.

The computations show that the smallest number of distinct loops is 10 and the largest number 16. While there are over 10^4 distinct 10-loop configurations, there is only one configuration with 16 loops, occurring in case III. All vertex configurations involve either four, three, or two distinct loops. As before, these can at best be reduced to two-strand configurations because the number of distinct circular DNA

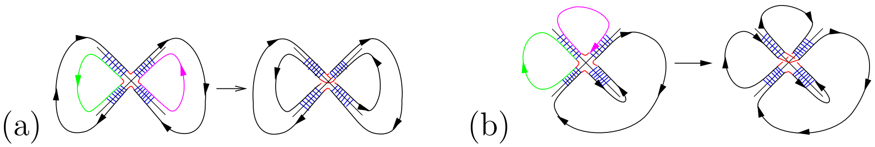


Fig. 20 Replacements of a 3-loop configuration by a single loop: (**a**) type I and (**b**) type II

strands is even in all cases. Such a reduction is possible via the replacements of type I and type II shown in Fig. 20.

If the edge length of the icosidodecahedron is such that none of the edges has an additional cross-over, the start configuration consists of 32 loops corresponding to the 32 faces of the polyhedron. Via 30 replacements of type I or type II, a duplex cage structure realized by two circular DNA molecules is obtained.

6 Discussion

We have performed a theoretical analysis of different types of icosahedral cages with minimal (two) DNA strands such that each edge is in a duplex structure. Focus was placed on the three icosahedrally symmetric polyhedra with the smallest number of vertices, the icosahedron with 12 vertices at the 5-fold axes of icosahedral symmetry, the dodecahedron with 20 vertices at the 3-fold axes, and the icosidodecahedron with 30 vertices at the 2-fold axes. These polyhedra are distinguished by the fact that they have uniform edge lengths and may therefore be easier to manufacture than other polyhedra with icosahedral symmetry.

We remark that polyhedral RNA cages have been observed also within the protein containers, called viral capsids that encapsulate, and hence provide protection for the viral genome [11, 12]. However, the RNA has to be unknotted for successful replication and, therefore, these cages usually do not appear in the form we have described here and are realized by the viral RNA in a different way [13, 14].

A comparative analysis reveals interesting features of the cage structures. In particular, for each of them, the minimal number of circular molecules needed to construct the cage is two. The icosidodecahedral cage is distinguished by the fact that its volume per surface ratio is the largest among the polyhedra considered here (and is also larger than those of the cages that have been realized experimentally to date). It may therefore be more suitable for nanotechnology applications in which the cages serve as containers for storage or the transport of a cargo.

We hope that the blueprints for the organization of the cages with a non-crystallographic symmetry suggested here may assist in their experimental realization. In particular, these blueprints suggest the structures of the junction molecules that may be used as basic building blocks for the self-assembly of those cages along the lines of [15, 16].

Acknowledgements AT has been supported by an EPSRC Springboard Fellowship, and RT by an EPSRC Advanced Research Fellowship. Some of this work has been carried out at the University of South Florida, and both AT and RT would like to thank the Department of Mathematics for their hospitality. NJ has been supported in part by the NSF grants CCF #0523928 and CCF #0726396.

References

1. Chen JH, Seeman NC (1991) Synthesis from DNA of a molecule with the connectivity of a cube. Nature 350:631–633
2. Goodman RP, Schaap IAT, Tardin CF, Erben CM, Berry RM, Schmidt CF, Turberfield AJ (2005) Rapid chiral assembly of rigid DNA building blocks for molecular nanofabrication. Science 310:1661–1665
3. Shih WM, Quispe JD, Joyce GF (2004) A 1.7-kilobase single-stranded DNA that folds into a nanoscale octahedron. Nature 427:618–621
4. Zhang Y, Seeman NC (1994) The construction of a DNA truncated octahedron. J Am Chem Soc 116:1661–1669
5. Destito G, Singh P, Koudelka KJ, Manchester M (2007) Assembling viral nanoparticles for vascular imaging and tumor-specific targeting. In: Foundations of nanoscience, self-assembled architectures and devices, proceedings of FNANO07, pp 2–4
6. Grayson NE, Taormina A, Twarock R (2009) DNA duplex cage structures with icosahedral symmetry. J Theor Comp Sci 410(15):1440–1447
7. Jonoska N, Twarock R (2008) Blueprints for dodecahedral DNA cages. J Phys A 41:304043–304057
8. Jonoska N, Saito M (2002) Boundary components of thickened graphs. In: Jonoska N, Seeman NC (eds) DNA7. Lecture notes in computer science, vol 2340. Springer, Heidelberg, p 70
9. Greenberg MJ, Harper JR (1981) Algebraic topology. Benjamin/Cummings, Redwood City
10. Ho PS, Eichman BF (2001) The crystal structures of DNA holliday junctions. Curr Opin Struct Biol 11:302–308
11. Tang L, Johnson KN, Ball LA, Lin T, Yeager M, Johnson JE (2001) The structure of pariacoto virus reveals a dodecahedral cage of duplex RNA. Nat Struct Biol 8:77–83
12. van den Worm SHE, Koning RE, Warmenhoven HJ, Koerten HK, van Duin J (2006) Cryo electron microscopy reconstructions of the Leviviridae unveil the densest icosahedral RNA packing possible. J Mol Biol 363:858–865
13. Toropova K, Basnak G, Twarock R, Stockley PG, Ranson NA (2008) The three-dimensional structure of genomic RNA in bacteriophage MS2: implications for assembly. J Mol Biol 375:824–836
14. Rudnick J, Bruinsma R (2005) Icosahedral packing of RNA viral genomes. Phys Rev Lett 94:038101–038104
15. Sa-Ardyen P, Jonoska N, Seeman NC (2004) Self-assembly of irregular graphs whose edges are DNA helix axes. J Am Chem Soc 126:6648–6657
16. Sa-Ardyen P, Jonoska N, Seeman NC (2003) Self-assembling DNA graphs. Nat Comput 2:427–438

Applying Symmetric Enumeration Method to One-Dimensional Assembly of Rotatable Tiles

Satoshi Kobayashi

Abstract Motivated by the increasing importance of the analysis of a complicated reaction system where molecules are interacting in various ways to produce a huge number of compounds of molecules, the author's previous work proposed a new approach, called Symmetric Enumeration Method (SEM), to the efficient analysis of such reaction systems. The proposed theory provided a general method for the efficient computation of equilibrium states. In this paper, we will review the results of the theory of SEM, and apply it to the equilibria analysis of a one-dimensional assembly system of tiles which can be rotated around three axes, i.e., the x-, y-, and z-axes. Although the growth of a tile assembly is restricted to only one-dimensional direction, the exhaustive method of generating all tile assemblies and obtaining equilibria is intractable since the number of assemblies could be exponential with respect to the number of tiles given to the system. We will show that this equilibria analysis can be transformed into a convex programming problem with a set of variables of size polynomial with respect to the number of input tiles.

1 Introduction

Since the pioneering work by Adleman [1], the paradigm of *DNA computing* (in a broad sense, *molecular computing*) has emerged and attracted much attention from computer scientists, molecular biologists, DNA nanotechnologists, etc. [3, 11, 13–15]. The principle of DNA computing paradigm essentially relies on the DNA hybridization process, but it is in essence error-prone [4]. Therefore, it is substantially important to design a set of DNA sequences or tiles with which we can obtain a maximum concentration of a target molecular architecture. In order to evaluate a given set of DNA sequences or tiles in this respect, we need to devise a methodology for efficiently computing the concentration of the target assembly at the equilibrium state.

Motivated by the increasing importance of the analysis of Hybridization Reaction Systems (HRSs, for short), the author proposed a new approach to the efficient analysis of equilibrium state of HRSs by overcoming the combinatorial explosion problem of resultant assemblies [8–10]. The proposed theory provides a *general*

S. Kobayashi (✉)
Department of Computer Science, University of Electro-Communications, 1-5-1, Chofugaoka, Chofu, Tokyo 182-8585, Japan
e-mail: satoshi@cs.uec.ac.jp

A. Condon et al. (eds.), *Algorithmic Bioprocesses*, Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_10, © Springer-Verlag Berlin Heidelberg 2009

method for the efficient computation of equilibrium states. Thus, it can be applied to various kinds of HRSs other than those of DNA and RNA molecules. As far as the author's knowledge, this was the first attempt to formulate such a general theory for computing equilibrium state of combinatorially complex hybridization reaction systems.

The key idea exists in the *locality* of HRSs. By locality, we intuitively mean the physical property that the free energy of an assembly X of molecules can be computed as the sum of free energies of all local substructures of X. For instance, the free energy of a single RNA and DNA molecule at the secondary structure level can be calculated as the sum of free energies of all local substructures such as hairpin loops, bulge loops, internal loops, multiple loops, etc. The proposed theory gave a theoretical formulation of locality of HRSs using graph theory. It defines an HRS with high locality as the one having the following properties: (1) there exists a weighted directed hypergraph G with initial and final vertices such that the set of hyperpaths from initial vertices to final vertices is in many-to-one correspondence with the set of assemblies of molecules, (2) the weight of a hyperpath is equivalent to the free energy of its corresponding assembly, (3) the hypergraph G has some symmetric structures which capture the symmetric property of the space of assemblies. In this formulation, a hyperarc of a hyperpath can be regarded as a substructure of its corresponding assembly. With this key concept of locality of HRSs, the paper established a general theory for computing equilibrium state of HRSs.

In this paper, we will review the results of the theory of *Symmetric Enumeration Method* [8–10], and apply it to the equilibria analysis of one-dimensional assembly system of tiles which can be rotated around three axes, i.e., x, y, z-axes. Although the growth of a tile assembly is restricted in only one-dimensional direction, the exhaustive method of generating all tile assemblies and obtaining its equilibria is intractable since the number of assemblies could be exponential with respect to the number of tiles given to the HRS. We will show that this equilibria analysis can be transformed into a convex programming problem with a set of variables of size polynomial with respect to the number of input tiles.

Adleman's work [2] on the equilibrium state analysis of linear tile assembly is related to our work. There are two main and important different points between these works: (1) although Adleman's analysis is based on a probabilistic model of chemical reactions, we rely on the concentration based model, and (2) Adleman's analysis focuses on linear tile assembly in which rotation of tiles are not allowed and is simpler than the assembly system discussed in this paper.

Dirks et al. proposed an interesting method for computing equilibria of interacting RNA molecules by using the dynamic programming method and the convex programming method [5]. The symmetric enumeration method is a very general framework which can be applied to various hybridization reaction systems, and is based on a totally different idea from their approach and uses only convex programming method.

After providing the definition of HRSs in Sect. 2, we will give the theory of symmetric enumeration method in Sect. 3. Section 4 gives the definition of tile assembly system and its important properties. In Sect. 5, we will give three different implementations of symmetric enumeration schemes for the tile assembly system.

2 Equilibrium of Hybridization Reaction System

For a set N of numbers, by N_+ and N_{++}, we denote the subsets of N consisting of all nonnegative and positive numbers in N, respectively. By $\mathbf{R}$ and $\mathbf{Z}$, we denote the set of real numbers and integers, respectively.

Let $\mathcal{M}$ be a set of *molecules* and $\mathcal{A}$ be a set of *assemblies of molecules* consisting of molecules in $\mathcal{M}$. For $x \in \mathcal{M}$ and $X \in \mathcal{A}$, by $\#_x(X)$, we denote the number of molecules x contained in an assembly X. A *reaction rule over* $\mathcal{A}$ is given by a pair of $\mathcal{X}_1$ and $\mathcal{X}_2$ of finite multisets consisting of elements of $\mathcal{A}$ such that the following equation holds:

$$\sum_{X \in \mathcal{X}_1} \#_x(X) = \sum_{X \in \mathcal{X}_2} \#_x(X) \quad (\forall x \in \mathcal{M}). \tag{1}$$

Note that the sum over a multiset counts elements redundantly for their multiple occurrences. This equality constraint (1) corresponds to the law of conservation of each molecule. A reaction rule $(\mathcal{X}_1, \mathcal{X}_2)$ is usually denoted by $\mathcal{X}_1 \rightleftharpoons \mathcal{X}_2$. In case of $\mathcal{X}_1 = \{X_1, \ldots, X_{n_1}\}$ and $\mathcal{X}_2 = \{Y_1, \ldots, Y_{n_2}\}$, where multiple occurrences of assemblies are allowed, we often write:

$$X_1 + \cdots + X_{n_1} \rightleftharpoons Y_1 + \cdots + Y_{n_2}.$$

A *distribution* of a set $\mathcal{U}$ is a function from $\mathcal{U}$ to $\mathbf{R}_+$. Usually, we use notations, $[\ ]$, $[\ ]_1$, $[\ ]_2, \ldots$, etc., for representing distributions. For example, for a distribution $[\ ]$ of $\mathcal{A}$ and an assembly $X \in \mathcal{A}$, $[X]$ represents a concentration of the assembly X. A *distribution* of $\mathcal{M}$ is especially called an *initial distribution*. If we have a set $\mathcal{M}$ of molecules with its initial distribution $[\ ]_0$, then any distribution $[\ ]$ of $\mathcal{A}$ should satisfy the following equation:

$$\sum_{X \in \mathcal{A}} \#_x(X) \cdot [X] = [x]_0 \quad (\forall x \in \mathcal{M}). \tag{2}$$

This equality constraint corresponds to the law of conservation of each molecule.

For instance, let us consider two molecules α and β, and an assembly $\alpha\beta$ consisting of molecules α and β. Note that each of α and β is itself an assembly consisting of only one molecule. Thus, we can consider a reaction rule $\alpha + \beta \rightleftharpoons \alpha\beta$. Equilibrium state of this reaction rule is determined by free energies $E(\alpha)$, $E(\beta)$, and $E(\alpha\beta)$ of assemblies α, β, and $\alpha\beta$, respectively. More precisely, the distribution $[\ ]$ at the equilibrium state should satisfy:[1]

$$\frac{[\alpha\beta]}{[\alpha][\beta]} = e^{-(E(\alpha\beta)-(E(\alpha)+E(\beta)))}.$$

[1] In this paper, the free energy $E(X)$ of X is a dimensionless quantity, i.e., $E(X)$ is the free energy per mol of X divided by the physical quantity $k_B T$, where k_B is Boltzmann constant and T is absolute temperature of the reaction system.

We will give the definition of this kind of *equilibrium equation* in a general setting. For a reaction rule $\mathcal{X}_1 \rightleftharpoons \mathcal{X}_2$, its equilibrium equation is given by

$$e^{\sum_{X \in \mathcal{X}_1} E(X)} \times \prod_{X \in \mathcal{X}_1} [X] = e^{\sum_{X \in \mathcal{X}_2} E(X)} \times \prod_{X \in \mathcal{X}_2} [X]. \tag{3}$$

In summary, a hybridization reaction system (HRS, for short) is defined by $P = (\mathcal{M}, \mathcal{A}, \{\#_x \mid x \in \mathcal{M}\}, \mathcal{R}, E, [\ ]_0)$, where $\mathcal{M}$ is a nonempty set of molecules, $\mathcal{A}$ is a nonempty set of assemblies consisting of molecules in $\mathcal{M}$, $\#_x$ is a function from $\mathcal{A}$ to $\mathbf{Z}_+$ such that $\#_x(X)$ indicates the number of molecules x contained in an assembly X, $\mathcal{R}$ is a set of reaction rules satisfying (1), E is a free energy function from $\mathcal{A}$ to $\mathbf{R}$, and $[\ ]_0$ is an initial distribution of $\mathcal{M}$. In case that $\mathcal{M}$, $\mathcal{A}$, and $\mathcal{R}$ are finite, we say that P is a *finite* HRS.

The problem of interest is to find an equilibrium state of P, i.e., a distribution $[\ ]$ of $\mathcal{A}$ satisfying equilibrium equations (3) of all $r \in \mathcal{R}$ and conservation laws (2) of all molecules $x \in \mathcal{M}$. Such a distribution $[\ ]$ is called an *equilibrium state* of P.

For instance, let us define an HRS for the above example reaction $\alpha + \beta \rightleftharpoons \alpha\beta$. Consider an HRS $P = (\mathcal{M}, \mathcal{A}, \{\#_x \mid x \in \mathcal{M}\}, \mathcal{R}, E, [\ ]_0)$, where $\mathcal{M} = \{\alpha, \beta\}$, $\mathcal{A} = \{\alpha, \beta, \alpha\beta\}$, $\mathcal{R} = \{\alpha + \beta \rightleftharpoons \alpha\beta\}$, and a function $\#$ is defined by: $\#_\alpha(\alpha) = 1$, $\#_\alpha(\beta) = 0$, $\#_\alpha(\alpha\beta) = 1$, $\#_\beta(\alpha) = 0$, $\#_\beta(\beta) = 1$, $\#_\beta(\alpha\beta) = 1$.

Then the problem is to find a distribution $[\ ]$ of $\mathcal{A}$ satisfying:

$$e^{E(\alpha\beta)} \times [\alpha\beta] = e^{E(\alpha)+E(\beta)} \times [\alpha][\beta],$$
$$[\alpha] + [\alpha\beta] = [\alpha]_0, \qquad [\beta] + [\alpha\beta] = [\beta]_0.$$

An HRS is said to be *consistent* if there is a distribution of $\mathcal{A}$ satisfying (2) for all $x \in \mathcal{M}$. It is said to be *inconsistent* if it is not consistent. In actuality, we can construct an inconsistent HRS. However, we can ignore such possibility in most of real applications in the sense discussed below.

In most of real applications, assemblies of molecules are usually constructed by reactions using molecules in $\mathcal{M}$. Therefore, the elements of $\mathcal{M}$ take part in the reactions defined by the rule set $\mathcal{R}$. Therefore, in general, we can assume that $\mathcal{M} \subseteq \mathcal{A}$ holds. We say that an HRS is *normal* if $\mathcal{M} \subseteq \mathcal{A}$ holds. Then we have the following proposition.

Proposition 1 A normal finite HRS is consistent.

An initial distribution $[\ ]_0$ of $\mathcal{M}$ is said to be *normal* if $[x]_0 > 0$ holds for every $x \in \mathcal{M}$.

Proposition 2 A normal finite HRS with normal initial distribution can have a distribution $[\ ]$ of $\mathcal{A}$ satisfying the equations (2) for all $x \in \mathcal{M}$ and $[X] > 0$ for all $X \in \mathcal{A}$.

In the sequel of this paper, we will assume the following condition (A0):

(A0) Any HRS in this paper is a normal finite HRS with a normal initial distribution.

Let $P = (\mathcal{M}, \mathcal{A}, \{\#_x \mid x \in \mathcal{M}\}, \mathcal{R}, E, [\,]_0)$ be an HRS. The*free energy* $FE(P, [\,])$ of P under distribution $[\,]$ of $\mathcal{A}$ is defined by

$$FE\big(P, [\,]\big) = \sum_{X \in \mathcal{A}} E(X) \cdot [X] + \sum_{X \in \mathcal{A}} [X]\big(\log[X] - 1\big). \tag{4}$$

In this paper, we define $0 \log 0 = 0$. Note that for any $X \in \mathcal{A}$, $E(X)$ is a constant. Free energy $FE(P, [\,])$ can be regarded as a function with respect to the *variables* $[X]$'s $(X \in \mathcal{A})$. We often simply write $FE(P)$ instead of $FE(P, [\,])$ if the context allows.

Consider the following minimization problem:

Free Energy Minimization Problem (FEMP)
minimize: $FE(P)$
subject to:

$$\sum_{X \in \mathcal{A}} \#_x(X) \cdot [X] = [x]_0 \quad (\forall x \in \mathcal{M}),$$

$$[X] \geq 0 \quad (\forall X \in \mathcal{A}).$$

The following theorem is a well-known result (but we do not know who is the first to find it).

Theorem 1 *A distribution* $[\,]$ *of* $\mathcal{A}$ *is an equilibrium state of* P *if* $[\,]$ *is a minimizer of FEMP.*

Therefore, the problem of computing equilibrium state can be reduced to FEMP. In this paper, we are interested in the case that the cardinality of $\mathcal{A}$ is tremendously larger than that of $\mathcal{M}$. We will give an approach to overcome such a difficulty.

3 Symmetric Enumeration Method (SEM)

3.1 Hypergraphs

Basic notions and definitions related to directed hypergraphs will be introduced in this subsection mainly based on [6, 7], but some notions are slightly different from the originals.

A *directed hypergraph* G is a pair (V, Eg), where V is a finite set of *vertices*, and Eg is a finite set of *hyperarcs* associated with two functions $t : Eg \rightarrow V$ and $H : Eg \rightarrow 2^V$. A directed hypergraph is simply called a *hypergraph* in this paper.

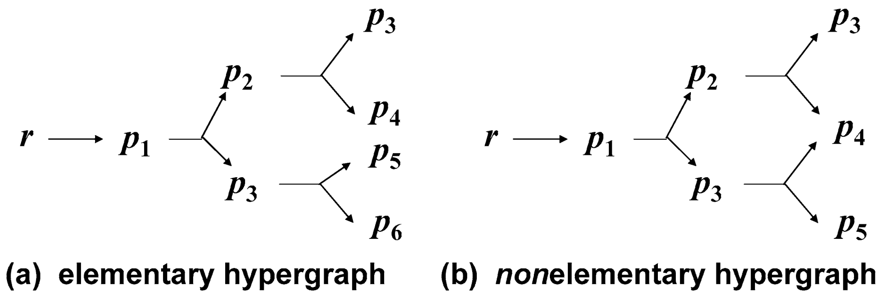


Fig. 1 Elementarity and nonelementary hypergraphs

A hyperarc e is interpreted as an arrow from a *tail* $t(e)$ to a set $H(e)$ of *heads*.[2] In this definition, we allow multihyperarcs, i.e., there can be more than one distinct hyperarcs with the same heads and the same tail. For a vertex v, a hyperarc e such that $v = t(e)$ ($v \in H(e)$) is called an *outgoing* (*entering*) hyperarc of v. For a vertex v, by v_{out} (v_{in}), we denote the set of outgoing (entering) hyperarcs of v. For a set W of vertices, we define $W_{\text{out}} = \bigcup_{v \in W} v_{\text{out}}$ and $W_{\text{in}} = \bigcup_{v \in W} v_{\text{in}}$. By V_0 and V_f, we denote the set of vertices $v \in V$ such that $v_{\text{in}} = \emptyset$ and $v_{\text{out}} = \emptyset$, respectively. Elements of V_0 and V_f are called *initial vertices* and *final vertices*, respectively.

A *path from s to u* in G is a sequence $s = v_1, e_1, v_2, e_2, \ldots, e_q, v_{q+1} = u$ of vertices v_i $(i = 1, \ldots, q+1)$ and hyperarcs e_i $(i = 1, \ldots, q)$ such that $v_i = t(e_i)$ and $v_{i+1} \in H(e_i)$ for $i = 1, \ldots, q$. If $s \in H(e_q)$ holds, the path is called a *cycle*. We say that G is *acyclic* if it contains no cycles.

A hypergraph $G' = (V', Eg')$ is called a *subhypergraph* of $G = (V, Eg)$ if $V' \subseteq V$ and $Eg' \subseteq Eg$ hold. Let x be an element of $V \cup Eg$. For a sub-hypergraph $G' = (V', Eg')$, we write $x \in G'$ if $x \in V' \cup Eg'$ holds. For a subset W of $V \cup Eg$, we write $W \subseteq G'$ if $x \in G'$ holds for every $x \in W$.

Let $r \in V$ and $S \subseteq V$. A *hyperpath of G from the root r to the sink set S* is a minimal acyclic subhypergraph γ of G such that r and S are contained in γ and every vertex of γ, except for those in S has exactly one outgoing hyperarc. A hyperpath is said to be *empty* if it contains only one vertex and no arcs (i.e., $S = \{r\}$). A hyperpath is said to be *elementary* if every vertex, except for r, has exactly one entering hyperarc. For a hypergraph $G = (V, Eg)$, by $PT(G)$, we denote the set of all hyperpaths from some root $r \in V_0$ to some sink set S with $S \subseteq V_f$. A hypergraph G is said to be *elementary* if every hyperpath in $PT(G)$ is elementary. We say that G is *reduced* if every hyperpath in $PT(G)$ is not an empty hyperpath. See Fig. 1 for examples of elementary and nonelementary acyclic hypergraphs.

[2] In the original definition, a hyperarc has a head and a set of tails.

3.2 Symmetric Enumeration of Assemblies by Hypergraphs

Let G be a reduced acyclic elementary hypergraph $G = (V, Eg)$. Let ϕ be an injective and surjective mapping from $V \cup Eg$ to $V \cup Eg$ such that $\phi(V) = V$ and $\phi(Eg) = Eg$ hold. For a subhypergraph $G' = (V', Eg')$ of G, by $\phi(G')$, we denote a graph $(\phi(V'), \phi(Eg'))$. If ϕ satisfies $\phi(PT(G)) = PT(G)$, ϕ is called a *path mapping*.

Let Φ be a set of path mappings of G. Φ is called a *path mapping group* if Φ constitutes a group under composition. For an element $s \in V \cup Eg$, we define a set $\Phi(s) = \{\phi(s) \mid \phi \in \Phi\}$. In case that Φ is a path mapping group, Φ naturally introduces an equivalence relation $\equiv_\Phi$ over Eg by $e_1 \equiv_\Phi e_2$ iff $e_1 = \phi(e_2)$ for some $\phi \in \Phi$. Then for $e \in Eg$, $\Phi(e)$ is regarded as an equivalence class with respect to $\equiv_\Phi$ containing e. For a set $S \subseteq Eg$, $\Phi(S)$ is defined as a multiset $\Phi(S) = \{\Phi(s) \mid s \in S\}$. Note that $\Phi(S)$ is a multiset, i.e., if s_1 and s_2 with $e_1 \equiv_\Phi e_2$ are contained in S, $\Phi(s_1)$ and $\Phi(s_2)$ represent a same set, but $\Phi(S)$ contains both of them redundantly. For $\gamma \in PT(G)$, $out(\gamma, \Phi)$ is defined as a multiset $out(\gamma, \Phi) = \{\Phi(v_{\text{out}}) \mid v \in V - V_0 - V_f$ such that $v \in \gamma\}$. Note that $out(\gamma, \Phi)$ is a multiset, i.e., even if there exist distinct $v_1, v_2 \in \gamma$ such that $\Phi((v_1)_{\text{out}}) = \Phi((v_2)_{\text{out}})$, those are counted redundantly in $out(\gamma, \Phi)$. Intuitively speaking, $out(\gamma, \Phi)$ represents a substructure of G along the hyperpath γ focusing on its outgoing hyperarcs in view of the equivalence relation $\equiv_\Phi$. For a hyperpath γ of G, we define a set $\Phi(\gamma) = \{\phi(\gamma) \mid \phi \in \Phi\}$.

A *weight function* w of G is a function from Eg to $\mathbf{R}$. A function f from $PT(G)$ to $\mathbf{R}$ is said to be *locally definable* if there exists a weight function $\overline{f}$ from Eg to $\mathbf{R}$ such that for every $\gamma \in PT(G)$,

$$f(\gamma) = \sum_{e \in Eg \text{ s.t. } e \in \gamma} \overline{f}(e)$$

holds. In this paper, we consider a reduced acyclic elementary hypergraph $G = (V, Eg)$ associated with a set of weight functions. By $\mathcal{W}(G)$, we denote the set of weight functions associated with G. A path mapping ϕ is said to be *weight preserving* if for any $w \in \mathcal{W}(G)$ and any $e \in Eg$, $w(e) = w(\phi(e))$ holds. A set Φ of path mappings is said to be *weight preserving* if every $\phi \in \Phi$ is weight preserving. Φ is said to be *structure preserving* if for every $\gamma \in PT(G)$ and $\phi \in \Phi$, $out(\gamma, \Phi) = out(\phi(\gamma), \Phi)$ holds.

Now, we will define the key concept of *locality* by using the above definitions and notations. Let $P = (\mathcal{M}, \mathcal{A}, \{\#_x \mid x \in \mathcal{M}\}, \mathcal{R}, E, [\ ]_0)$ be an HRS and consider a reduced acyclic elementary hypergraph $G = (V, Eg)$. Let ψ be a surjective function from $PT(G)$ to $\mathcal{A}$. For $X \in \mathcal{A}$ and $\gamma \in PT(G)$, we define $r_X = |\psi^{-1}(X)|$ and $r_\gamma = |\psi^{-1}(\psi(\gamma))|$, where r_X and r_γ are called a *rank* of X and a *rank* of γ, respectively. A *modified free energy function* E_r from $\mathcal{A}$ to $\mathbf{R}$ is defined as:

$$E_r(X) = E(X) + \log r_X.$$

This function E_r is a free energy function with compensation logarithmic factor related to the rank. This modified free energy function plays an important role for establishing the theory for equilibrium computation.

A triple $\mathcal{S} = (P, G, \psi)$ is called an *enumeration scheme* (ES) if $E_r \circ \psi$ and $\#_x \circ \psi$ are locally definable for every $x \in \mathcal{M}$. Let ϵ be a weight function to locally define $E_r \circ \psi$. For every $x \in \mathcal{M}$, let σ_x be a weight function to locally define $\#_x \circ \psi$. We define a set $\mathcal{W}(G)$ of weight functions associated with G as $\mathcal{W}(G) = \{\epsilon\} \cup \{\sigma_x \mid x \in \mathcal{M}\}$. An enumeration scheme $\mathcal{S} = (P, G, \psi)$ is said to be *symmetric* if there exists a path mapping group Φ such that:

(1) Φ is weight preserving,
(1) Φ is structure preserving, and
(3) $\Phi(\gamma) = \psi^{-1}(\psi(\gamma))$ holds for every $\gamma \in PT(G)$.

3.3 Reducing Number of Variables by Symmetric Enumeration Scheme

The difficulty for solving FEMP is that the cardinality of $\mathcal{A}$ is very large in real applications. In this subsection, we will explain a novel method (given in [10]) to reduce the number of variables of FEMP in case that the following assumptions hold:

(A1) An HRS P to be investigated has an enumeration scheme $\mathcal{S} = (P, G, \psi)$, and
(A2) the enumeration scheme in (A1) is symmetric.

Let $P = (\mathcal{M}, \mathcal{A}, \{\#_x \mid x \in \mathcal{M}\}, \mathcal{R}, E, []_0)$ be an HRS, $G = (V, Eg)$ be a reduced acyclic elementary hypergraph associated with a set $\mathcal{W}(G)$ of weight functions $\mathcal{W}(G) = \{\epsilon\} \cup \{\sigma_x \mid x \in \mathcal{M}\}$, where ϵ and σ_x are weight functions for locally defining $E_r \circ \psi$ and $\#_x \circ \psi$, respectively. For convenience, we often write X_γ instead of $\psi(\gamma)$. We define $PT(X) = \psi^{-1}(X)$, i.e., $PT(X)$ is the set of paths representing X.

Consider the following minimization problem:

Free Energy Minimization Problem (FEMP$_*$)
minimize:

$$FE_*\big(P, (w_e \mid e \in Eg)\big) \overset{\text{def}}{\equiv} \sum_{e \in Eg} \epsilon(e) \cdot w_e + \sum_{e \in Eg} w_e(\log w_e - 1)$$
$$- \sum_{v \in V - V_0 - V_f} w_v(\log w_v - 1)$$

subject to:

$$\sum_{e \in Eg} \sigma_x(e) \cdot w_e = [x]_0 \quad (\forall x \in \mathcal{M}),$$

$$\sum_{e \in v_{\text{in}}} w_e = \sum_{e \in v_{\text{out}}} w_e \quad (\forall v \in V - V_0 - V_f),$$

$$w_e = w_{\phi(e)} \quad (\forall e \in Eg,\ \forall \phi \in \Phi),$$

$$w_e \geq 0 \quad (\forall e \in Eg),$$

where w_v's are sums of variables w_e's, i.e., $w_v = \sum_{e \in v_{\text{out}}} w_e$. Note that the variables of FEMP$_*$ are w_e's ($e \in Eg$). Therefore, the number of variables are reduced from $|\mathcal{A}|$ in FEMP to $|Eg|$ in FEMP$_*$. We often omit the second argument of $FE_*(P, (w_e \mid e \in Eg))$, and simply write $FE_*(P)$ if the context allows.

As will be shown in Theorem 3, the objective function of FEMP$_*$ is convex. Therefore, FEMP$_*$ has an optimal solution.

Theorem 2 [10] *Assume* (A1) *and* (A2). *Let* $(\tilde{w}_e \mid e \in Eg)$ *be a minimizer of FEMP*$_*$. *Then the distribution* $[\ ]_*$ *defined by* (5) *and* (6) *based on* $\tilde{w}_e$*'s is a minimizer of FEMP.*

$$[\gamma]_+ = \frac{\prod_{e \in Eg \text{ s.t. } e \in \gamma} w_e}{\prod_{v \in V - V_0 - V_f \text{ s.t. } v \in \gamma} w_v}, \tag{5}$$

$$[X]_* = \sum_{\gamma \in PT(X)} [\gamma]_+. \tag{6}$$

Theorem 3 [10] *The objective function of FEMP*$_*$ *is convex over* $\mathbf{R}^m_{++}$, *where* $m = |Eg|$.

Therefore, we can solve FEMP by solving FEMP$_*$ with a convex programming method [12].

4 One-Dimensional Assembly of Rotatable Tiles

We consider a linear assembly system of tiles $a_1, \ldots, a_k$, where we allow three different 180-degree rotations $\theta_x, \theta_y, \theta_z$ of them. The rotation θ_x (θ_y, θ_z, respectively) is a 180-degree rotation on the x-axis (y-axis, z-axis, respectively) (see Fig. 2). So, for a tile p, we have 4 different orientations: p (at its original direction), $\theta_x(p)$, $\theta_y(p)$, and $\theta_z(p)$, each graphically corresponding to $\boxed{\text{p}}$, $\boxed{\text{d}}$, $\boxed{\text{b}}$, and $\boxed{\text{q}}$, respectively. We assume here that for each tile $p \in \Sigma = \{a_1, \ldots, a_k\}$, the above 4 orientations are distinguishable from each other, i.e., each tile p itself does not have

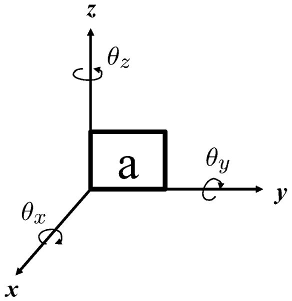


Fig. 2 Rotation of a tile

symmetric property. For convenience, we will prepare symbols, p_x, p_y, and p_z for representing $\theta_x(p)$, $\theta_y(p)$, and $\theta_z(p)$, respectively.

In this HRS, we assume that every vertical edge (i.e., an edge which is parallel to z-axis) can be bonded to each other by a chemical reaction rule. Note that rotations θ_x, θ_y, and θ_z keep those edges *vertical*. Then a linear assembly of rotations of tiles can be represented by a string over $\Gamma = \{p, p_x, p_y, p_z \mid p \in \Sigma\}$. We extend the mappings θ_x, θ_y, and θ_z over the domain Γ in a natural manner to those over the domain Γ^* so that θ_x and θ_z might be antimorphisms and θ_y be a morphism, i.e., the following conditions might hold: $\theta_x(uv) = \theta_x(v)\theta_x(u)$, $\theta_z(uv) = \theta_z(v)\theta_z(u)$ and $\theta_y(uv) = \theta_y(u)\theta_y(v)$ for $u, v \in \Gamma^*$. Then two strings w_1 and w_2 over Γ represent a same linear assembly if and only if $w_2 = \theta_q(w_1)$ holds for some $q \in \{x, y, z\}$. Thus, we introduce an equivalence relation $\equiv$ over Γ^* so that $w_1 \equiv w_2$ holds if and only if $w_2 = \theta_q(w_1)$ for some $q \in \{x, y, z\}$. Then a set $\mathcal{A}_n$ of linear assemblies of length at most n can be defined as $\Gamma^{\leq n}/\equiv$. The set $\mathcal{M} = \Gamma/\equiv$ can be regarded as a set of molecules. Free energy $E(w)$ of each assembly $w = a_1 \cdots a_k \in \Gamma^+$ is defined as $E(w) = sym(w) + \sum_{i=1}^{k-1} eg(a_i, a_{i+1})$ using free energy function eg for local substructures, where $sym(w)$ is an entropic term related to the symmetric property of the assembly w and $eg(x, y)$ is the free energy value corresponding to the boundary substructure between the tiles x and y. Note that $eg(p, q) = eg(\theta_x(q), \theta_x(p)) = eg(\theta_y(p), \theta_y(q)) = eg(\theta_z(q), \theta_z(p))$ holds for $p, q \in \Gamma$. Consider a corresponding finite HRS $P = (\mathcal{M}, \mathcal{A}_n, \{\#_x \mid x \in \mathcal{M}\}, \mathcal{R}, E, [\ ]_0)$ where $\mathcal{R}$ is a finite set of chemical reactions among all possible combinations of assemblies in $\mathcal{A}_n$, and $\#_x$ is a function from $\mathcal{A}_n$ to $\mathbf{Z}_+$ such that $\#_x(X)$ represents the number of molecules x contained in X.

We first give some basic symmetric properties of θ_x, θ_y, and θ_z.

Proposition 3 The following relations hold:

(1) $\theta_p \circ \theta_p$ is an identity function for any $p \in \{x, y, z\}$,
(2) $\theta_p \circ \theta_q = \theta_q \circ \theta_p$ for any $p, q \in \{x, y, z\}$,
(3) $\theta_x \circ \theta_y = \theta_z$, $\theta_y \circ \theta_z = \theta_x$, $\theta_z \circ \theta_x = \theta_y$,
(4) for any $w \in \Gamma^+$, $\theta_y(w) \neq w$,
(5) for any $w \in \Gamma^+$ of odd length, $w \neq \theta_p(w)$ holds for $p \in \{x, z\}$,
(6) for any $w \in \Gamma^+$ and distinct $p, q, r \in \{x, y, z\}$, $w = \theta_p(w)$ implies $\theta_q(w) = \theta_r(w)$.

We say that a tile represented by $w \in \Gamma^+$ is of symmetric degree d if

$$d = \left|\{q \in \{x, y, z\} \mid \theta_q(w) = w\}\right| + 1.$$

Then we have the following proposition.

Proposition 4 For any $X \in \Gamma^+/\equiv$, either $|X| = 2$ or $|X| = 4$ holds. Furthermore, the symmetric degree of X is given by $\frac{4}{|X|}$.

Proof We first show that $|X| = 2$ or $|X| = 4$ holds.

Let $X \in \Gamma^+/\equiv$ and $w \in X$. Consider any $w' \in X$ with $w' \neq w$. Then there exists $p \in \{x, y, z\}$ such that $\theta_p(w) = w'$. Therefore, there exist at most three such w's. Thus, we have $|X| \leq 4$ for any $X \in \Gamma^+/\equiv$.

Assume that $|X| = 1$ holds for some $X \in \Gamma^+/\equiv$ and let $w \in X$. Then we have $w = \theta_x(w) = \theta_y(w) = \theta_z(w)$. However, by Proposition 3 (4), $w \neq \theta_y(w)$ holds, a contradiction.

Assume that $|X| = 3$ holds for some $X \in \Gamma^+/\equiv$. Then we have $w = \theta_p(w)$ for some $w \in X$ and $p \in \{x, y, z\}$. By Proposition 3 (6), we have $\theta_q(w) = \theta_r(w)$ for distinct $q, r \in \{x, y, z\}$ with $p \neq q$ and $p \neq r$. Therefore, $|X| \leq 2$ holds, a contradiction.

Finally, we will show that the symmetric degree of an assembly X is given by $\frac{4}{|X|}$. By the discussion above, we need to consider the cases of $|X| = 2$ and $|X| = 4$.

Let $w \in \Gamma^+$ such that w represents an assembly X and recall that X is mathematically defined by $X = \{w, \theta_x(w), \theta_y(w), \theta_z(w)\}$.

Consider the case of $|X| = 4$. In this case, there exists no $p \in \{x, y, z\}$ such that $\theta_p(w) = w$ for some $w \in X$, since otherwise $|X| < 4$ holds. Therefore, by definition, the symmetric degree of X is 1.

Consider the case of $|X| = 2$. In this case, there exist distinct $p, q \in \{x, y, z\}$ such that $\theta_p(w) = \theta_q(w)$. Let $r \in \{x, y, z\} - \{p, q\}$. By $\theta_p(w) = \theta_q(w)$ and Proposition 3 (1)(3), $w = \theta_p \circ \theta_p(w) = \theta_p \circ \theta_q(w) = \theta_r(w)$ holds. By $|X| = 2$, we can conclude that $w = \theta_r(w) \neq \theta_p(w) = \theta_q(w)$ holds. Therefore, the symmetric degree of X is 2. □

Consider $w_1 = ab_yab_x$. Then, $\theta_x(w_1) = ba_xb_za_x$, $\theta_y(w_1) = a_yba_yb_z$, and $\theta_z(w_1) = b_ya_zb_xa_z$, where $|\{w_1, \theta_x(w_1), \theta_y(w_1), \theta_z(w_1)\}| = 4$ holds. The symmetric degree of this assembly is 1.

For $w_2 = ab_xba_x$, we have $\theta_x(w_2) = ab_xba_x$, $\theta_y(w_2) = a_yb_zb_ya_z$, and $\theta_z(w_2) = a_yb_zb_ya_z$, where $w_2 = \theta_x(w_2)$, $\theta_y(w_2) = \theta_z(w_2)$ and $|\{w_1, \theta_x(w_1), \theta_y(w_1), \theta_z(w_1)\}| = 2$ hold. The symmetric degree of this assembly is 2.

These propositions are very important for constructing a symmetric enumeration scheme of the HRS P.

5 Applying SEM to One-Dimensional Tile Assembly

We will apply Symmetric Enumeration Method (SEM) to the tile assembly system introduced in Sect. 4. In order to show the effectiveness of the SEM, we will give three examples of symmetric enumeration schemes for the HRS P in Sect. 4.

The enumeration schemes given in Sects. 5.1 and 5.2 are based on the following assumption (T) concluded by stochastic physics:

(T) the entropic term $sym(w)$ for an assembly X represented by $w \in \Gamma^+$ is given by $+\log(d)$ where d is the symmetric degree of X.

The enumeration scheme given in Sect. 5.3 does not depend on the assumption (T), but only assumes that:

(T′) for any $w_1, w_2 \in \Gamma^+$, if the symmetric degree of assemblies represented by w_1 and w_2 are equivalent to each other, then $sym(w_1) = sym(w_2)$ holds.

In the sequel, we define $k = |\Sigma|$, and by n we denote the maximum length of assemblies.

The enumeration scheme given in Sect. 5.1 has $O(k^2n^2)$ hyperarcs in its hypergraph definition. On the other hand, the number of hyperarcs of the hypergraphs in Sects. 5.2 and 5.3 is $O(k^3n)$.

5.1 Symmetric ES Under Assumption (T)

We will construct an enumeration scheme $\mathcal{S} = (P, G_1, \psi_1)$ using a graph $G_1 = (V_1, Eg_1)$ defined below:

$$V_1 = \{I, F\} \cup \{V(l, i, p) \mid 1 \le l \le n, 1 \le i \le l, p \in \Gamma\},$$

$$Eg_1 = \{e(l, i, p_1, p_2) \mid 1 \le l \le n, 1 \le i \le n-1, p_1, p_2 \in \Gamma\} \cup \{g(p) \mid p \in \Sigma\}.$$

The number of hyperarcs in G_1 is $O(k^2n^2)$. See Fig. 3 for the main structure of G_1.

First, we will explain the definition of G_1 in a intuitive manner. The hypergraph G_1 is decomposed into n disjoint graph components C_l $(l = 1, \dots, n)$ such that each C_l enumerates assemblies of length l. C_1 is the most simple component consisting of vertex set $\{I, F\}$ and hyperarc set $\{g(p) \mid p \in \Gamma\}$. Initial and final vertices of $g(p)$ $(p \in \Gamma)$ are I and F, respectively. We generate a tile p by a hyperarc of the form $g(p)$.

The component C_l $(l = 2, \dots, n)$ consists of the set of vertices of the form $V(l, i, p) \in V_1$ and the set of hyperarcs of the form $e(l, i, p_1, p_2) \in Eg_1$. Starting from an initial vertex of the form $V(l, 1, p_1)$, we will move to a sequence of vertices $V(l, 2, p_2), V(l, 3, p_3), \dots, V(l, l, p_{l-1})$ reaching to a final vertex $V(l, l, p_l)$:

$$V(l, 1, p_1) \overset{e(l,1,p_1,p_2)}{\to} V(l, 2, p_2) \overset{e(l,2,p_2,p_3)}{\to} \cdots \overset{e(l,l-1,p_{l-1},p_l)}{\to} V(l, l, p_l),$$

which generates an assembly $p_1 \cdots p_l$ from left to right. Note that the number of hyperarcs of the above hyperpath is $l - 1$, but the number of tiles to be generated is l. Thus, we will generate two tiles at hyperarcs located at the middle of the path as follows. Let $m = \lfloor \frac{l}{2} \rfloor$.

In case that l is even, the middle hyperarc $e(l, m, p_m, p_{m+1})$ generates two tiles p_m and p_{m+1}. The other hyperarcs of the form $e(l, i, p_1, p_2)$ generate a tile p_1 if $i < m$ and a tile p_2 if $i > m$.

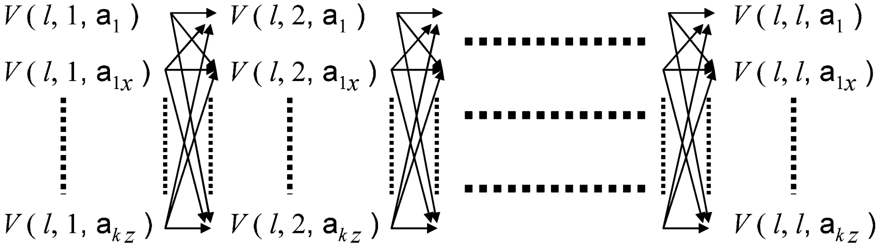


Fig. 3 Hypergraph component C_l of G_1 $(l \ge 2)$

In case that l is odd, there are two middle hyperarcs $e(l, m, p_m, p_{m+1})$ and $e(l, m+1, p_{m+1}, p_{m+2})$. By these hyperarcs together, we generate three tiles p_m, p_{m+1} and p_{m+2}. This can be done by assigning p_m and a left half of p_{m+1} to $e(l, m, p_m, p_{m+1})$ and by assigning p_{m+2} and a right half of p_{m+1} to $e(l, m+1, p_{m+1}, p_{m+2})$. The other hyperarcs of the form $e(l, i, p_1, p_2)$ generate a tile p_1 if $i < m$ and a tile p_2 if $i > m+1$.

The formal definition of the tails and heads of all hyperarcs are given below:

$$\begin{aligned}
t\bigl(e(l, i, p_1, p_2)\bigr) &= p_1 \quad (1 < l \le n,\ 1 \le i < l,\ p_1,\ p_2 \in \Gamma),\\
H\bigl(e(l, i, p_1, p_2)\bigr) &= \{p_2\} \quad (1 < l \le n,\ 1 \le i < l,\ p_1,\ p_2 \in \Gamma),\\
t\bigl(g(p)\bigr) &= I \quad (p \in \Sigma),\\
H\bigl(g(p)\bigr) &= F \quad (p \in \Sigma).
\end{aligned}$$

The definition of the weight functions $\sigma_{1,p}$ for locally defining $\#_p \circ \psi_1$ is given as follows:

$$\sigma_{1,p}\bigl(g(q)\bigr) = \begin{cases} 1 & \text{if } p = q, \\ 0 & \text{otherwise,} \end{cases} \quad (p \in \Sigma),$$

$$\sigma_{1,p}\bigl(e(l, i, p_1, p_2)\bigr) = \begin{cases} 2 & \text{if } l \text{ is even, } i = \lfloor \frac{l}{2} \rfloor \text{ and } p = p_1 = p_2, \\ 1 & \text{if } l \text{ is even, } i = \lfloor \frac{l}{2} \rfloor \text{ and } (p = p_1 \text{ xor } p = p_2), \\ 1 & \text{if } l \text{ is even, } i < \lfloor \frac{l}{2} \rfloor \text{ and } p = p_1, \\ 0 & \text{if } l \text{ is even, } i < \lfloor \frac{l}{2} \rfloor \text{ and } p \ne p_1, \\ 1 & \text{if } l \text{ is even, } i > \lfloor \frac{l}{2} \rfloor \text{ and } p = p_2, \\ 0 & \text{if } l \text{ is even, } i > \lfloor \frac{l}{2} \rfloor \text{ and } p \ne p_2, \end{cases}$$

$$\sigma_{1,p}\bigl(e(l, i, p_1, p_2)\bigr) = \begin{cases} 1.5 & \text{if } l \text{ is odd, } i = \lfloor \frac{l}{2} \rfloor \text{ and } p = p_1 = p_2, \\ 1 & \text{if } l \text{ is odd, } i = \lfloor \frac{l}{2} \rfloor,\ p = p_1 \text{ and } p \ne p_2, \\ 0.5 & \text{if } l \text{ is odd, } i = \lfloor \frac{l}{2} \rfloor,\ p \ne p_1 \text{ and } p = p_2, \\ 0 & \text{if } l \text{ is odd, } i = \lfloor \frac{l}{2} \rfloor,\ p \ne p_1 \text{ and } p \ne p_2, \\ 1.5 & \text{if } l \text{ is odd, } i = \lfloor \frac{l}{2} \rfloor + 1 \text{ and } p = p_1 = p_2, \\ 1 & \text{if } l \text{ is odd, } i = \lfloor \frac{l}{2} \rfloor + 1,\ p = p_2 \text{ and } p \ne p_1, \\ 0.5 & \text{if } l \text{ is odd, } i = \lfloor \frac{l}{2} \rfloor + 1,\ p \ne p_2 \text{ and } p = p_1, \\ 0 & \text{if } l \text{ is odd, } i = \lfloor \frac{l}{2} \rfloor + 1,\ p \ne p_2 \text{ and } p \ne p_1, \\ 1 & \text{if } l \text{ is odd, } i < \lfloor \frac{l}{2} \rfloor,\ p = p_1, \\ 0 & \text{if } l \text{ is odd, } i < \lfloor \frac{l}{2} \rfloor,\ p \ne p_1, \\ 1 & \text{if } l \text{ is odd, } i > \lfloor \frac{l}{2} \rfloor + 1,\ p = p_2, \\ 0 & \text{if } l \text{ is odd, } i > \lfloor \frac{l}{2} \rfloor + 1,\ p \ne p_2, \end{cases}$$

$$(1 \le i \le l,\ p_1, p_2 \in \Gamma).$$

In order to introduce a weight function ϵ_1 for locally defining the modified free energy E_r, we will first investigate rank terms $\log r_X$ for various assemblies X. Consider the following hyperpaths:

$$\begin{aligned}
\alpha_1 &= V(8,1,a_{1x}) \to V(8,2,a_2) \to V(8,3,a_{3z}) \to V(8,4,a_2) \to V(8,5,a_{2x}) \\
&\to V(8,6,a_{3y}) \to V(8,7,a_{2x}) \to V(8,8,a_1), \\
\alpha_2 &= V(8,1,a_{1z}) \to V(8,2,a_{2y}) \to V(8,3,a_{3x}) \to V(8,4,a_{2y}) \to V(8,5,a_{2z}) \\
&\to V(8,6,a_3) \to V(8,7,a_{2z}) \to V(8,8,a_{1y}), \\
\alpha_3 &= V(8,1,a_{1x}) \to V(8,2,a_3) \to V(8,3,a_{2y}) \to V(8,4,a_1) \to V(8,5,a_{1z}) \\
&\to V(8,6,a_{2x}) \to V(8,7,a_{3z}) \to V(8,8,a_{1y}), \\
\alpha_4 &= V(8,1,a_{1z}) \to V(8,2,a_{3y}) \to V(8,3,a_2) \to V(8,4,a_{1y}) \to V(8,5,a_{1x}) \\
&\to V(8,6,a_{2z}) \to V(8,7,a_{3x}) \to V(8,8,a_1), \\
\alpha_5 &= V(8,1,a_{2x}) \to V(8,2,a_{3x}) \to V(8,3,a_2) \to V(8,4,a_{1z}) \to V(8,5,a_2) \\
&\to V(8,6,a_{3x}) \to V(8,7,a_3) \to V(8,8,a_2), \\
\alpha_6 &= V(8,1,a_{2x}) \to V(8,2,a_{3x}) \to V(8,3,a_3) \to V(8,4,a_{2x}) \to V(8,5,a_{1y}) \\
&\to V(8,6,a_{2x}) \to V(8,7,a_3) \to V(8,8,a_2), \\
\alpha_7 &= V(8,1,a_{2z}) \to V(8,2,a_{3z}) \to V(8,3,a_{2y}) \to V(8,4,a_{1x}) \to V(8,5,a_{2y}) \\
&\to V(8,6,a_{3z}) \to V(8,7,a_{3y}) \to V(8,8,a_{2y}), \\
\alpha_8 &= V(8,1,a_{2z}) \to V(8,2,a_{3z}) \to V(8,3,a_{3y}) \to V(8,4,a_{2z}) \to V(8,5,a_1) \\
&\to V(8,6,a_{2z}) \to V(8,7,a_{3y}) \to V(8,8,a_{2y}).
\end{aligned}$$

The corresponding assemblies are given by

$$\begin{aligned}
\psi_1(\alpha_1) &\to u_1 = a_{1x}a_2a_{3z}a_2a_{2x}a_{3y}a_{2x}a_1, \\
\psi_1(\alpha_2) &\to u_2 = a_{1z}a_{2y}a_{3x}a_{2y}a_{2z}a_3a_{2z}a_{1y}, \\
\psi_1(\alpha_3) &\to u_3 = a_{1x}a_3a_{2y}a_1a_{1z}a_{2x}a_{3z}a_{1y}, \\
\psi_1(\alpha_4) &\to u_4 = a_{1z}a_{3y}a_2a_{1y}a_{1x}a_{2z}a_{3x}a_1, \\
\psi_1(\alpha_5) &\to u_5 = a_{2x}a_{3x}a_2a_{1z}a_2a_{3x}a_3a_2, \\
\psi_1(\alpha_6) &\to u_6 = a_{2x}a_{3x}a_3a_{2x}a_{1y}a_{2x}a_3a_2, \\
\psi_1(\alpha_7) &\to u_7 = a_{2z}a_{3z}a_{2y}a_{1x}a_{2y}a_{3z}a_{3y}a_{2y}, \\
\psi_1(\alpha_8) &\to u_8 = a_{2z}a_{3z}a_{3y}a_{2z}a_1a_{2z}a_{3y}a_{2y}.
\end{aligned}$$

Note that $\theta_y(u_1) = u_2$, $\theta_y(u_3) = u_4$, $\theta_x(u_5) = u_6$, $\theta_y(u_5) = u_7$, and $\theta_z(u_5) = u_8$ hold. Therefore, we have

$$r_{\alpha_1} = 2, \qquad r_{\alpha_2} = 2, \qquad r_{\alpha_3} = 2, \qquad r_{\alpha_4} = 2,$$

$$
\begin{array}{llll}
r_{\psi_1(\alpha_1)} = 2, & r_{\psi_1(\alpha_2)} = 2, & r_{\psi_1(\alpha_3)} = 2, & r_{\psi_1(\alpha_4)} = 2, \\
r_{\alpha_5} = 4, & r_{\alpha_6} = 4, & r_{\alpha_7} = 4, & r_{\alpha_8} = 4, \\
r_{\psi_1(\alpha_5)} = 4, & r_{\psi_1(\alpha_6)} = 4, & r_{\psi_1(\alpha_7)} = 4, & r_{\psi_1(\alpha_8)} = 4.
\end{array}
$$

Then, the modified free energy function E_r is given as follows for $\psi_1(\alpha_1)$, $\psi_1(\alpha_3)$, and $\psi_1(\alpha_5)$:

$$
\begin{aligned}
E_r\bigl(\psi_1(\alpha_1)\bigr) &= \log 2 + \log 2 + eg(a_{1x}, a_2) + eg(a_2, a_{3z}) + eg(a_{3z}, a_2) + eg(a_2, a_{2x}) \\
&\quad + eg(a_{2x}, a_{3y}) + eg(a_{3y}, a_{2x}) + eg(a_{2x}, a_1) \\
&= E_r\bigl(\psi_1(\alpha_2)\bigr), \\
E_r\bigl(\psi_1(\alpha_3)\bigr) &= \log 2 + \log 2 + eg(a_{1x}, a_3) + eg(a_3, a_{2y}) + eg(a_{2y}, a_1) + eg(a_1, a_{1z}) \\
&\quad + eg(a_{1z}, a_{2x}) + eg(a_{2x}, a_{3z}) + eg(a_{3x}, a_{1y}) \\
&= E_r\bigl(\psi_1(\alpha_4)\bigr), \\
E_r\bigl(\psi_1(\alpha_5)\bigr) &= \log 1 + \log 4 + eg(a_{2x}, a_{3x}) + eg(a_{3x}, a_2) + eg(a_2, a_{1z}) \\
&\quad + eg(a_{1z}, a_2) + eg(a_2, a_{3x}) + eg(a_{3x}, a_3) + eg(a_3, a_2) \\
&= E_r\bigl(\psi_1(\alpha_6)\bigr) = E_r\bigl(\psi_1(\alpha_7)\bigr) = E_r\bigl(\psi_1(\alpha_8)\bigr),
\end{aligned}
$$

where the first logarithmic term of each modified free energy originates from the entropic factor related to the rotational symmetry of the assembly, and the second logarithmic term is the logarithmic factor related to the rank of each assembly. In this way, the modified free energy $E_r(X)$ of a linear assembly X contains the constant logarithmic term $\log 4$ whether X is symmetric or not. One of the way to locally define the function E_r is to assign to each hyperarc h the free energy of local substructures generated by h, and furthermore to add the constant term $\frac{\log 4}{2}$ to each initial and final hyperarc. Formal definition of the weight function ϵ_1 for locally defining E_r is given by

$$
\begin{aligned}
\epsilon_1\bigl(e(l, 1, p_1, p_2)\bigr) &= \frac{\log 4}{2} + eg(p_1, p_2) \quad (1 \le l \le n,\ p_1, p_2 \in \Gamma), \\
\epsilon_1\bigl(e(l, i, p_1, p_2)\bigr) &= eg(p_1, p_2) \quad (1 \le l \le n,\ 1 < i < l-1,\ p_1, p_2 \in \Gamma), \\
\epsilon_1\bigl(e(l, l-1, p_1, p_2)\bigr) &= \frac{\log 4}{2} + eg(p_1, p_2) \quad (1 \le l \le n,\ p_1, p_2 \in \Gamma), \\
\epsilon_1\bigl(g(p)\bigr) &= 0 \quad (p \in \Sigma).
\end{aligned}
$$

Next, we discuss on the symmetric properties of G_1. Let us define the following path mapping ϕ_p for each $p \in \{x, y, z\}$:

$$
\begin{aligned}
&\phi_p(I) = F, \\
&\phi_p(F) = I,
\end{aligned}
$$

$$\phi_p\bigl(V(l,i,q)\bigr) = \begin{cases} V(l, l-i+1, \theta_p(q)), & \text{if } p \in \{x, z\}, \\ V(l, i, \theta_p(q)), & \text{otherwise}, \end{cases}$$

$$(1 \le l \le n,\ 1 \le i \le l,\ q \in \varGamma),$$

$$\phi_p\bigl(e(l,i,p_1,p_2)\bigr) = \begin{cases} e(l, l-i, \theta_p(p_2), \theta_p(p_1)), & \text{if } p \in \{x, z\}, \\ e(l, i, \theta_p(p_1), \theta_p(p_2)), & \text{otherwise}, \end{cases}$$

$$(1 \le l \le n,\ 1 \le i \le l,\ p_1, p_2 \in \varGamma),$$

$$\phi_p\bigl(g(p)\bigr) = g(p) \quad (p \in \varSigma).$$

For instance, we have $\phi_y(\alpha_1) = \alpha_2$, $\phi_y(\alpha_3) = \alpha_4$, $\phi_x(\alpha_5) = \alpha_6$, $\phi_y(\alpha_5) = \alpha_7$, and $\phi_z(\alpha_5) = \alpha_8$.

It is straightforward to see that ϕ_p is weight preserving since $eg(p,q) = eg(\theta_x(q), \theta_x(p)) = eg(\theta_y(p), \theta_y(q)) = eg(\theta_z(q), \theta_z(p))$ holds for $p, q \in \varGamma$, and since $\sigma_{1,p}$ and ϵ_1 are carefully defined so that the weight values of them are assigned symmetrically around the middle point of the graph component C_l.

Let ϕ_0 be the identity mapping from $V_1 \cup Eg_1$ to $V_1 \cup Eg_1$. Now, we should check that the set $\varPhi_1 = \{\phi_0, \phi_x, \phi_y, \phi_z\}$ is a path mapping group and is structure preserving. Since each ϕ_p corresponds to θ_p for $p \in \{x, y, z\}$, it is clear that $\varPhi_1$ is a path mapping group.

Consider an equivalence relation over Eg_1 introduced by $\varPhi_1$, and for each hyperarc $h \in Eg_1$, by $\langle h \rangle$ we denote its equivalence class containing h. For instance, we have

$$\begin{aligned} out(\varPhi_1, \alpha_1) = \bigl\{ &\bigl\{\bigl\langle e(8,2,a_2,p)\bigr\rangle \mid p \in \varGamma\bigr\}, \bigl\{\bigl\langle e(8,3,a_{3z},p)\bigr\rangle \mid p \in \varGamma\bigr\}, \\ &\bigl\{\bigl\langle e(8,4,a_2,p)\bigr\rangle \mid p \in \varGamma\bigr\}, \bigl\{\bigl\langle e(8,5,a_{2x},p)\bigr\rangle \mid p \in \varGamma\bigr\}, \\ &\bigl\{\bigl\langle e(8,6,a_{3y},p)\bigr\rangle \mid p \in \varGamma\bigr\}, \bigl\{\bigl\langle e(8,7,a_{2x},p)\bigr\rangle \mid p \in \varGamma\bigr\}\bigr\} \\ = out&(\varPhi_1, \alpha_2), \\ out(\varPhi_1, \alpha_3) = \bigl\{ &\bigl\{\bigl\langle e(8,2,a_3,p)\bigr\rangle \mid p \in \varGamma\bigr\}, \bigl\{\bigl\langle e(8,3,a_{2y},p)\bigr\rangle \mid p \in \varGamma\bigr\}, \\ &\bigl\{\bigl\langle e(8,4,a_1,p)\bigr\rangle \mid p \in \varGamma\bigr\}, \bigl\{\bigl\langle e(8,5,a_{1z},p)\bigr\rangle \mid p \in \varGamma\bigr\}, \\ &\bigl\{\bigl\langle e(8,6,a_{2x},p)\bigr\rangle \mid p \in \varGamma\bigr\}, \bigl\{\bigl\langle e(8,7,a_{3z},p)\bigr\rangle \mid p \in \varGamma\bigr\}\bigr\} \\ = out&(\varPhi_1, \alpha_4), \\ out(\varPhi_1, \alpha_5) = \bigl\{ &\bigl\{\bigl\langle e(8,2,a_{3x},p)\bigr\rangle \mid p \in \varGamma\bigr\}, \bigl\{\bigl\langle e(8,3,a_2,p)\bigr\rangle \mid p \in \varGamma\bigr\}, \\ &\bigl\{\bigl\langle e(8,4,a_{1z},p)\bigr\rangle \mid p \in \varGamma\bigr\}, \bigl\{\bigl\langle e(8,5,a_2,p)\bigr\rangle \mid p \in \varGamma\bigr\}, \\ &\bigl\{\bigl\langle e(8,6,a_{3x},p)\bigr\rangle \mid p \in \varGamma\bigr\}, \bigl\{\bigl\langle e(8,7,a_3,p)\bigr\rangle \mid p \in \varGamma\bigr\}\bigr\} \\ = out&(\varPhi_1, \alpha_6). \end{aligned}$$

It is straightforward to see that $out(\varPhi_1, \alpha_5) = out(\varPhi_1, \alpha_7) = out(\varPhi_1, \alpha_8)$ holds. In this way, it is easy to see that $\varPhi_1$ is structure preserving. Furthermore, it holds that $\psi_1(\phi_p(\alpha)) = \psi_1(\alpha)$ holds for every $\alpha \in PT(G_1)$ and $p \in \{x, y, z\}$, i.e., ϕ_p maps a given hyperpath α to its rotated counterpart representing a same linear assembly.

Thus, we have $\Phi_1(\alpha) = \psi_1^{-1}(\psi_1(\alpha))$ holds for every $\alpha \in PT(G_1)$. In summary, the enumeration scheme $\mathcal{S} = (P, G_1, \psi_1)$ is symmetric.

Theorem 4 *$\mathcal{S}_1 = (P, G_1, \psi_1)$ is a symmetric enumeration scheme.*

In order to compute the equilibrium of a linear assembly system P, we need to solve a convex programming problem with a set of variables of size $|Eg_1| = O(k^2n^2)$, where k is the number of tiles in Σ and n is the maximum length of assemblies.

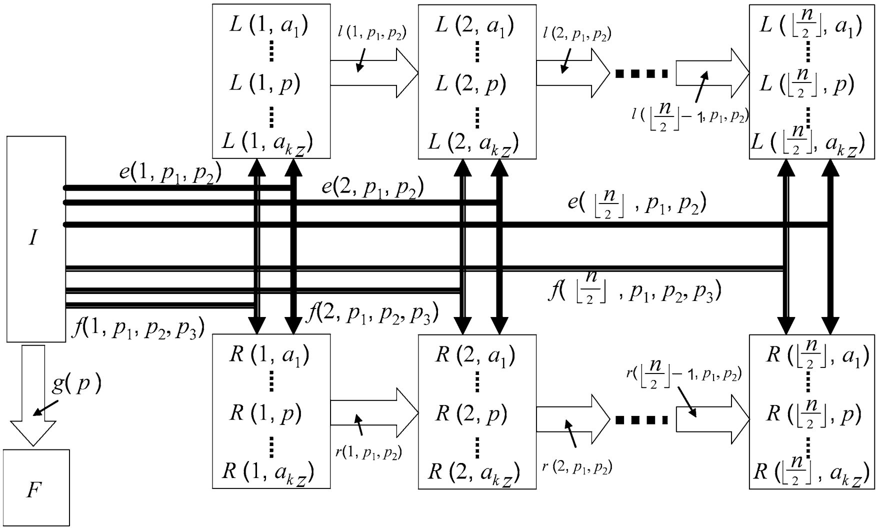
5.2 Another Symmetric ES Under Assumption (T)

We will construct an enumeration scheme $\mathcal{S} = (P, G_2, \psi_2)$ using a graph $G_2 = (V_2, Eg_2)$ defined below:

$$\begin{aligned} V_2 &= \{I, F\} \cup \left\{L(i, p), R(i, p) \mid 1 \le i \le \left\lfloor \tfrac{n}{2} \right\rfloor, p \in \Gamma \right\}, \\ Eg_2 &= \left\{e(i, p_1, p_2), f(i, p_1, p_2, p_3) \mid 1 \le i \le \left\lfloor \tfrac{n}{2} \right\rfloor, p_1, p_2, p_3 \in \Gamma \right\} \\ &\quad \cup \left\{l(i, p_1, p_2), r(i, p_1, p_2) \mid 1 \le i \le \left\lfloor \tfrac{n}{2} \right\rfloor - 1,\ p_1, p_2 \in \Gamma \right\} \\ &\quad \cup \left\{g(p) \mid p \in \Sigma \right\}. \end{aligned}$$

The number of hyperarcs in G_2 is $O(k^3n)$.

Figure 4 shows a schematic view of the graph G_2. Starting from the initial vertex I, we can apply a hyperarc either of the form $e(i, p_1, p_2)$, $f(i, p_1, p_2, p_3)$, or



$L(1, a_1)$
$L(1, p)$
$L(1, a_{k_Z})$
$l(1, p_1, p_2)$
$L(2, a_1)$
$L(2, p)$
$L(2, a_{k_Z})$
$l(2, p_1, p_2)$
$l(\lfloor \frac{n}{2} \rfloor - 1, p_1, p_2)$
$L(\lfloor \frac{n}{2} \rfloor, a_1)$
$L(\lfloor \frac{n}{2} \rfloor, p)$
$L(\lfloor \frac{n}{2} \rfloor, a_{k_Z})$
I
$e(1, p_1, p_2)$
$e(2, p_1, p_2)$
$e(\lfloor \frac{n}{2} \rfloor, p_1, p_2)$
$f(\lfloor \frac{n}{2} \rfloor, p_1, p_2, p_3)$
$f(1, p_1, p_2, p_3)$
$f(2, p_1, p_2, p_3)$
$g(p)$
F
$R(1, a_1)$
$R(1, p)$
$R(1, a_{k_Z})$
$r(1, p_1, p_2)$
$R(2, a_1)$
$R(2, p)$
$R(2, a_{k_Z})$
$r(2, p_1, p_2)$
$r(\lfloor \frac{n}{2} \rfloor - 1, p_1, p_2)$
$R(\lfloor \frac{n}{2} \rfloor, a_1)$
$R(\lfloor \frac{n}{2} \rfloor, p)$
$R(\lfloor \frac{n}{2} \rfloor, a_{k_Z})$

Fig. 4 Schematic view of enumeration graph for linear assemblies

$g(p)$ for some $p_1, p_2, p_3 \in \Gamma$, $p \in \Sigma$ and $1 \le i \le \lfloor \frac{n}{2} \rfloor$. Applying $g(p)$ finishes the hyperpath immediately at the final vertex F, by which we mean the generation of a single tile p. Application of $e(i, p_1, p_2)$ means the generation of two tiles $p_1 p_2$ at the middle of a linear assembly of even length to be generated. Application of $f(i, p_1, p_2, p_3)$ means the generation of three tiles $p_1 p_3 p_2$ at the middle of a linear assembly of odd length to be generated. After applying $e(i, p_1, p_2)$ or $f(i, p_1, p_2, p_3)$, the hyperpath is split to $L(i, p_1)$ and $R(i, p_2)$. A hyperarc of the form $l(i, p_1, p_2)$ ($r(i, p_1, p_2)$) outgoing from $L(i, p_1)$ ($R(i, p_1)$) generates a new tile p_2 to the left (right) of the tile p_1. In this way, split paths generate tiles of the same length from the middle to the both ends, thus a hyperpath starting from the hyperarc of the form $e(i, p_1, p_2)$ ($f(i, p_1, p_2, p_3)$) generates a linear assembly of even length (odd length). The formal definition of the tails and heads of all hyperarcs are given below:

$$t\big(e(i, p_1, p_2)\big) = I \quad \big(1 \le i \le \lfloor \tfrac{n}{2} \rfloor, p_1, p_2 \in \Gamma\big),$$
$$H\big(e(i, p_1, p_2)\big) = \big\{L(i, p_1), R(i, p_2)\big\} \quad \big(1 \le i \le \lfloor \tfrac{n}{2} \rfloor,\ p_1, p_2 \in \Gamma\big),$$
$$t\big(f(i, p_1, p_2, p_3)\big) = I \quad \big(1 \le i \le \lfloor \tfrac{n}{2} \rfloor,\ p_1, p_2, p_3 \in \Gamma\big),$$
$$H\big(f(i, p_1, p_2, p_3)\big) = \big\{L(i, p_1), R(i, p_2)\big\} \quad \big(1 \le i \le \lfloor \tfrac{n}{2} \rfloor,\ p_1, p_2, p_3 \in \Gamma\big),$$
$$t\big(l(i, p_1, p_2)\big) = L(i, p_1) \quad \big(1 \le i \le \lfloor \tfrac{n}{2} \rfloor - 1,\ p_1, p_2 \in \Gamma\big),$$
$$H\big(l(i, p_1, p_2)\big) = \big\{L(i + 1, p_2)\big\} \quad \big(1 \le i \le \lfloor \tfrac{n}{2} \rfloor - 1,\ p_1, p_2 \in \Gamma\big),$$
$$t\big(r(i, p_1, p_2)\big) = R(i, p_1) \quad \big(1 \le i \le \lfloor \tfrac{n}{2} \rfloor - 1, p_1, p_2 \in \Gamma\big),$$
$$H\big(r(i, p_1, p_2)\big) = \big\{R(i + 1, p_2)\big\} \quad \big(1 \le i \le \lfloor \tfrac{n}{2} \rfloor - 1,\ p_1, p_2 \in \Gamma\big),$$
$$t\big(g(p)\big) = I \quad (p \in \Sigma),$$
$$H\big(g(p)\big) = F \quad (p \in \Sigma).$$

Some examples of hyperpaths of G_2 in case of $n = 6$ (n: maximum length of assemblies) are shown in Figs. 5 and 6. The hyperpaths γ_1, γ_2, γ_3, and γ_4 generate linear assemblies $w_1 = a_{3z} a_{2x} a_1 a_{1x} a_2 a_{3y}$, $w_2 = a_{3x} a_{2z} a_{1y} a_{1z} a_{2y} a_3$, $w_3 =$

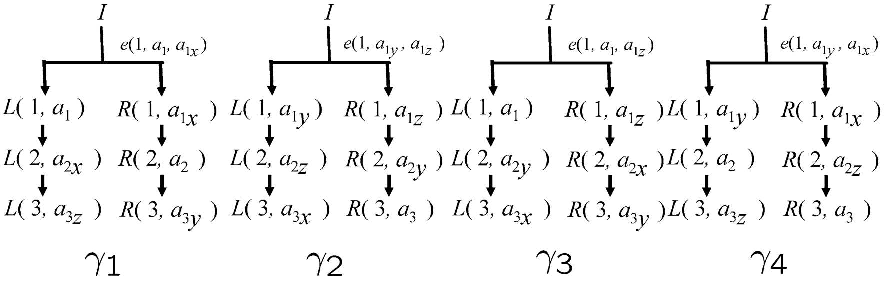


Fig. 5 Hyperpaths of G_2

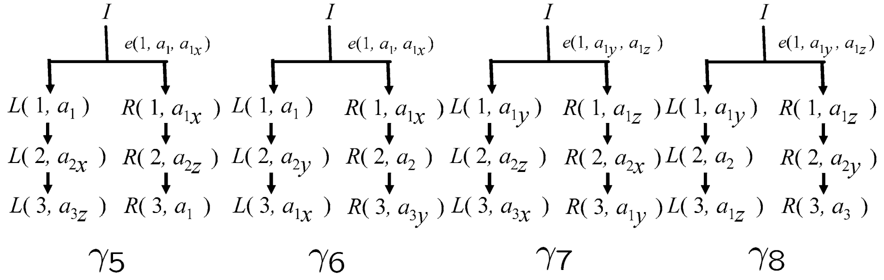


Fig. 6 Hyperpaths of G_2

$a_{3x}a_{2y}a_1a_{1z}a_{2x}a_{3y}$, and $w_4 = a_{3z}a_2a_{1y}a_{1x}a_{2z}a_3$, respectively. Note that $\theta_x(w_1) = w_1$, $\theta_x(w_2) = w_2$ and $\theta_y(w_1) = w_2$ hold, i.e., w_1 and w_2 represent a same linear assembly and are rotationally symmetric around x-axis, and that $\theta_z(w_3) = w_3$, $\theta_z(w_4) = w_4$, and $\theta_y(w_3) = w_4$ hold, i.e., w_3 and w_4 represent a same linear assembly and are rotationally symmetric around z-axis. The hyperpaths γ_5, γ_6, γ_7, and γ_8 generate linear assemblies $w_5 = a_{3z}a_{2x}a_1a_{1x}a_{2z}a_1$, $w_6 = a_{1x}a_{2y}a_1a_{1x}a_2a_{3y}$, $w_7 = a_{3x}a_{2z}a_{1y}a_{1z}a_{2x}a_{1y}$, and $w_8 = a_{1z}a_2a_{1y}a_{1z}a_{2y}a_3$, respectively. In this case, we have $w_6 = \theta_x(w_5)$, $w_7 = \theta_y(w_5)$, and $w_8 = \theta_z(w_5)$. Therefore, w_5, w_6, w_7, and w_8 represent a same tile assembly and its symmetric degree is 1. Therefore, we have

$$r_{\gamma_1} = 2, \quad r_{\gamma_2} = 2, \quad r_{\gamma_3} = 2, \quad r_{\gamma_4} = 2,$$
$$r_{\psi_2(\gamma_1)} = 2, \quad r_{\psi_2(\gamma_2)} = 2, \quad r_{\psi_2(\gamma_3)} = 2, \quad r_{\psi_2(\gamma_4)} = 2,$$
$$r_{\gamma_5} = 4, \quad r_{\gamma_6} = 4, \quad r_{\gamma_7} = 4, \quad r_{\gamma_8} = 4,$$
$$r_{\psi_2(\gamma_5)} = 4, \quad r_{\psi_2(\gamma_6)} = 4, \quad r_{\psi_2(\gamma_7)} = 4, \quad r_{\psi_2(\gamma_8)} = 4.$$

Then the modified free energy function E_r gives the following values for $\psi_2(\gamma_1)$, $\psi_2(\gamma_3)$, and $\psi_2(\gamma_5)$:

$$\begin{aligned}
E_r\big(\psi_2(\gamma_1)\big) &= \log 2 + \log 2 + eg(a_{3z}, a_{2x}) + eg(a_{2x}, a_1) \\
&\quad + eg(a_1, a_{1x}) + eg(a_{1x}, a_2) + eg(a_2, a_{3y}) \\
&= E_r\big(\psi_2(\gamma_2)\big), \\
E_r\big(\psi_2(\gamma_3)\big) &= \log 2 + \log 2 + eg(a_{3x}, a_{2y}) + eg(a_{2y}, a_1) \\
&\quad + eg(a_1, a_{1z}) + eg(a_{1z}, a_{2x}) + eg(a_{2x}, a_{3y}) \\
&= E_r\big(\psi_2(\gamma_4)\big), \\
E_r\big(\psi_2(\gamma_5)\big) &= \log 1 + \log 4 + eg(a_{3z}, a_{2x}) + eg(a_{2x}, a_1) \\
&\quad + eg(a_1, a_{1x}) + eg(a_{1x}, a_{2z}) + eg(a_{2z}, a_1) \\
&= E_r\big(\psi_2(\gamma_6)\big) = E_r\big(\psi_2(\gamma_7)\big) = E_r\big(\psi_2(\gamma_8)\big),
\end{aligned}$$

where the first logarithmic term of each modified free energy originates from the entropic term related to the rotational symmetry of the assembly, and the second logarithmic term is the logarithmic factor related to the rank of each assembly. In this way, the modified free energy $E_r(X)$ of a linear assembly X contains the constant logarithmic term $\log 4$ whether X is symmetric or not. One of the way to locally define the function E_r is to assign to each hyperarc h the free energy of local substructures generated by h, and furthermore to add the constant term $\log 4$ to each hyperarc outgoing from I. Formal definition of the weight function ϵ_2 for locally defining E_r is given by

$$\begin{aligned}
&\epsilon_2\bigl(e(i,p_1,p_2)\bigr) = \log 4 + eg(p_1,p_2) \quad \bigl(1 \le i \le \lfloor \tfrac{n}{2} \rfloor,\ p_1,p_2 \in \varGamma\bigr),\\
&\epsilon_2\bigl(f(i,p_1,p_2,p_3)\bigr) = \log 4 + eg(p_1,p_3) + eg(p_3,p_2)\\
&\quad \bigl(1 \le i \le \lfloor \tfrac{n}{2} \rfloor,\ p_1,p_2,p_3 \in \varGamma\bigr),\\
&\epsilon_2\bigl(l(i,p_1,p_2)\bigr) = eg(p_2,p_1) \quad \bigl(1 \le i \le \lfloor \tfrac{n}{2} \rfloor - 1,\ p_1,p_2 \in \varGamma\bigr),\\
&\epsilon_2\bigl(r(i,p_1,p_2)\bigr) = eg(p_1,p_2) \quad \bigl(1 \le i \le \lfloor \tfrac{n}{2} \rfloor - 1,\ p_1,p_2 \in \varGamma\bigr),\\
&\epsilon_2\bigl(g(p)\bigr) = 0 \quad (p \in \varSigma).
\end{aligned}$$

The definition of the weight functions $\sigma_{2,x}$ for locally defining $\#_x \circ \psi_2$ are omitted, because they are clear from the definition of G_2 and the explanation of the enumerating process by G_2. (For each $x \in \mathcal{M}$ and hyperarc h, we just need to count the number of tiles x appearing in the local substructures generated by h.)

Finally, we will discuss the symmetric property of the enumeration scheme $\mathcal{S}_2 = (P, G_2, \psi_2)$. For each $p \in \{x,y,z\}$, consider the following injective and surjective mapping ϕ_p from $V_2 \cup Eg_2$ to $V_2 \cup Eg_2$:

$$\begin{aligned}
&\phi_p(I) = I,\\
&\phi_p(F) = F,\\
&\phi_p\bigl(L(i,q)\bigr) = \begin{cases} R(i,\theta_p(q)), & \text{if } p \in \{x,z\},\\ L(i,\theta_p(q)), & \text{otherwise} \end{cases} \quad \bigl(1 \le i \le \lfloor \tfrac{n}{2} \rfloor,\ q \in \varGamma\bigr),\\
&\phi_p\bigl(R(i,q)\bigr) = \begin{cases} L(i,\theta_p(q)), & \text{if } p \in \{x,z\},\\ R(i,\theta_p(q)), & \text{otherwise} \end{cases} \quad \bigl(1 \le i \le \lfloor \tfrac{n}{2} \rfloor,\ q \in \varGamma\bigr),\\
&\phi_p\bigl(e(i,p_1,p_2)\bigr) = \begin{cases} e(i,\theta_p(p_2),\theta_p(p_1)), & \text{if } p \in \{x,z\},\\ e(i,\theta_p(p_1),\theta_p(p_2)), & \text{otherwise} \end{cases}\\
&\quad \bigl(1 \le i \le \lfloor \tfrac{n}{2} \rfloor,\ p_1,p_2 \in \varGamma\bigr),\\
&\phi_p\bigl(f(i,p_1,p_2,p_3)\bigr) = \begin{cases} f(i,\theta_p(p_2),\theta_p(p_1),\theta_p(p_3)), & \text{if } p \in \{x,z\},\\ f(i,\theta_p(p_1),\theta_p(p_2),\theta_p(p_3)), & \text{otherwise} \end{cases}\\
&\quad \bigl(1 \le i \le \lfloor \tfrac{n}{2} \rfloor,\ p_1,p_2,p_3 \in \varGamma\bigr),\\
&\phi_p\bigl(l(i,p_1,p_2)\bigr) = \begin{cases} r(i,\theta_p(p_1),\theta_p(p_2)), & \text{if } p \in \{x,z\},\\ l(i,\theta_p(p_1),\theta_p(p_2)), & \text{otherwise} \end{cases}\\
&\quad \bigl(1 \le i \le \lfloor \tfrac{n}{2} \rfloor - 1,\ p_1,p_2 \in \varGamma\bigr),
\end{aligned}$$

$$\begin{aligned}
&\phi_p\bigl(r(i,p_1,p_2)\bigr)=\begin{cases} l(i,\theta_p(p_1),\theta_p(p_2)), & \text{if } p\in\{x,z\},\\ r(i,\theta_p(p_1),\theta_p(p_2)), & \text{otherwise}\end{cases}\\
&\quad\bigl(1\le i\le \lfloor \tfrac{n}{2}\rfloor-1,\ p_1,p_2\in\Gamma\bigr),\\
&\phi_p\bigl(g(p)\bigr)=g(p)\quad (p\in\Sigma).
\end{aligned}$$

It is straightforward to see that for each $p\in\{x,y,z\}$, ϕ_p is a path mapping. Let ϕ_0 be an identity mapping from $V_2\cup Eg_2$ to $V_2\cup Eg_2$. Then the set $\Phi_2=\{\phi_0,\phi_x,\phi_y,\phi_z\}$ is a path mapping group. Since for each $p\in\{x,y,z\}$, ϕ_p is weight preserving, Φ_2 is weight preserving. It is clear that $\psi_2(\phi_p(\gamma))=\psi_2(\gamma)$ holds for every $\gamma\in PT(G_2)$ and $p\in\{x,y,z\}$, i.e., ϕ_p maps a given hyperpath γ to its rotated counterpart representing a same linear assembly. Thus, we have $\Phi_2(\gamma)=\psi_2^{-1}(\psi_2(\gamma))$ holds for every $\gamma\in PT(G_2)$.

Consider an equivalence relation over Eg_2 introduced by Φ_2, and for each hyperarc $h\in Eg_2$, by $\langle h\rangle$, we denote its equivalence class containing h. For instance, we have

$$\begin{aligned}
out(\Phi_2,\gamma_1)&=\bigl\{\bigl\{\bigl\langle l(1,a_1,p)\bigr\rangle\mid p\in\Gamma\bigr\},\bigl\{\bigl\langle l(2,a_{2x},p)\bigr\rangle\mid p\in\Gamma\bigr\},\\
&\qquad\bigl\{\bigl\langle r(1,a_{1x},p)\bigr\rangle\mid p\in\Gamma\bigr\},\bigl\{\bigl\langle r(2,a_2,p)\bigr\rangle\mid p\in\Gamma\bigr\}\bigr\}\\
&=\bigl\{\bigl\{\bigl\langle l(1,a_{1y},p_y)\bigr\rangle\mid p\in\Gamma\bigr\},\bigl\{\bigl\langle l(2,a_{2z},p_y)\bigr\rangle\mid p\in\Gamma\bigr\},\\
&\qquad\bigl\{\bigl\langle r(1,a_{1z},p_y)\bigr\rangle\mid p\in\Gamma\bigr\},\bigl\{\bigl\langle r(2,a_{2y},p_y)\bigr\rangle\mid p\in\Gamma\bigr\}\bigr\}\\
&=out(\Phi_2,\gamma_2),\\
out(\Phi_2,\gamma_3)&=\bigl\{\bigl\{\bigl\langle l(1,a_1,p)\bigr\rangle\mid p\in\Gamma\bigr\},\bigl\{\bigl\langle l(2,a_{2y},p)\bigr\rangle\mid p\in\Gamma\bigr\},\\
&\qquad\bigl\{\bigl\langle r(1,a_{1z},p)\bigr\rangle\mid p\in\Gamma\bigr\},\bigl\{\bigl\langle r(2,a_{2x},p)\bigr\rangle\mid p\in\Gamma\bigr\}\bigr\}\\
&=\bigl\{\bigl\{\bigl\langle l(1,a_{1y},p_y)\bigr\rangle\mid p\in\Gamma\bigr\},\bigl\{\bigl\langle l(2,a_2,p_y)\bigr\rangle\mid p\in\Gamma\bigr\},\\
&\qquad\bigl\{\bigl\langle r(1,a_{1x},p_y)\bigr\rangle\mid p\in\Gamma\bigr\},\bigl\{\bigl\langle r(2,a_{2z},p_y)\bigr\rangle\mid p\in\Gamma\bigr\}\bigr\}\\
&=out(\Phi_2,\gamma_4),\\
out(\Phi_2,\gamma_5)&=\bigl\{\bigl\{\bigl\langle l(1,a_1,p)\bigr\rangle\mid p\in\Gamma\bigr\},\bigl\{\bigl\langle l(2,a_{2x},p)\bigr\rangle\mid p\in\Gamma\bigr\},\\
&\qquad\bigl\{\bigl\langle r(1,a_{1x},p)\bigr\rangle\mid p\in\Gamma\bigr\},\bigl\{\bigl\langle r(2,a_{2z},p)\bigr\rangle\mid p\in\Gamma\bigr\}\bigr\}\\
&=\bigl\{\bigl\{\bigl\langle r(1,a_{1x},p_x)\bigr\rangle\mid p\in\Gamma\bigr\},\bigl\{\bigl\langle r(2,a_2,p_x)\bigr\rangle\mid p\in\Gamma\bigr\},\\
&\qquad\bigl\{\bigl\langle l(1,a_1,p_x)\bigr\rangle\mid p\in\Gamma\bigr\},\bigl\{\bigl\langle l(2,a_{2y},p_x)\bigr\rangle\mid p\in\Gamma\bigr\}\bigr\}\\
&=out(\Phi_2,\gamma_6).
\end{aligned}$$

Furthermore, it is straightforward to see that $out(\Phi_2,\gamma_5)=out(\Phi_2,\gamma_7)=out(\Phi_2,\gamma_8)$ holds. In this way, it is easy to see that Φ_2 is structure preserving, since ϕ_0 and ϕ_p for each $p\in\{x,y,z\}$ are graph isomorphisms of G_2. In summary, the enumeration scheme $\mathcal{S}=(P,G_2,\psi_2)$ is symmetric.

Theorem 5 *$\mathcal{S}_2 = (P, G_2, \psi_2)$ is a symmetric enumeration scheme.*

In order to compute the equilibrium of a linear assembly system P, we need to solve a convex programming problem with a set of variables of size $|Eg_2| = O(k^3 n)$, where k is the number of tiles in Σ and n is the maximum length of assemblies.

5.3 Symmetric ES Under Assumption (T′)

In the previous subsections, we give symmetric enumeration schemes under the assumption (T). In this subsection, we will give a symmetric enumeration scheme without the help of the assumption (T). Because of space constraint, we will only give a rough sketch of how to construct a symmetric enumeration scheme $\mathcal{S}_3 = (P, G_3, \psi_3)$ under the assumption (T′).

The key idea to implement such an enumeration scheme is described below. For $i = 1, 2$, let $\mathcal{A}_i$ be the set of assemblies of symmetric degree i. Then by Proposition 4, for each $i = 1, 2$, the assemblies in $\mathcal{A}_i$ should have the same rank. Therefore, for each $i = 1, 2$, the modified free energy $E_r(X)$ contains the same values for all $X \in \mathcal{A}_i$ with respect to the entropic term related to symmetric degree and the logarithmic term related to rank. Thus, it might be effective to enumerate assemblies of different symmetric degrees separately by different graph components M_1 and M_2 which are disjoint from each other.

Formally, $\mathcal{A}_1$ and $\mathcal{A}_2$ are given by $\mathcal{A}_1 = \{X \in \Gamma^+/\equiv \mid |X| = 4\}$ and $\mathcal{A}_2 = \{X \in \Gamma^+/\equiv \mid |X| = 2\}$. We will enumerate $\mathcal{A}_1$ and $\mathcal{A}_2$ separately by disjoint graph components.

Since the construction of enumeration graph for $\mathcal{A}_1$ is more complicated than that for $\mathcal{A}_2$, we will first describe how to generate representations of assemblies in $\mathcal{A}_2$. A graph component M_2 for enumerating $\mathcal{A}_2$ is schematically illustrated in Fig. 7. The enumeration process by M_2 could be *intuitively* seen as a derivation process of generating strings in $\bigcup_{X \in \mathcal{A}_2} X$ using a context free grammar. For instance, consider the following hyperpath:

$$I \to S_x(1, p_1) \to S_x(2, p_2) \to \cdots \to S_x(m, p_m),$$

where $m = \lfloor \frac{n}{2} \rfloor$. This hyperpath generates a tile assembly X by producing a symmetric pair of tiles $(p_i, \theta_x(p_i))$ step by step from the middle to the both ends of X. More precisely, when moving from I to $S_x(1, p_1)$, we generate two tiles $p_1\theta_x(p_1)$ at the middle. We next move from $S_x(1, p_1)$ to $S_x(2, p_2)$ to produce p_2 and $\theta_x(p_2)$ on both sides resulting in a partial assembly $p_2 p_1 \theta_x(p_1)\theta_x(p_2)$. Continuing this process, we will generate a sequence of tiles $p_m \cdots p_1 \theta_x(p_1) \cdots \theta_x(p_m)$ whose symmetric degree is 2. Since we can move from the initial vertex I to any vertex of the form $S_x(i, p)$ $(i = 1, \ldots, m,\ p \in \Gamma)$, we can generate any assemblies of length at most n and symmetric degree 2.

The part of M_2 consisting of the vertices $S_z(i, p)$ $(i = 1, \ldots, m, p \in \Gamma)$ generates a tile assembly symmetric around z-axis in a similar manner.

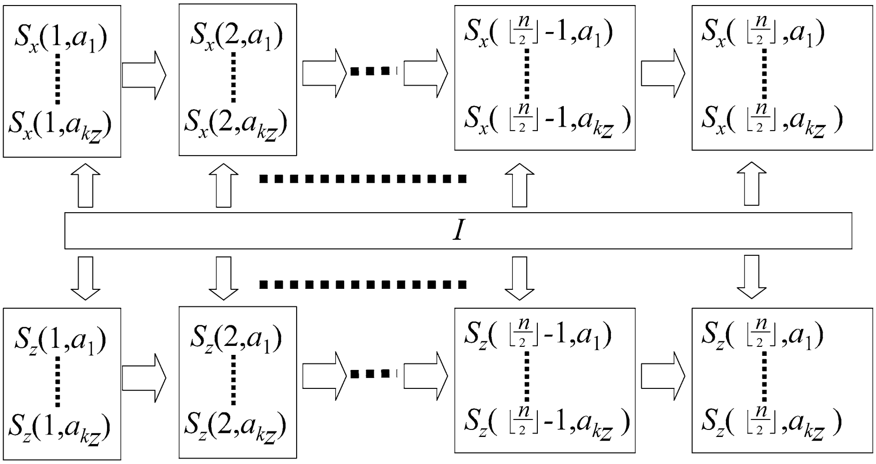

$S_x(1,a_1)$
$S_x(1,a_{kZ})$
$S_x(2,a_1)$
$S_x(2,a_{kZ})$
$S_x(\lfloor\frac{n}{2}\rfloor\text{-}1,a_1)$
$S_x(\lfloor\frac{n}{2}\rfloor\text{-}1,a_{kZ})$
$S_x(\lfloor\frac{n}{2}\rfloor,a_1)$
$S_x(\lfloor\frac{n}{2}\rfloor,a_{kZ})$
I
$S_z(1,a_1)$
$S_z(1,a_{kZ})$
$S_z(2,a_1)$
$S_z(2,a_{kZ})$
$S_z(\lfloor\frac{n}{2}\rfloor\text{-}1,a_1)$
$S_z(\lfloor\frac{n}{2}\rfloor\text{-}1,a_{kZ})$
$S_z(\lfloor\frac{n}{2}\rfloor,a_1)$
$S_z(\lfloor\frac{n}{2}\rfloor,a_{kZ})$

Fig. 7 Schematic view of hypergraph component M_2

The construction of graph component M_1 for generating $\mathcal{A}_1$ consists of two subcomponents M_1' and M_1''. The component M_1' generates assemblies in $\mathcal{A}_1$ of length even, and M_1'' generates those in $\mathcal{A}_1$ of length odd. Note that any odd length assembly is in $\mathcal{A}_1$, and thus the construction of M_1'' is almost similar to the part of G_2 starting from the initial hyperarcs of the form $f(i, p_1, p_2, p_3)$. So, we omit the definition of M_1'' and describe that of M_1' briefly. A schematic view of M_1' is illustrated in Fig. 8. We will now describe a generation process of an assembly X of symmetric degree 1. The process starts generating tiles from the middle to the both ends of X. The subblock of M_1' consisting of vertices of the form $A_r(i, q)$ ($r \in \{x, z\}$, $i = 1, \ldots, m$, $q \in \Gamma$) generates the r-axis symmetric part located at the middle of X. A hyperarc (*indicated by dashed arrow*) from a tail $A_r(i, p_1)$ to heads $B(i + 1, p_2)$ and $C(i + 1, p_3)$ ($r \in \{x, z\}, i = 1, \ldots, m$, $p_1, p_2, p_3 \in \Gamma$) generates a pair of tiles which is asymmetric around r-axis. Finally, from the vertices $B(i + 1, p_2)$ and $C(i + 1, p_3)$, we will generate the rest of left and right segments located at the both ends of X. For instance, consider a hyperpath in Fig. 9. It generates an assembly $a_1a_{2x}a_{3y}a_1a_{1z}a_{3x}a_2a_{1z}$. The weight definitions of the hypergraph G_3 is omitted since it might be straightforward to define them from the explanation above. As is explained already, in each graph component of M_1 and M_2, the entropic term related to symmetric degree and the logarithmic term related to the rank are constant values. Thus, such an constant value can be assigned to each initial hyperarc. In this way, we can construct an enumeration scheme satisfying the symmetric properties.

Theorem 6 $\mathcal{S}_3 = (P, G_3, \psi_3)$ *is a symmetric enumeration scheme.*

If we apply the scheme $\mathcal{S}_3$ for computing equilibrium of P, we need to solve a convex programming problem with a set of variables of size $|Eg_l| = O(k^3n)$, where k is the number of tiles in Σ and n is the maximum length of assemblies.

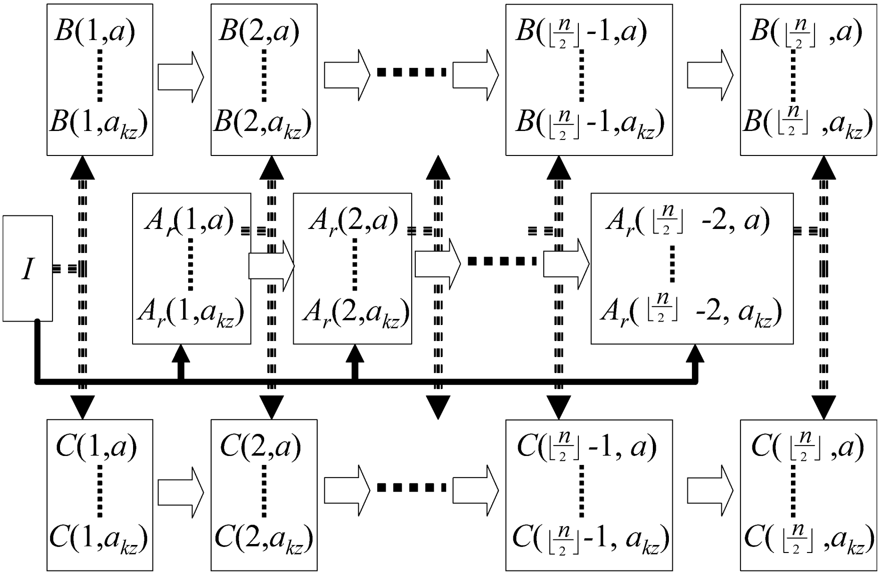
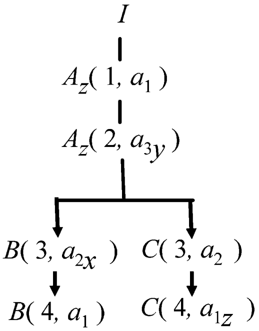

$B(1,a)$
$B(1,a_{kz})$
$B(2,a)$
$B(2,a_{kz})$
$B(\lfloor\frac{n}{2}\rfloor-1,a)$
$B(\lfloor\frac{n}{2}\rfloor-1,a_{kz})$
$B(\lfloor\frac{n}{2}\rfloor,a)$
$B(\lfloor\frac{n}{2}\rfloor,a_{kz})$
I
$A_r(1,a)$
$A_r(1,a_{kz})$
$A_r(2,a)$
$A_r(2,a_{kz})$
$A_r(\lfloor\frac{n}{2}\rfloor-2,a)$
$A_r(\lfloor\frac{n}{2}\rfloor-2,a_{kz})$
$C(1,a)$
$C(1,a_{kz})$
$C(2,a)$
$C(2,a_{kz})$
$C(\lfloor\frac{n}{2}\rfloor-1,a)$
$C(\lfloor\frac{n}{2}\rfloor-1,a_{kz})$
$C(\lfloor\frac{n}{2}\rfloor,a)$
$C(\lfloor\frac{n}{2}\rfloor,a_{kz})$

Fig. 8 Schematic view of hypergraph component M_1

Fig. 9 A hyperpath in M_1



6 Concluding Remarks

We reviewed the result of [8–10] and applied the symmetric enumeration method to an assembly system of tiles which are rotatable around x, y, z-axes. We gave three examples of enumeration schemes for the assembly system. Two of them assumed that

(T) the entropic term $sym(w)$ for an assembly X represented by $w \in \Gamma^+$ is given by $+\log(d)$ where d is the symmetric degree of X.

This is a natural request from the theory of stochastic physics. The other one, $\mathcal{S}_3$, assumed that

(T′) for any $w_1, w_2 \in \Gamma^+$, if the symmetric degree of assemblies represented by w_1 and w_2 are equivalent to each other, then $sym(w_1) = sym(w_2)$ holds.

Although this assumption is too general for real applications to chemical reaction systems, the fact that we can establish a symmetric enumeration scheme under (T′) tells us the effectiveness and generality of the proposed theory of the symmetric enumeration method.

Acknowledgements This work was supported in part by Grant-in-Aid for Scientific Research on Priority Area No.14085205, Ministry of Education, Culture, Sports, Science, and Technology, Japan, and is supported by Grant-in-Aid for Scientific Research (C) No. 17500192, Japan Society for the Promotion of Science.

References

1. Adleman L (1994) Molecular computation of solutions to combinatorial problems. Science 266:1021–1024
2. Adleman L, Cheng Q, Goel A, Huang M, Wasserman H (2000) Linear self-assemblies: equilibria, entropy, and convergence rates. Unpublished manuscript
3. Benneson A, Gil B, Ben-Dor U, Adar R, Shapiro E (2004) An autonomous molecular computer for logical control of gene expression. Nature 429:423–429
4. Condon AE (2003) Problems on rna secondary structure prediction and design. In: Proceedings of ICALP'2003. Lecture notes in computer science, vol 2719. Springer, Berlin, pp 22–32
5. Dirks R, Bois J, Schaeffer J, Winfree E, Pierce N (2007) Thermodynamic analysis of interacting nucleic acid strands. SIAM Rev 49:65–88
6. Gallo G, Longo G, Nguyen S, Pallottino S (1993) Directed hypergraphs and applications. Discrete Appl Math 40:177–201
7. Gallo G, Scutella MG (1999) Directed hypergraphs as a modelling paradigm. Technical report TR-99-02, Dipartmento di Informatica, Universita di Pisa
8. Kobayashi S (2006) A new approach to computing equilibrium state of combinatorial chemical reaction systems. Technical report CS 06-01, Department of Computer Science, University of Electro-Communications
9. Kobayashi S (2007) A new approach to computing equilibrium state of combinatorial hybridization reaction systems. In: Proceedings of workshop on computing and communications from biological systems: theory and applications. CD-ROM, paper 2376
10. Kobayashi S (2008) Symmetric enumeration method: a new approach to computing equilibria. Technical report CS 08-01, Department of Computer Science, University of Electro-Communications
11. Lipton RJ (1995) DNA solution of hard computational problems. Science 268:542–545
12. Nesterov Y, Nemirovskii A (1993) Interior-point polynomial algorithms in convex programming. SIAM, Philadelphia
13. Păun G, Rozenberg G, Salomaa A (1998) DNA computing—new computing paradigms. Texts in theoretical computer science—an EATCS series. Springer, Berlin
14. Rothemund P, Papadakis N, Winfree E (2004) Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biol 2:e424
15. Winfree E, Liu F, Wenzler L, Seeman NC (1998) Design and self-assembly of two-dimensional DNA crystals. Nature 394:539–544

A Self-assembly Model of Time-Dependent Glue Strength

Sudheer Sahu, Peng Yin, and John H. Reif

Abstract Self-assembly is a ubiquitous process in which small objects self-organize into larger and complex structures. In 2000, Rothemund and Winfree proposed a Tile Assembly Model as a mathematical model for theoretical studies of self-assembly. We propose a refined self-assembly model in which the glue strength between two juxtaposed tiles is a function of the time they have been in neighboring positions. We then present an implementation of our model using strand displacement reactions on DNA tiles. Under our model, we can demonstrate and study catalysis and self-replication in the tile assembly. We then study the tile complexity for assembling shapes in our model and show that a thin rectangle of size $k \times N$ can be assembled using $O((\log(N))/\log\log(N))$ types of tiles, demonstrating the glue model has additional capabilities over the prior tiling assembly model. We also describe a method to implement with DNA tiles our model of time-dependant glue strength.

1 Introduction

1.1 Motivation

Self-assembly is a ubiquitous process in which small objects self-organize into larger and complex structures. Examples in nature are numerous: atoms self-assemble into molecules, molecules into cells, cells into tissues, and so on. Recently, self-assembly has also been demonstrated as a powerful technique for constructing nanoscale objects. For example, a wide variety of DNA lattices made from self-assembled branched DNA molecules (DNA tiles) [10, 21, 23–25, 43, 45, 46] have been successfully constructed. Peptide self-assembly provides another nanoscale example [9]. Self-assembly is also used for mesoscale constructions using capillary forces [8, 30] or magnetic forces [1].

Sudheer Sahu was a PhD student in Duke University when this research was conducted.

S. Sahu (✉)
Department of Computer Science, Duke University, Box 90129, Durham, NC 27708, USA
e-mail: sudheer@cs.duke.edu

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_11, © Springer-Verlag Berlin Heidelberg 2009

1.2 Prior Models for Tile Assembly

Mathematical studies of tiling date back to the 1960s when Wang introduced his tiling model [39]. The initial focus of research in this area was toward the decidability/undecidability of the tiling problem [29]. A revival in the study of tiling was instigated in 1996 when Winfree proposed the simulation of computation [44] using self-assembly of DNA tiles.

In 2000, Rothemund and Winfree [32] proposed an *Abstract Tile Assembly* (*ATA*) *Model*, which is a mathematical model for theoretical studies of self-assembly. This model was later extended by Adleman et al. to include the time complexity of generating specified assemblies [3]. Later work includes combinatorial optimization, complexity problems, fault tolerance, and topology changes, in the abstract tile assembly model as well as in some of its variants [4–7, 11–14, 16, 19, 20, 22, 26, 27, 31, 33–35, 37, 38, 41, 42].

Adleman introduced a reversible model [2], and studied the kinetics of the reversible linear self-assemblies of tiles. Winfree also proposed a kinetic assembly model to study the kinetics of the self-assembly [40]. Apart from these basic models, various generalized models of self-assembly are also studied [6, 18]: namely, multiple temperature model, flexible glue model, and q-tile model.

1.3 Needs for New Models for Tile Assembly

Though all these models contribute greatly toward a good understanding of the process of self-assembly, there are still a few things that could not be easily explained or modeled (for example, the process of catalysis and self-replication in tile assembly). Recall that catalysis is the phenomenon in which an external substance facilitates the reaction of other substances, without itself being used up in the process. A catalyst provides an alternative route of reaction where the activation energy is lower than the original chemical reaction and increase the reaction rate. Adleman [2] has posed an open question if we could model the process of catalysis in the self-assembly of tiles. Self-replication process is one of the fundamental process of nature, in which a system creates copies of itself. For example, DNA is self-replicated during cell division and is transmitted to offspring during reproduction. A material device that can self-replicate is ambition of many engineering disciplines. The biggest incentive is to achieve a low manufacturing cost because self-replication avoids the costs of labor, capital, and distribution in conventional manufactured goods. In an evolving field like nanotechnology, manufacturing costs of molecular machines can become extremely large in the absence of self-replication. Recently, Schulman and Winfree show self-replication using the growth of DNA crystals [36], but their system requires shear forces to separate the replicated units. In this paper, we propose a new model, in which catalysis and self-replication is possible without external intervention. In our new model, which is built on the basic framework of the ATA model, the glue strength between different glues is dependent on the time for which they have remained together.

The rest of the paper is organized as follows. First, we define the prior ATA model as well as our new model formally in Sect. 2.2. We then put forth a method to physically implement such a system in Sect. 3. Then we present the processes of catalysis and self-replication in tile assembly in our model in Sects. 4 and 5, respectively. In Sect. 6, we discuss the tile complexity of assembly of various shapes in our model, beginning with the assembly of thin rectangles in Sect. 6.1 and the extension to other shapes in Sect. 6.2. We conclude with the discussion of our results and future research directions in Sect. 7.

2 Tiling Assembly Models

2.1 The Abstract Tiling Assembly (ATA) Model

The *Abstract Tile Assembly* (*ATA*) *model* was proposed by Rothemund and Winfree [32] in 2000. Intuitively speaking, a *tile* in the ATA model is a unit square where each side of the square has a *glue* from a set Σ associated with it. In this paper, we use the terms *pad* and side of the tile interchangeably. Formally, a tile is an ordered quadruple $(\sigma_n, \sigma_e, \sigma_s, \sigma_w) \in \Sigma^4$, where σ_n, σ_e, σ_s, and σ_w represent the *northern*, *eastern*, *southern*, and *western* side glues of the tile, respectively. Σ also contains a special symbol *null*, which is a zero-strength glue. T denotes the set of all tiles in the system. A tile cannot be rotated. So, $(\sigma_1, \sigma_2, \sigma_3, \sigma_4) \neq (\sigma_2, \sigma_3, \sigma_4, \sigma_1)$. Also defined are various projection functions $n : T \to \Sigma$, $e : T \to \Sigma$, $s : T \to \Sigma$, and $w : T \to \Sigma$, where $n(\sigma_1, \sigma_2, \sigma_3, \sigma_4) = \sigma_1$, $e(\sigma_1, \sigma_2, \sigma_3, \sigma_4) = \sigma_2$, $s(\sigma_1, \sigma_2, \sigma_3, \sigma_4) = \sigma_3$, and $w(\sigma_1, \sigma_2, \sigma_3, \sigma_4) = \sigma_4$.

A glue-strength function $g : \Sigma \times \Sigma \to \mathbb{R}$ determines the glue strength between two abutting tiles. $g(\sigma, \sigma') = g(\sigma', \sigma)$ is the strength between two tiles that abut on sides with glues σ and σ'. If $\sigma \neq \sigma'$, $g(\sigma, \sigma') = 0$; otherwise, it is a positive value. It is also assumed that $g(\sigma, \mathit{null}) = 0$, $\forall \sigma \in \Sigma$. In the tile set T, there is a specified unique *seed* tile s. There is a system parameter to control the assembly known as *temperature* and denoted as τ. All the ingredients described above constitute a *tile system*, a quadruple $\langle T, s, g, \tau \rangle$. A *configuration* is a snapshot of the assembly. More formally, it is the mapping from $\mathbb{Z}^2$ to $T \cup \{\mathit{EMPTY}\}$ where *EMPTY* is a special tile $(\mathit{null}, \mathit{null}, \mathit{null}, \mathit{null})$, indicating a tile is not present. For a configuration C, a tile $A = (\sigma_n, \sigma_e, \sigma_s, \sigma_w)$ is attachable at position (i, j) iff $C(i, j) = \mathit{EMPTY}$ and $g(\sigma_e, w(C(i, j+1))) + g(\sigma_n, s(C(i+1, j))) + g(\sigma_w, e(C(i, j-1))) + g(\sigma_s, n(C(i-1, j))) \geq \tau$, where indices i and j increase toward north and east directions, respectively.

Assembly takes place sequentially starting from a seed tile s at a known position. One key aspect of this ATA model is that the glues are constant over time. For a given tile system, any assembly that can be obtained by starting from the *seed* and adding tiles one by one, is said to be *produced*. An assembly is called to be *terminally* *produced* if no further tiles can be added to it. The *tile complexity* of a shape S is the size of the smallest tile set required to uniquely and terminally assemble S under a given assembly model. One of the well-known results is that the tile complexity of self-assembly of a square of size $N \times N$ in ATA model is $\Theta(\frac{\log N}{\log \log N})$ [3, 32].

2.2 Our Time-Dependent Glue (TDG) Model

We propose a time-dependent glue model, which is built on the framework described above. In this model, the glue-strength between two tiles is dependent upon the time for which the two tiles have remained together.

Let τ be the temperature of the system. Tiles are defined as in the ATA model. However, in our model, glue strength function, g, is extended to contain a third argument that specifies the time for which the two sides of tiles are in contact. Formally speaking, g is defined as $g : \Sigma \times \Sigma \times \mathbb{R} \rightarrow \mathbb{R}$.

In $g(\sigma, \sigma', t)$, the argument t is the time for which two sides of the tiles with glue-labels σ and σ' have been juxtaposed. For every pair (σ, σ'), the value $g(\sigma, \sigma', t)$ increases with t up to a maximum limit and then takes a constant value determined by σ and σ'. We define the time when g reaches this maximum as *time for maximum strength* and denote it as $\gamma : \Sigma \times \Sigma \rightarrow \mathbb{R}$. Note $g(\sigma, \sigma', t) = g(\sigma, \sigma', \gamma(\sigma, \sigma'))$ for $t \geq \gamma(\sigma, \sigma')$.

The *minimum interaction time* is a function $\mu : \Sigma \times \Sigma \rightarrow \mathbb{R}$. For every pair (σ, σ'), a function $\mu(\sigma, \sigma')$ is defined as the minimum time for which the two tiles with abutting glue symbols σ and σ' stay together. If $g(\sigma, \sigma', \mu(\sigma, \sigma')) \geq \tau$, the two tiles will stay together; otherwise, they will separate if there is no other force holding them in their abutting positions. An example of glue-strength function is shown in Fig. 1. Intuitively speaking, μ serves as the minimum time required by the pads to decide whether they want to separate or remain joined. We further define $\mu(\sigma, \mathit{null}) = 0$, $\gamma(\sigma, \mathit{null}) = 0$, and $g(\sigma, \mathit{null}, t) = 0$.

Next, we give the justification and estimation of μ for a pair (σ, σ') of glues. Let $g(\sigma, \sigma', t)$ be the glue-strength function. For more realistic estimation of μ, consider

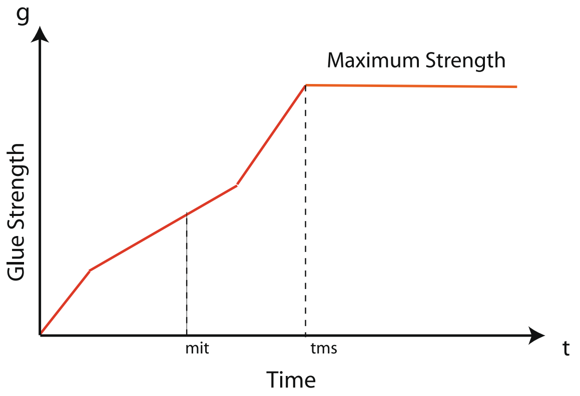


Fig. 1 Figure illustrates the concept of time-dependent glue strength, minimum interaction time, and time for maximum strength

a physical system in which, in addition to association, dissociation reactions also occur. Let $p(b)$ be the probability of dissociation when the bond strength is b, where $p(b)$ can be determined using Winfree's kinetic model [40]. Assume that $f(t)$ be the probability that no dissociation takes place in the time interval $[0, t]$, and assume the time-interval δt is so small that bond strength $g(\sigma, \sigma', t)$ does not change in the time-interval t and $t + \delta t$. Then

$$\begin{aligned} f(t+\delta t) &= f(t) \cdot \bigl(1 - p\bigl(g(\sigma, \sigma', t)\bigr)\bigr)^{\delta t}, \\ \frac{f(t+\delta t)}{f(t)} &= \bigl(1 - p\bigl(g(\sigma, \sigma', t)\bigr)\bigr)^{\delta t}, \\ \frac{f(t+\delta t)}{f(t)} &= \exp\bigl(-\delta t \cdot p\bigl(g(\sigma, \sigma', t)\bigr)\bigr). \end{aligned}$$

The probability that the dissociation takes place between time t and $t + \delta t$ is given by $f(t) \cdot (1 - \exp(-\delta t \cdot p(g(\sigma, \sigma', t))))$. Since μ is defined as the time for which two glues are expected to remain together once they come in contact, its expected value is

$$E[\mu] = \lim_{\delta t \to 0} \sum_{t=0}^{\infty} t \cdot f(t) \cdot \bigl(1 - \exp\bigl(-\delta t \cdot p\bigl(g(\sigma, \sigma', t)\bigr)\bigr)\bigr).$$

Hence, based on the knowledge of glue-strength function, it is possible to determine the expected minimum interaction time for a pair (σ, σ'). For simplicity, we will use the expected value of μ as the actual value of μ for a pair of glues (σ, σ').

Next, we illustrate the time-dependent model with an example of the addition of a single tile to an aggregate. In a configuration C, when a position (i, j) becomes available for the addition of a tile A, it will stay at (i, j) for a time interval t_0, where $t_0 = \max\{\mu(e(A), w(C(i, j+1))), \mu(n(A), s(C(i+1, j))), \mu(w(A), e(C(i, j-1))), \mu(s(A), n(C(i-1, j)))\}$. Recall that our model requires that if two tiles ever come in contact, they will stay together till the minimum interaction time of the corresponding glues.

After this time interval t_0, if $g(e(A), w(C(i, j+1)), t_0) + g(n(A), s(C(i+1, j)), t_0) + g(w(A), e(C(i, j-1)), t_0) + g(s(A), n(C(i-1, j)), t_0) < \tau$, tile A will detach; otherwise, A will continue to stay at position (i, j).

We describe in the next section a method to implement our model of time-dependent glue strength with DNA tiles.

3 Implementation of Time-Dependent Glue Model

In this section, we propose an implementation of time-dependent glue model using DNA. Structurally, DNA is a long polymer of simple units called nucleotides, which are held together by a backbone made of sugars and phosphate groups. This backbone carries four types of bases (A, C, T, and G). These bases form complementary pairs (A is complementary to T and C is complementary to G) in a sense

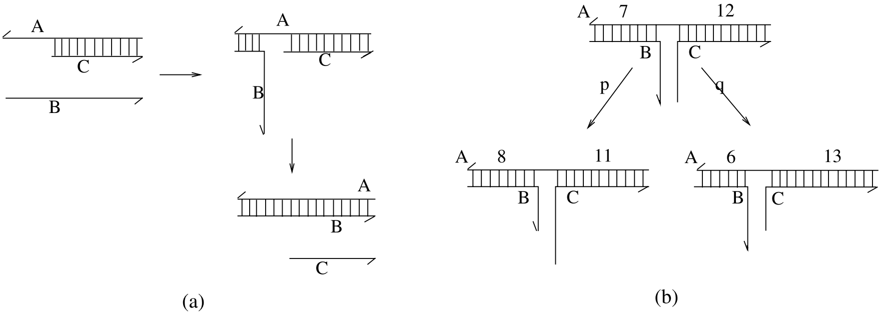
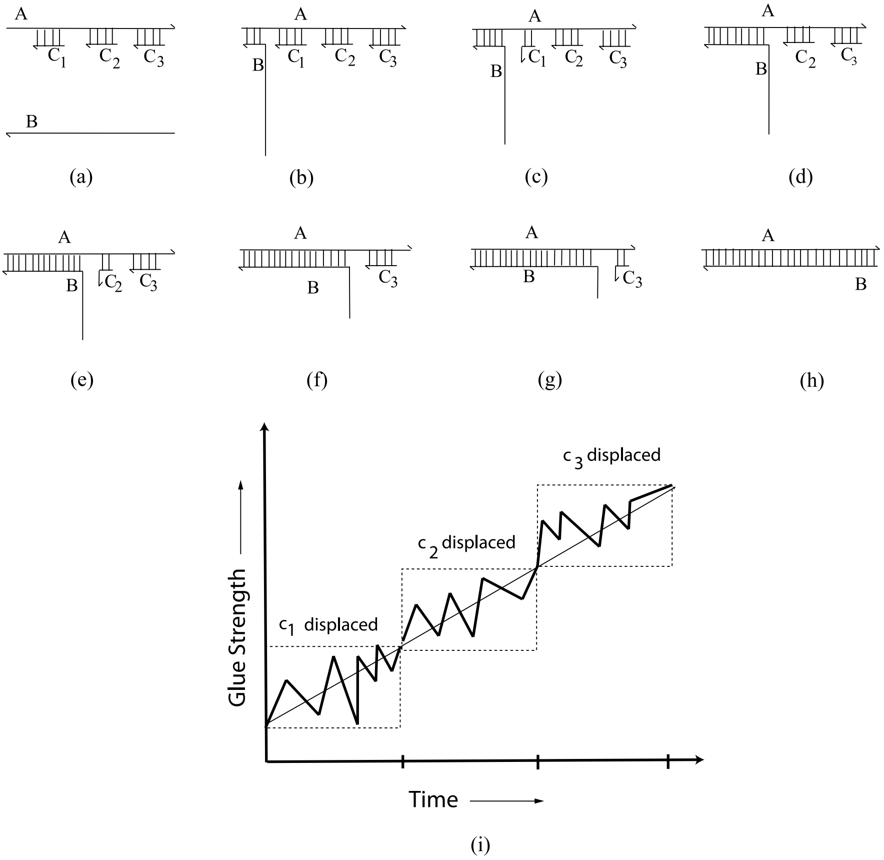


Fig. 2 (**a**) Illustrates the process of strand displacement. (**b**) Shows a single step of strand-displacement as single step of random walk. In (b), the numbers represent the number of DNA base pairs

that each base can form hydrogen bonds with the complementary base, also known as Watson–Crick base-pairing. The hydrogen bonding between complementary base pairs from two DNA strands results in their intertwining in the shape of a double helix, known as double stranded DNA (dsDNA). Individual separate strands are known as single stranded DNA (ssDNA). The direction of a DNA strand is defined in terms of its asymmetric ends, referred to as $5'$ and $3'$ ends. The $5'$ end terminates at the phosphate group attached to the fifth carbon atom in the sugar ring, while the $3'$ end terminates at hydroxyl group attached to the third carbon atom in the sugar-ring. In a double helix, direction of the nucleotides in one strand is opposite to their direction in the other strand.

The process of combining complementary, single stranded nucleic acids into a double stranded DNA molecule is called *DNA hybridization*. If the hydrogen bonds between the nucleotides in two hybridizing DNA strands build up sequentially, the total binding force between the two strands will increase with time up to the complete hybridization, which will provide a simple way of obtaining time-dependent glue strength between DNA tiles. However, even if we assume that the hybridization of two complementary DNA strands is instantaneous, we can design a multistep binding mechanism to implement the idea of time-dependent glue strength, which exploits the phenomenon of strand displacement.

Figure 2(a) illustrates the process of strand displacement in which strand B displaces strand C from strand A. Figure 2(b) illustrates one step during this process. At any time either the hybridization of B with A (and hence dehybridization of C from A) or hybridization of C with A (and hence dehybridization of B from A) can proceed with certain probability. Hence, we can model the strand displacement process as a random walk, with forward direction corresponding to hybridization between B and A, and backward direction corresponding to hybridization between C and A. A *one-dimensional unbiased random walk* is a process in which any step in forward or backward direction is taken with probability 0.5 independent of previous steps. The average straight-line distance between start and finish points of a one-dimensional random walk after n steps is on the order of $\sqrt{n}$, and hence expected number of steps to cover a distance n is $O(n^2)$ [15, 17, 28]. In order to model



(a) (b) (c) (d)

(e) (f) (g) (h)

(i)

Fig. 3 (**a**) to (**h**) Illustrate a mechanism by which strand displacement reaction is used to implement time-dependent glue between two pads. They show step by step removal of C_i's by B from A. In (**i**), an imaginary graph illustrates the variation of glue-strength between A and B w.r.t. time

the strand displacement, we can assume that the step length in this random walk is 1 base-pair long. Hence, if the length of C is n bases, the expected number of steps required for B to replace C is $O(n^2)$.

Next, we describe the design of the pads of DNA tiles with time-dependent glue using the above mechanism of strand displacement. To make the glue between pad A and pad B time-dependent, we need a construction similar to the one in Fig. 3(a). The strand representing pad A has various smaller strands (C_i's, called *protector strands*) hybridized to it as shown in Fig. 3(a). The strand B will displace these protector strands C_i sequentially.

Let the variable γ here will be the time required for B to displace all the C_i's. In the case when there are k different small strands C_i of length n_i attached to A, γ is $\sum_{i=1}^{k} n_i^2$.

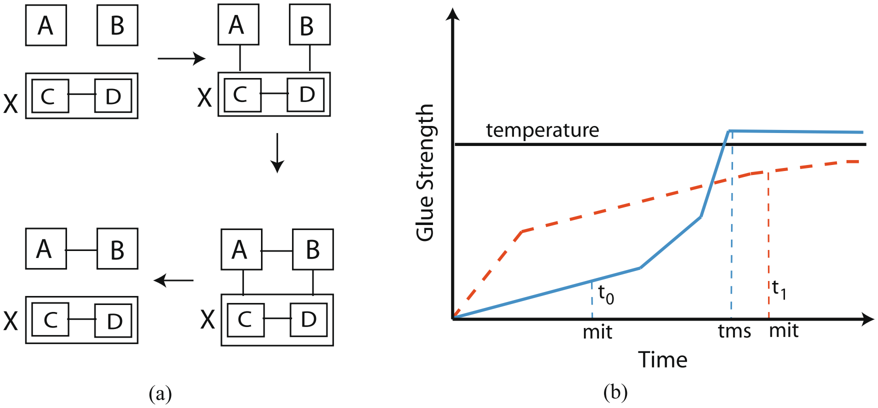
Figure 3 gives the step by step illustration of the above process. The variation of glue strength between A and B is shown in Fig. 3(i). By controlling the length of various C_i's (i.e. $n_1, n_2, \ldots, n_k$), we can control the glue-strength function g for a pair of tile-pads (or glues). Thus, we have shown a method to render the DNA tiles the characteristic of time-dependent glue strength.

An interesting property is that the individual strand displacement of B against C_i is modeled as an unbiased one dimensional random walk, but the complete process described above can be viewed as *roughly* monotonic. As shown in Fig. 3(i), the strength of the hybridization between strand A and strand B increases in a roughly monotonic fashion with the removal of every C_i. However during the individual competition between B and C_i, the increase is not monotonic.

4 Catalysis

Catalysis is the phenomenon in which an external substance facilitates the reaction of other substances, without itself being used up in the process. A catalyst provides an alternative route of reaction where the activation energy is lower than the original chemical reaction and increase the reaction rate. Catalysts participate in reactions but are neither reactants nor products of the reaction they catalyze. The following question was posed by Adleman [2]: Can we model the process of catalysis in self-assembly of tiles? In this section, we present a model for catalysis in self-assemby of tiles using our time-dependent glue model. Now, consider a super-tile $\mathcal{X}$ (composed of two attached tiles C and D) and two single tiles A and B as shown in Fig. 4(a). We describe below how $\mathcal{X}$ can serve as a catalyst for the assembly of A and B. Assume $t_0 = \mu(e(A), w(B))$ such that $g(e(A), w(B), t_0)$ is less than the temperature τ. Let $\mu(s(A), n(C)) = \mu(s(B), n(D)) = t_1 > t_0$. Also, assume $g(s(A), n(C), t_1) + g(s(B), n(D), t_1) < \tau$ and $g(e(A), w(B), t_1) \geq \tau$.

The graph in Fig. 4(b) illustrates an example set of required conditions for the glue-strength functions in the system. $A \cdot B$ represents a tile A bounded to a tile B.



(a)

(b)

Fig. 4 (**a**) Shows catalyst $\mathcal{X}$ with the tiles C and D catalyzes the formation of $A \cdot B$. (**b**) Shows the conditions required for catalysis in terms of the glue-strength function. *Solid line* shows the plot of $g(e(A), w(B), t)$ and *dashed line* shows the plot of $g(s(A), n(C), t) + g(s(B), n(D), t)$

To show that $\mathcal{X}$ acts as a catalyst, we first show that without $\mathcal{X}$ stable $A \cdot B$ cannot form. Next, we show that $A \cdot B$ will form when $\mathcal{X}$ is present and $\mathcal{X}$ will be recovered unchanged after the formation of $A \cdot B$.

Without $\mathcal{X}$ in the system, A and B can only be held in neighboring positions for time $t_0 = \mu(e(A), w(B))$, since $g(e(A), w(B), t_0) < \tau$. Hence, at t_0, A and B will fall apart.

However, in the presence of $\mathcal{X}$, the situation changes. Supertile $\mathcal{X}$ has two neighboring tiles C and D. Tiles A and B attach themselves to C and D as shown in Fig. 4(a). Since we let $\mu(s(A), n(C)) = \mu(s(B), n(D)) = t_1 > t_0$, tiles A and B are held in the same position for time t_1. By our construction, as shown in Fig. 4(b), the following two events will occur at time t_1:

- At t_1, the glue strength between A and B is $g(e(A), w(B), t_1) \geq \tau$, and hence A and B will be glued together. That is, in the presence of $\mathcal{X}$, A and B remain together for a longer time, producing stably glued $A \cdot B$.
- At t_1, the total glue strength between $A \cdot B$ and $\mathcal{X}$ is $g(s(A), n(C), t_1) + g(s(B), n(D), t_1) < \tau$, and the glued $A \cdot B$ will fall off $\mathcal{X}$. $\mathcal{X}$ is recovered unchanged from the reaction and the catalysis is complete. Now, $\mathcal{X}$ is ready to catalyze other copies of A and B.

Note that if only A (resp. B) comes in to attach with C (resp. D), it will fall off at the end of time $\mu(s(A), n(C))$ (resp. $\mu(s(B), n(D))$). If assembled $A \cdot B$ comes in, it will also fall off, at time t_1. These two reactions are futile reactions, and do not block the desired catalysis reaction. However, as the concentration of $A \cdot B$ increases and the concentration of unattached A and B decreases, the catalysis efficiency of $\mathcal{X}$ will decrease due to the increased probability of the occurrence of futile reaction between $A \cdot B$ and $C \cdot D$.

5 Self-replication

Self-replication process is one of the fundamental process of nature, in which a system creates copies of itself. For example, DNA is self-replicated during cell division and is transmitted to offspring during reproduction. A material device that can self-replicate is ambition of many engineering disciplines. The biggest incentive is to achieve a low manufacturing cost because self-replication avoids the costs of labor, capital, and distribution in conventional manufactured goods. In an evolving field like nanotechnology, manufacturing costs of molecular machines can become extremely large in the absence of self-replication. We discuss below an approach to model the process of self-replication in DNA tiles assembly using our time-dependent glue model.

Our approach is built on the above described process of catalysis: a product $A \cdot B$ catalyzes the formation of $C \cdot D$, which in turn catalyzes the formation of $A \cdot B$, and hence an exponential growth of self-replicated $A \cdot B$ and $C \cdot D$ takes place.

More precisely, let $t_0 < t_1$, and consider tiles A, B, C, and D, such that

$$\begin{aligned}\mu\big(e(A), w(B)\big) = \mu\big(e(C), w(D)\big) &= t_0,\\ \mu\big(s(A), n(C)\big) = \mu\big(s(B), n(D)\big) &= t_1,\\ g\big(e(A), w(B), t_0\big) = g\big(e(C), w(D), t_0\big) &< \tau,\\ g\big(e(A), w(B), t_1\big) = g\big(e(C), w(D), t_1\big) &> \tau,\\ g\big(s(A), n(C), t_1\big) + g\big(s(B), n(D), t_1\big) &< \tau.\end{aligned}$$

A system containing these four types of tiles has two states.

State 1. If there is no template $A \cdot B$ or $C \cdot D$ in the system, no assembled supertile exists since no two tiles can be held together long enough to form strong enough glue between them such that they become stably glued. Since $\mu(e(A), w(B)) = \mu(e(C), w(D)) = t_0$ and $g(e(A), w(B), t_0) = g(e(C), w(D), t_0) < \tau$, neither stable $A \cdot B$ nor stable $C \cdot D$ can form. Similarly, $\mu(s(A), n(C)) = \mu(s(B), n(D)) = t_1$, $g(s(A), n(C), t_1) < \tau$, and $g(s(B), n(D), t_1) < \tau$ implies that neither stable $A \cdot C$ nor stable $B \cdot D$ can form.

State 2. In contrast, if there is an initial copy of stable $A \cdot B$ in the system, self-replication occurs as follows. $A \cdot B$ serves as catalyst for the formation of $C \cdot D$, and $C \cdot D$ and $A \cdot B$ separate from each other at the end of the catalysis period, as described in Sect. 4; in turn, $C \cdot D$ serves as catalyst for the formation of $A \cdot B$. Thus, we have a classical self-replication system: one makes a copy of itself via its complement. The number of the initial template $(A \cdot B)$ and its complement $(C \cdot D)$ grows exponentially in such a system as long as there are sufficient numbers of free A, B, C, and D tiles to be made into pairs.

Hence, if the system is in state 1, it needs a triggering activity (formation of a stable $A \cdot B$ or $C \cdot D$) to go into state 2. Once the system is in state 2, it starts

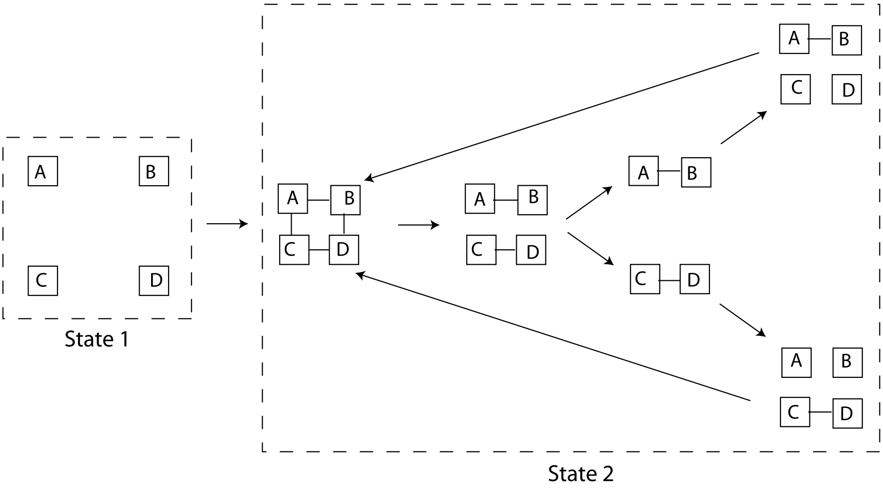


Fig. 5 A schematic of self-replication

the self-replication process. Figure 5 illustrates the process of self-replication in the assembly of tiles.

If the system is in state 1, then the triggering activity (formation of a stable $A \cdot B$ or $C \cdot D$) can take place only if A, B, C, D coposition themselves so that the east side of A faces the west side of B and the south side of A faces the north side of C, and at the same time the south side of B faces the north side of D. In such a situation, A and C will remain abutted until time t_1, B and D will remain abutted until time t_1, and A and B (and C and D) might also remain together for time t_1, producing stable $A \cdot B$ and stable $C \cdot D$. And this will bring the system to state 2. Such copositioning of 4 tiles is a very low probability event. However, among other conditions, appropriate copositioning of unstable $A \cdot B$ and unstable $C \cdot D$, or unstable $A \cdot C$ and unstable $B \cdot D$ can also perturb a system in state 1 and triggers tremendous changes by bringing the system to state 2 where self-replication occurs.

6 Tile Complexity Results

6.1 Tile Complexity Results for Thin Rectangles

In the ATA model, the tile complexity of assembling an $N \times N$ square is $\Theta(\frac{\log N}{\log\log N})$ [3, 32]. It is also known that the upper bound on the tile complexity of assembling a $k \times N$ rectangle in the ATA model is $O(k + N^{1/k})$ and that the lower bound on tile complexity of assembling a $k \times N$ rectangle is $\Omega(\frac{N^{1/k}}{k})$ [6]. For small values of k, this lower-bound is asymptotically larger than $O(\frac{\log N}{\log\log N})$. Here, we claim that in our model, as in the multitemperature model defined in [6], a $k \times N$ rectangle can be self-assembled using $O(\frac{\log N}{\log\log N})$ types of tiles, even for small values of k. The proof technique follows the same spirit as in [6].

Theorem 1 *In the time-dependent glue model, the tile complexity of self-assembling a $k \times N$ rectangle for an arbitrary integer $k \geq 2$ is $O(\frac{\log N}{\log\log N})$.*

Proof The tile complexity of self-assembling a $k \times N$ rectangle is $O(N^{\frac{1}{k}} + k)$ for the ATA model [6]. In the time-dependent glue model, we can use the similar idea as in [6] to reduce the tile complexity of assembling thin rectangles. For given k and N, build a $j \times N$ rectangle with $j > k$ such that the glues among the first k rows become strong after their μ (*minimum interaction time*), while the glues among the last $j-k$ rows do not become as strong. First, k rows are called *stable rows*, and last $j - k$ rows are called *volatile rows*. As such, these $j - k$ volatile rows, will disassemble from the assembly after certain time leaving the target $k \times N$ rectangle consisting of only the stable rows.

The tile set required to accomplish this construction is shown in Fig. 6, which is similar to the one used in [6]. For more detailed illustration of this tile set, refer to [6]. First, a j-digit m-base counter is assembled as follows. Starting from the west

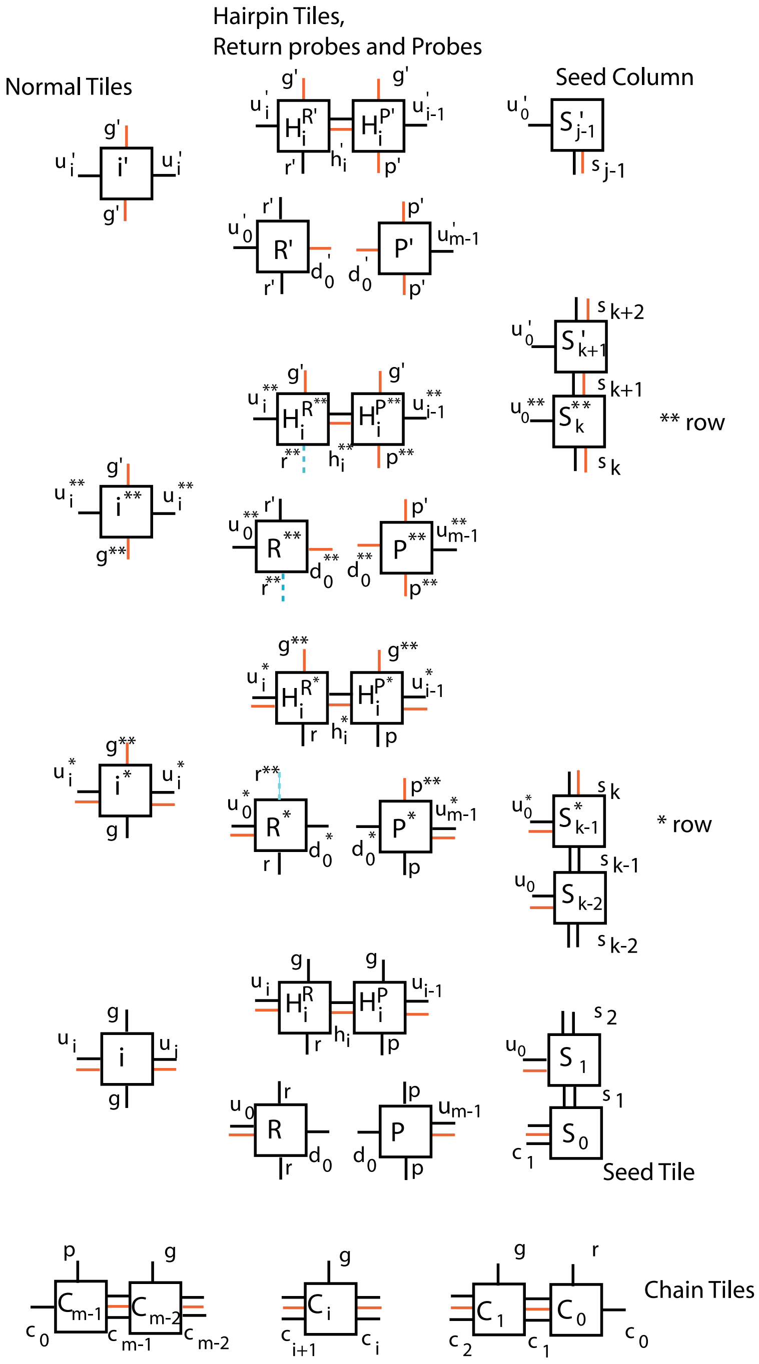


Fig. 6 Tile set to construct a $k \times N$ rectangle using only $O(N^{1/j} + j)$ tiles. The glue-strength functions of *red*, *blue*, and *black glues* are defined in the proof

edge of the seed tile, a chain of length m is formed in the first row using m chain tiles. At the same time, tiles in the seed column also start assembling. It should be noted that first k tiles in the seed column have sufficient glue strength and they are stable. Now starting from their west edges, the 0 normal tiles start filling the $m-1$ columns in the upper rows. Then the hairpin tiles H_1^P and H_1^R assemble in the second row, which causes the assembly of further m chain tiles in the first row, and the assembly of 1 normal tiles in the second row (and 0 normal tiles in the upper rows) in the next section of m columns. Generally speaking, whenever a C_{m-1} chain tile is assembled in the first row, probe tiles in the upper rows are assembled until reaching a row that does not contain an $m-1$ normal tile. In such a row, the appropriate hairpin tiles are assembled and this further propagates the assembly of return probe tiles downward until the first row is reached, where a C_0 chain tile gets assembled. This again starts an assembly of a chain of length m. The whole process is repeated until a $j \times m^j$ rectangle is assembled.

Next, we describe our modifications which are required for the $j-k$ upper volatile rows to get disassembled after the complete assembly of the $j \times m^j$ rectangle. First of all, we need to have a special $(k+1)$-th row ($**$ row), which will assemble to the north of the kth row ($*$ row), as shown in Fig. 6.

The operating temperature $\tau = 2$. Assume that for all glue-types, $\mu = t_0$ and $\gamma = t_1$. There are three kinds of glues shown in Fig. 6: black, red, and blue. Assume that the glue-strength function for a single black glue is $g_{\text{black}}(t)$, a single red glue is $g_{\text{red}}(t)$, and a single blue glue is $g_{\text{blue}}(t)$. They are defined as

$$g_{\text{black}}(t) = \begin{cases} \frac{4t}{5t_0}, & t < t_0, \\ \frac{4}{5} + \frac{t-t_0}{5(t_1-t_0)}, & t_0 \le t < t_1, \\ 1, & t \ge t_1, \end{cases}$$

$$g_{\text{red}}(t) = \begin{cases} \frac{2t}{5t_0}, & t < t_0, \\ \frac{2}{5} + \frac{t-t_0}{10(t_1-t_0)} & t_0 \le t < t_1, \\ \frac{1}{2} & t \ge t_1, \end{cases}$$

$$g_{\text{blue}}(t) = \begin{cases} \frac{2t}{5t_0}, & t < t_0, \\ \frac{2}{5}, & t \ge t_0. \end{cases}$$

Multiple glues shown on the same side of a tile in Fig. 6 are additive. For example, the glue-strength between C_i and C_{i+1} $(0 \le i \le m-2)$ is $2g_{\text{black}}(t) + g_{\text{red}}(t)$.

This system will start assembling like a base $N^{1/j}$ counter of j digits, as briefed above and detailed in [3, 6]. It will first construct a rectangle of size $j \times N$ using $N^{1/j} + j$ type of tiles. Once the rectangle is complete, the tile on the north-west corner will start the required disassembly of the upper $(j-k)$ volatile rows, which results in the formation of a $k \times N$ rectangle. We call these two phases *Assembly phase* and *Disassembly phase*, respectively, and describe them below.

Assembly Phase. In the assembly phase, we aim at constructing a $j \times N$ rectangle. In the time dependent model, the assembly proceeds as in the ATA model until the assembly of P^* tile in the kth row (the distinguished $*$ row). At this point, an $H^{R^{**}}$ tile is required to get assembled. However, when the $H^{R^{**}}$ tile is assembled in the $(k+1)$-th row, the total support on $H^{R^{**}}$ from its east neighbor is only $\frac{4}{5}+\frac{2}{5}<2$ at the end of μ. Thus, $H^{R^{**}}$ must obtain additional support; otherwise, it will get disassembled, blocking the desired assembly process. The additional support comes both from its south neighbor and its west neighbor. (1) On the south front, tile R^* can arrive and be incorporated in the kth row (the distinguished $*$ row) of the assembly. It holds $H^{R^{**}}$ for another time interval of μ and provides a support of $\frac{2}{5}$. Further note that during this second interval, an R tile can be assembled in the $(k-1)$-th row, and the R^* tile in the kth row will then have support 2 at μ, and hence stay attached. In addition, tile R has support 2 at μ, so it will also stay attached. Regarding $H^{R^{**}}$, the end result is that it receives an additional *stable* support $\frac{2}{5}$ from its south neighbor. However, the maximum support from both the south and the east is at most $1+\frac{1}{2}+\frac{2}{5}$, which is still less than $\tau=2$. Fortunately, additional rescue comes from the west. (2) On the west front, an i^{**} tile can get attached to $H^{R^{**}}$, and stabilize it by raising its total support above 2. However, this support is insufficient, in the sense that i^{**} itself needs additional support from its own west and south neighbors to stay attached. If this support cannot come in time, that is, before μ, i^{**} will get disassembled, in turn causing the disassembly of $H^{R^{**}}$. The key observation here is that this assembly/disassembly is a reversible dynamic process: the disassembly may stop and start going backward (i.e., assembling again) at any point. Thus, in a dynamic, reversible fashion, the target structure of the assembly phase, namely the $j\times N$ rectangle, can be eventually constructed.

The above added complication is due to the fact that we require the $H^{R^{**}}$ tiles in the $(k+1)$-th row to get a total support of <2 from the south and the east. This is crucial because during the subsequent disassembly phase (as we describe next) the desired disassembly can only carry through if the total support of each volatile tile from the south and the east is <2.

Disassembly Phase. In the disassembly phase, we will remove the $j-k$ volatile rows, and reach the final target structure, a $k\times N$ rectangle. Once the $j\times N$ rectangle is complete, the tile T at the north-west corner (P' tile in the jth row) initiates the disassembly. When the μ of the glue-pairs between tile T and its neighbors is over, tile T will get detached because the total glue strength that it has accumulated is $\frac{4}{5}+\frac{2}{5}<\tau=2$. Note that unlike the above case for $H^{R^{**}}$, no additional support can come from the west for tile T since T is the west-most tiles. As such, T is doomed to get disassembled. With T gone, T's east neighbor will get removed next, since it now has a total glue strength $\leq 1+\frac{1}{2}<\tau$. Similarly, all the tiles in this row will get removed one by one, followed by the removal of the tiles in the next row (south row). Such disassembly of the tiles continues until we are left with the target rectangle of size $k\times N$, whose constituent tiles, at this stage, all have a total glue strength no less than $\tau=2$, and hence stay stably attached.

Note that, similar as in the assembly phase, the volatile tiles that just got removed might come back. But again, ultimately they will have to *all* fall off (after the μ), and produce the desired $k\times N$ rectangle.

Concluding the Proof. We can construct a $k \times N$ rectangle using $O(N^{1/j} + j)$ type of tiles (where $j > k$). As in [6], it can be reduced to $O(\frac{\log N}{\log\log N})$ by choosing $j = \frac{\log N}{\log\log N - \log\log\log N}$. □

6.2 Further Tiling Assemblies for Interesting Shapes

Thin rectangles can serve as building blocks for the construction of many other interesting shapes. One example is a square of size $N \times N$ with a large square hole of size $k \times k$ (for $k \sim N$). Under the ATA model, the lower bound can be shown to be $\Omega(\frac{(k)^{\frac{2}{N-k}}}{N-k})$ by a lower bound argument similar to the one in [6]. Note that as $N - k$ decreases, i.e., the square hole in the square increases, the lower bound increases. In the case when $N - k$ is smaller than $\frac{\log N}{\log\log N - \log\log\log N}$, the lower bound is more than $\frac{\log N}{\log\log N}$. In the case when $N - k$ is a small constant, the complexity is almost N^c, where c is some constant < 1. However, in the time-dependent model, the tile complexity of this shape can be reduced to $O(\frac{\log k}{\log\log k})$ even for small values of $N - k$, using our thin rectangle construction.

The basic idea is quite simple. We sequentially grow four different thin rectangles in four different directions: one rectangle northward, one westward, one southward, and one eastward. The dimensions of each of these rectangles is $(\frac{N-k-2}{2}) \times (k+2)$. They will make up the major part of the square's sides as shown in Fig. 7(a). The required tile set consists of four different groups of tile sets: each one growing a $(\frac{N-k-2}{2}) \times (k+2)$ rectangle in one direction. Each of these rectangles can be constructed by $O(\frac{\log k}{\log\log k})$ types of tiles as discussed in the proof of Theorem 1. We

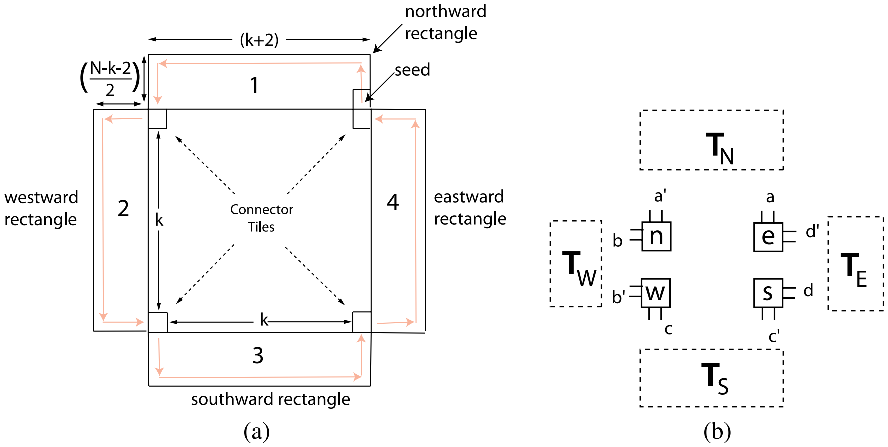
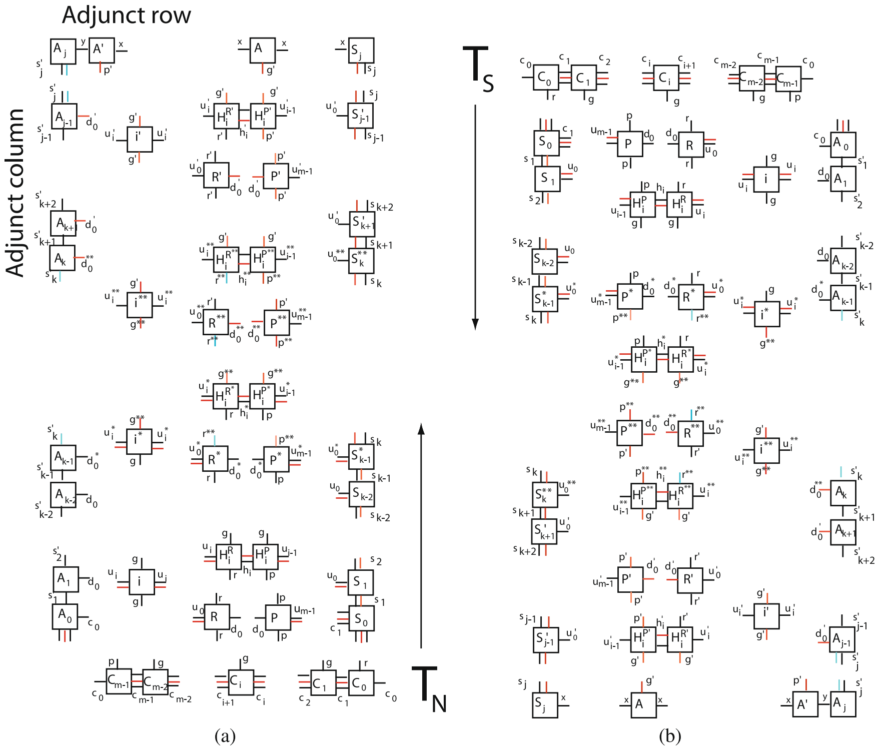


Fig. 7 (**a**) Direction of the *red arrow* shows the direction of construction of a square with a hole, starting from the indicated seed. (**b**) A complete tile set for the square with hole. Sets T_N, T_S, T_W, T_E are shown in Figs. 8, 9, and 10



(a) (b)

Fig. 8 (**a**) Displays the tiles from the sets T_N required for the construction of $N \times N$ square with a hole of size $k \times k$ in the center. (**b**) Displays the tiles from the sets T_S required for the construction of $N \times N$ square with a hole of size $k \times k$ in the center. It should be noted that symbols in (a), (b), Figs. 9, and 10 are from different namespaces. It means that a glue-symbol x in T_N is different from a glue-symbol x in T_S, T_W, or T_E, and they cannot interact

refer to these groups of tile sets as T_N, T_W, T_S, and T_E (Fig. 7 (b)). The complete details of tile sets T_N, T_W, T_S, and T_E are shown in Figs. 8, 9, and 10.

As shown in the center in Fig. 7(b), we need some additional tiles (tiles n, e, s, and w) to connect these four different rectangles with each other in order to complete the desired square with a hole. We call them *connector tiles*. Note that the glues on the sides of connector tiles match with the glues of *seed* tiles of the corresponding rectangles. After the completion of one rectangle, the corresponding connector tile should assemble and provide a path for the assembly of another rectangle. For example, the assembly of the connector tile n takes place after the assembly of the west-most column of the northward rectangle from the tile set T_N, and triggers the assembly of the westward rectangle.

In each of the thin rectangles, a special row and a special column is needed that can assist the assembly of the corresponding connector tile. We call these special rows and columns as *adjunct row* and *adjunct column*. The tiles required for the assembly of the adjunct row and column in the northward rectangle are shown in

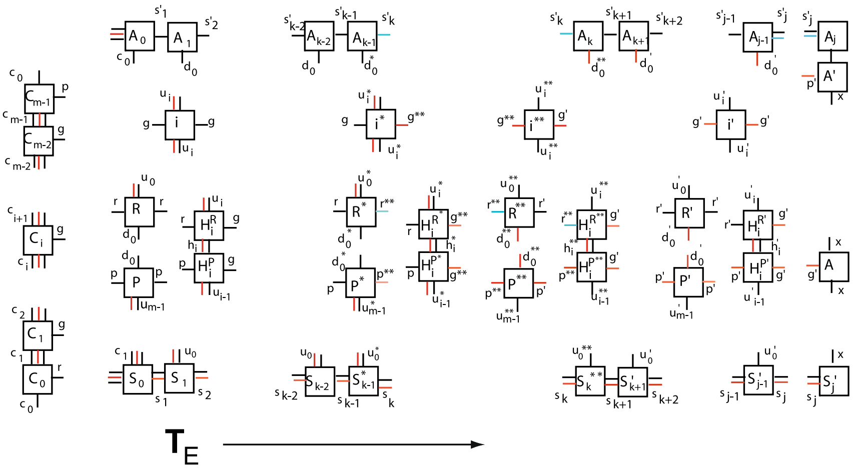


Fig. 9 Figure displays the tiles from the sets T_N and T_S required for the construction of $N \times N$ square with a hole of size $k \times k$ in the center

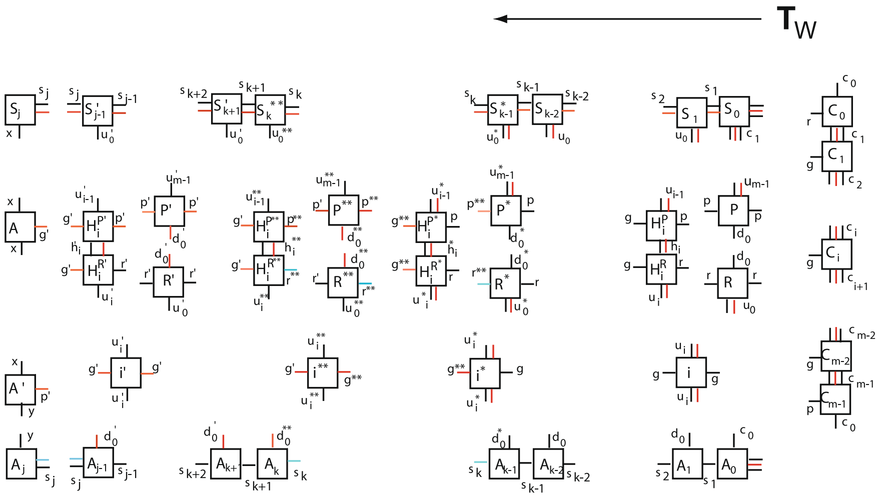


Fig. 10 Figure displays the tiles from the set T_W required for the construction of $N \times N$ square with a hole of size $k \times k$ in the center

Fig. 8. Note that the glues on the sides of these tiles are designed in such a way that they do not inhibit the disassembly phase in the construction of the corresponding thin rectangle.

Finally, we have gaps at the four corners this $N \times N$ square, and a $(k+2) \times (k+2)$ square hole in the center with exactly one tile present at each corner of the hole (Fig. 7). A constant number of type of tiles, referred to as *filler tiles*, will be needed to fill in these gaps, and obtain an $N \times N$ square with a $k \times k$ hole at its center.

The complete tile set T_N is the tile set described in the proof of Theorem 1 along with the tiles for adjunct row and column. The boundary tiles in T_N are modified slightly so that they can assist the assembly of appropriate filler tiles when required. Tile set T_W is formed by rotating every tile of T_N anticlockwise by 90°. It should be noted that a totally disjoint set of symbols for the glues should be used in T_W to avoid any interaction with the tiles in T_N. Similarly, the tile sets T_S and T_E can be obtained by further anticlockwise rotation of T_W by 90° and 180°, respectively.

Thus, the total number of tiles required is the sum of tiles required for each of the four thin rectangles, four connector tiles, constant number of filler tiles, and the tiles for adjunct row and column in each of the four rectangles. This is upper bounded by $O(\frac{\log k}{\log\log k})$. The assembly will grow in the manner shown in Fig. 7(a). Assuming without loss of generality that the *seed* is the seed tile of rectangle 1. Then first rectangle 1 will be constructed; then the connector to rectangle 1 and 2 will assemble; then rectangle 2 will assemble; then connector to rectangle 2 and 3; then rectangle 3; then connector to 3 and 4; finally rectangle 4 will get assembled. It should be noted that the filler tiles can assemble anytime during the assembly, whenever they get enough support to hold them.

7 Discussion and Future Work

In this paper, we defined a model in which the glue strength between tiles depends upon the time they have been abutting each other. Under this model, we demonstrate and analyze catalysis and self-replication, and show how to construct a thin $k \times N$ rectangle using $O(\frac{\log N}{\log\log N})$ tiles for constant $k > 0$. The upper bound on assembling a thin rectangle is obtained by applying similar assembly strategy as in the multitemperature model [6]. Thus, an interesting question is whether the multitemperature model can be simulated using our time-dependent model. It is also an open problem if under our model the lower bound of $\Omega(\frac{\log N}{\log\log N})$ for the tile complexity of an $N \times N$ square can be further improved.

Another interesting direction is to study the kinetics of the catalysis and self-replication analytically. Winfree's kinetic model [40] can be used to study them, but the challenge here is that the rate constant for the dissociation for a particular species varies with time because of changing glue strengths of its bonds. This makes the analytical study hard. However, these catalytic and self-replicating systems can be modeled as a continuous time Markov chain, and studied using computer simulation to obtain empirical results.

Acknowledgements The work is supported by NSF Grants CCF-0523555 and CCF-0432038.

References

1. http://mrsec.wisc.edu/edetc/selfassembly/

2. Adleman L (2000) Towards a mathematical theory of self-assembly. Tech Rep 00-722, University of Southern California
3. Adleman L, Cheng Q, Goel A, Huang M (2001) Running time and program size for self-assembled squares. In: Proceedings of the thirty-third annual ACM symposium on theory of computing. ACM, New York, pp 740–748
4. Adleman L, Cheng Q, Goel A, Huang M, Kempe D, de Espans P, Rothemund P (2002) Combinatorial optimization problems in self-assembly. In: Proceedings of the thirty-fourth annual ACM symposium on theory of computing. ACM, New York, pp 23–32
5. Adleman L, Kari J, Kari L, Reishus D (2002) On the decidability of self-assembly of infinite ribbons. In: Proceedings of the 43rd symposium on foundations of computer science, pp 530–537
6. Aggarwal G, Cheng Q, Goldwasser MH, Kao M, de Espanes PM, Schweller RT (2005) Complexities for generalized models of self-assembly. SIAM J Comput 24:1493–1515
7. Angelov S, Khanna S, Visontai M (2008) On the complexity of graph self-assembly in accretive systems. Nat Comput Int J Arch 7:183–201
8. Bowden N, Terfort A, Carbeck J, Whitesides G (1997) Self-assembly of mesoscale objects into ordered two-dimensional arrays. Science 276(11):233–235
9. Bruinsma R, Gelbart W, Reguera D, Rudnick J, Zandi R (2003) Viral self-assembly as a thermodynamic process. Phys Rev Lett 90(24):248101
10. Chelyapov N, Brun Y, Gopalkrishnan M, Reishus D, Shaw B, Adleman L (2004) DNA triangles and self-assembled hexagonal tilings. J Am Chem Soc 126:924–925
11. Chen H, Cheng Q, Goel A, Huang M, de Espanes P (2004) Invadable self-assembly: combining robustness with efficiency. In: Proceedings of the 15th annual ACM–SIAM symposium on discrete algorithms (SODA), pp 890–899
12. Cheng Q, de Espanes P (2003) Resolving two open problems in the self-assembly of squares. Tech Rep 03-793, University of Southern California
13. Cheng Q, Goel A, Moisset P (2004) Optimal self-assembly of counters at temperature two. In: Proceedings of the first conference on foundations of nanoscience: self-assembled architectures and devices
14. Cook M, Rothemund PWK, Winfree E (2004) Self-assembled circuit patterns. In: DNA based computers 9. Lecture notes in computer science, vol 2943. Springer, Berlin, pp 91–107
15. Feller W (1968) An introduction to probability theory and its applications, vol 1
16. Fujibayashi K, Murata S (2005) A method for error suppression for self-assembling DNA tiles. In: Lecture notes in computer science, vol 3384. Springer, Berlin, pp 113–127
17. Hughes BD (1995) Random walks and random environments, vol 1: random walks. Oxford University Press, New York
18. Kao M, Schweller R (2006) Reduce complexity for tile self-assembly through temperature programming. In: Proceedings of 17th annual ACM–SIAM symposium on discrete algorithms (SODA). ACM, New York, pp 571–580
19. Klavins E (2004) Directed self-assembly using graph grammars. In: Foundations of nanoscience: self assembled architectures and devices, Snowbird, UT
20. Klavins E, Ghrist R, Lipsky D (2004) Graph grammars for self-assembling robotic systems. In: Proceedings of the international conference on robotics and automation
21. LaBean TH, Yan H, Kopatsch J, Liu F, Winfree E, Reif JH, Seeman NC (2000) Construction, analysis, ligation and self-assembly of DNA triple crossover complexes. J Am Chem Soc 122:1848–1860
22. Lagoudakis M, LaBean T (2000) 2-D DNA self-assembly for satisfiability. In: DNA based computers V. DIMACS, vol 54. American Mathematical Society, Providence, pp 141–154
23. Liu D, Wang M, Deng Z, Walulu R, Mao C (2004) Tensegrity: construction of rigid DNA triangles with flexible four-arm DNA junctions. J Am Chem Soc 126:2324–2325
24. Mao C, Sun W, Seeman NC (1999) Designed two-dimensional DNA holliday junction arrays visualized by atomic force microscopy. J Am Chem Soc 121:5437–5443
25. Park SH, Pistol C, Ahn SJ, Reif JH, Lebeck AR, Dwyer C, LaBean TH (2006) Finite-size, fully addressable DNA tile lattices formed by hierarchical assembly procedures. Angew Chem Int Ed 45:735–739

26. Reif JH, Sahu S, Yin P (2005) Complexity of graph self-assembly in accretive systems and self-destructible systems. In: Lecture notes in computer science. Springer, Berlin, pp 257–274
27. Reif JH, Sahu S, Yin P (2006). Compact error-resilient computational DNA tilings. In: Nanotechnology: science and computation, pp 79–103
28. Revesz P (1990) Random walk in random and non-random environments. World Scientific, Singapore
29. Robinson R (1971) Undecidability and non periodicity of tilings of the plane. Invent Math 12:177–209
30. Rothemund P (2000) Using lateral capillary forces to compute by self-assembly. Proc Natl Acad Sci USA 97(3):984–989
31. Rothemund P (2001) Theory and experiments in algorithmic self-assembly. PhD thesis, University of Southern California
32. Rothemund P, Winfree E (2000) The program-size complexity of self-assembled squares (extended abstract). In: Proceedings of the thirty-second annual ACM symposium on theory of computing. ACM, New York, pp 459–468
33. Sa-Ardyen P, Jonoska N, Seeman NC (2003) Self-assembling DNA graphs. Nat Comput 2:427–438
34. Schulman R, Lee S, Papadakis N, Winfree E (2004) One dimensional boundaries for DNA tile self-assembly. In: DNA based computers 9. Lecture notes in computer science, vol 2943. Springer, Berlin, pp 108–125
35. Schulman R, Winfree E (2005) Programmable control of nucleation for algorithmic self-assembly. In: Lecture notes in computer science, vol 3384. Springer, Berlin, pp 319–328
36. Schulman R, Winfree E (2005) Self-replication and evolution of DNA crystals. In: The 13th European conference on artificial life (ECAL)
37. Soloveichik D, Winfree E (2006) Complexity of compact proofreading for self-assembled patterns. In: Lecture notes in computer science, vol 3892. Springer, Berlin, pp 305–324
38. Soloveichik D, Winfree E (2007) Complexity of self-assembled shapes. SIAM J Comput 36:1544–1569
39. Wang H (1961) Proving theorems by pattern recognition, II. Bell Syst Tech J 40:1–41
40. Winfree E (1998) Simulation of computing by self-assembly. Tech Rep 1998.22, Caltech
41. Winfree E (2006). Self-healing tile sets. In: Nanotechnology: science and computation, pp 55–78
42. Winfree E, Bekbolatov R (2004) Proofreading tile sets: error correction for algorithmic self-assembly. In: DNA based computers 9. Lecture notes in computer science, vol 2943. Springer, Berlin, pp 126–144
43. Winfree E, Liu F, Wenzler LA, Seeman NC (1998) Design and self-assembly of two-dimensional DNA crystals. Nature 394(6693):539–544
44. Winfree E, Yang X, Seeman NC (1999) Universal computation via self-assembly of DNA: some theory and experiments. In: Landweber L, Baum E (eds) DNA based computers II. DIMACS, vol 44. American Mathematical Society, Providence, pp 191–213
45. Yan H, LaBean TH, Feng L, Reif JH (2003) Directed nucleation assembly of DNA tile complexes for barcode-patterned lattices. Proc Natl Acad Sci USA 100(14):8103–8108
46. Yan H, Park SH, Finkelstein G, Reif JH, LaBean TH (2003) DNA-templated self-assembly of protein arrays and highly conductive nanowires. Science 301(5641):1882–1884

The Perils of Polynucleotides Revisited

Nadrian C. Seeman

Abstract DNA computing relies on the successful implementation of physical chemistry techniques involving oligonucleotides of prescribed sequence. Our laboratory has been involved in the assembly and manipulation of designed oligonucleotides in order to pursue studies in genetic recombination and nanofabrication. We have constructed a large number of unusual branched DNA motifs used to build a variety of DNA objects, lattices, and nanomechanical devices. Our experience with these systems has uncovered a large number of experimental pitfalls that may confront individuals working with DNA-based computation. A decade ago, we presented our experience in this area in the hope that we could help investigators to anticipate experimental problems that affect DNA computing schemes. Here, we review these points from the vantage point of further experience, indicating both modifications to the original criteria and new points as well.

> *...that to which we return with the greatest pleasure possesses*
> *the greatest power...*
> *Samuel Taylor Coleridge*

1 Introduction

It has been over a decade since Adleman performed the first experimental demonstration of DNA-based computation [1]. In that time, the endeavor has matured markedly. Many individuals have participated in it, and many lessons have been learned. A parallel enterprise, structural DNA nanotechnology, which dates from the early 1980s [2] has also grown during that time. This area entails the use of self-assembled nucleic acid constructs for nonbiological purposes. Many of those drawn to DNA-based computation have also participated in structural DNA nanotechnology (for a review of the area; see Ref. [3]), because the techniques and approaches of the two areas share many characteristics. The leadership that Grzegorz Rozenberg provided in establishing the organizational framework for the two communities was a major contribution to this phenomenon, and all of us owe him a debt of gratitude for his activities.

I come from the structural DNA nanotechnology community, an enterprise based on the idea of using branched DNA and cohesive single strands (primarily sticky

N.C. Seeman (✉)
Department of Chemistry, New York University, New York, NY 10003, USA
e-mail: ned.seeman@nyu.edu

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_12, © Springer-Verlag Berlin Heidelberg 2009

ends) to hold them together [2]. The goal of this effort are the assembly of nanoscale objects, nanomechanical devices, and designed periodic matter. The applications of these targets are to act as a way to solve the crystallization problem for biological macromolecules, for use as potential components in nanofabrication, and to develop nano-manufacturing methods for producing novel materials. In the mid-1990s, I wrote a paper in which I summarized our experiences in structural DNA nanotechnology to prevent those engaged in DNA-based computation from having to reinvent a number of wheels or from repeating the mistakes we made during the founding of structural DNA nanotechnology [3]. I think it is time now to review the points brought up in that paper, and to see which of them are relevant to the work going on today. There has been a lot of progress, and there have been many new developments, and it is valuable to discuss some of these issues in this context. In addition, there are some issues of which I was unaware at the time, and there were further points that my background made implicit to me, but which I find frequently necessary to mention to novices in the area.

2 Old Rules and New Perspectives

In this section, I will list the points that I made in the 1998 paper, and then annotate them with comments based on developments in the intervening decade.

[1] Symmetry Is Inimical to Control

If one treats symmetry in its broadest sense, the equal, or nearly equal free energies from two possible outcomes of an experiment, it is clear that the greatest control over molecular associations will be derived from systems that minimize symmetry, particularly sequence symmetry.

This point certainly applies to small systems, such as DX tiles [4] *or other motifs. However, Mao and colleagues* [5] *have shown that one can actually utilize symmetric systems to save materials, to minimize stoichiometry issues, and to increase the sizes of arrays. Yan and colleagues* [6] *have exploited symmetry so as to build finite arrays from limited numbers of tiles. Rothemund's introduction of DNA origami* [7] *is a large system that ignores sequence symmetry completely, without losing control. The issue is mostly a matter of resolution, and the* 6 *nm pixilation that Rothemund seeks is largely immune to the sequence symmetry considerations that are so important to the development of smaller high resolution arrangements.*

[2] Non-Watson–Crick Pairing in Unusual Structures

Watson–Crick pairing appears to be favored for linear duplex DNA. However, when it is not available, or when it is difficult to achieve, non-Watson–Crick pairing can occur. Every base appears capable of pairing with every other base, including itself [8].

*This point remains correct. Nevertheless, there are some stable arrangements of non-Watson–Crick base pairing that have been exploited in structural DNA nanotechnology. These include, G*4 *motifs* [9], *I motifs* [10], *and osculating interactions between RNA loops* [11]. *A crystal structure from Paukstelis et al.* [12] *demonstrates the spontaneous formation of parallel non-Watson–Crick interactions.*

[3] The Importance of Hybridization Protocols

One must make sure that the pairing structure one wants to achieve has an opportunity to overcome kinetic barriers. We routinely heat all mixtures to 90°C for a few minutes, and cool the structure slowly before using it. We also check for the presence of the structure on nondenaturing gels run under the conditions of the experiment, particularly the temperature and the same DNA and divalent cation concentrations.

In addition to choosing an appropriate hybridization protocol, it is key to note that one should beware of the possibility of inserting kinetic traps into the hybridization protocol. For example, GC-rich ends of a strand designed to form parallel crossovers might anneal first, leading to the inability of the central part of the molecule to do the appropriate wrapping necessary to produce the designed topology [13].

[4] The Importance of DNA Concentration and Environment

It is critical to use concentrations of DNA that are appropriate to forming the complex of interest. The significance of appropriate divalent cations, pH, and appropriate temperatures cannot be overestimated.

It is always important to remember that low concentrations of DNA, such as the 1–10 *nM concentrations typically used with DNA origami systems must have sticky ends that are long enough to cohere at the temperatures being used (particularly at room temperature).*

[5] Proper Estimation of DNA Dimensions

Often a crude model is inadequate to estimate the size of noncanonical DNA features, such as loops between double helical segments. It is useful to vary such parameters experimentally so as to optimize their design.

There is better modeling software available to day (e.g., [14, 15]) *than was available previously* [16]. *Nevertheless, it is important that models not be taken too literally. It is easy to introduce braided crossovers* [17] *into models when the programs do not know about the topological details of the system. Likewise, physical modeling should not be used as a Procrustean bed: There is a lot of flexibility in DNA,* *and one cannot always count on idealized structures, for which the force constants are often unknown, to be the materials with which one is dealing. For example, the* 3*-helix bundle* [18] *does not fit the criteria of a minimally-strained nanotube* [15], *yet it is readily constructed. On a smaller scale, one would not have imagined that the tensegrity triangle from Mao's group would be readily built, but it is a very successful structure* [19].

[6] Experimental Determination of DNA Features

One cannot assume that unusual DNA structures assume the shapes that one feels most comfortable drawing for them. Branched junctions were found not to look like crosses [20], tRNA molecules were found not to look like any of their predicted tertiary structures [21], and the overall range of structures available to any nucleic acid molecule is often much larger than one wishes to believe. Furthermore, that structure is often a function of conditions. One should predicate nothing on an unverified DNA tertiary structure.

Another point to emphasize is the resolution and range of the technique. Hydroxyl radical autofootprinting [20] *gives information on the nucleotide level* (∼0.5 *nm*) *for bulk materials. The most popular method in use today for structural characterization is atomic force microscopy* (*AFM*)*. This method provides information on individual molecules or on periodic and aperiodic arrays at* ∼5 *nm resolution. It is worth remembering that the finest structural tool, X-ray diffraction, is capable of providing information of* ∼0.1–0.3 *nm resolution, if the sample is sufficiently well ordered.*

[7] The Fidelity of Ligation Is High, but Not Perfect

A "hungry" ligase molecule may well ligate the wrong molecule to a sticky end, if it bears a close resemblance to the target molecule.

In the period since the previous paper was written, our laboratory has abandoned ligation to the extent possible. We find that ligation is often highly inefficient, as low as 70% *per nick in branched systems. By contrast, systems in which self-assembly is needed without ligation are much more efficient. Thus, it is possible to get all of the component strands of a DX molecule, for example, to associate into the motif producing a yield of* ∼100%. *Getting DX molecules to associate further, via sticky ends is similar to any problem in crystallization, where some big crystals and some small crystals are always produced.*

[8] DNA Molecules Breathe

The bases of DNA molecules unpair at times, and strands of the same sequence can displace each other. Fraying at the ends of duplex segments occurs often, and it may be rampant at higher temperatures. A well-known manifestation of this phenomenon is branch migration, an isomerization that naturally-occurring branched molecules undergo [22]: One cannot assume that particular molecules will remain paired noncovalently for very long time periods.

In recent years, dynamic species have become central to structural DNA nanotechnology [23] *and to programmed nucleic acid circuitry* [24, 25]. *Consequently, branch migration has become a central phenomenon exploited in nanomechanical devices, following the pioneering work of Yurke et al.* [26]. *It is worth noting that this process is affected by the presence of drugs and the divalent cations that are in the solution.*

[9] Long and Loose Closed DNA Molecules Form Topoisomers

The plectonemic nature of double helical DNA helices results in catenated closed molecules. There will be a distribution of topological linkage in long molecules [27]. Topological impurity can complicate the interpretation of results involving catenated molecules.

This point remains valid, but does not seem to be relevant to most activities at this time.

[10] Affinity Binding Is a Filtration Process

Affinity binding, such as biotin-streptavidin binding, is effective, but often imperfect. Multiple tags may be needed to achieve desired purification, because the tag may not be accessible to the receptor in all conformations.

This point remains valid, but there seem to be no major new issues involving it.

[11] Complex Ligation Mixtures Can Lead to Complex Products

The ligation of molecules that can form cyclic species can often be complicated by numerous products in addition to the target products.

Unwanted ligation in most systems can be controlled by limiting DNA concentration, but complex topological species are difficult to avoid. So far, this does not seem to be a major issue.

[12] Separation of Hydrogen-Bonded Molecules in Native Conditions Is Often Ineffective

In our experience, purification is most effective under totally denaturing conditions. To effect such separations, one must form topologically closed molecules. Nicked and unnicked hydrogen bonded complexes comigrate on gels in nondenaturing conditions.

This lesson remains correct.

[13] Restriction Endonucleases Often Produce Partial Digestion Products

This is particularly true if the substrate is not simple double-helical DNA. One must test all protocols for the effectiveness of the restriction enzymes involved, and it is often necessary to find means to remove undigested material.

One cannot expect synthetic molecules to be prepared completely free of errors. There seem always to be small remnants of impure materials that will be indigestible to restriction enzymes. If complete digestion is needed, the DNA should be prepared enzymatically using nucleoside triphosphates [28]. *Polymerases seem to be subject to the same issues* [29].

[14] Multimerization of Cyclic Structures Can Occur

Discrete target structures involve cyclic pairing schemes, e.g., $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$. One hopes that the final molecule "1" and the first molecule "1" are the same molecule, but cyclic schemes that do not take steps to ensure the identity of the two "1" molecules are in danger of multimerization if this phenomenon will promote favorable interactions or decrease unfavorable ones.

As with point 11 *above, control of concentration is key.*

[15] Base Stacking Is Often the Determining Interaction

It is tempting to build up hydrogen bonding schemes when one is designing DNA molecules, but base stacking may well turn out to be the dominant physical interaction in the system.

The importance of base stacking continues to be of key importance. In an era of complex DNA motifs, blunt ends are often found to be unexpectedly stacked. Overhangs often combat this tendency successfully. In addition, designs predicated on the notion that a system will not stack often are frustrated by the tendency of the system to stack. Switchback DNA [30] *and antijunctions* [31] *are excellent examples of this phenomenon.*

[16] Treat DNA as a Physical Chemical System

In dealing with DNA molecules, it is necessary to examine them under the conditions in which they will be used, even if that is sometimes inconvenient. It is useful to vary the temperature, concentration of DNA and the solution environment, and if using electrophoresis, the gel concentration. So, as to develop a system that works well for any purpose, one must vary the parameters that define the system, to understand its strengths and limitations. There are many "bibles" available to provide molecular biological protocols (e.g., [32]). These sources are invaluable, but we suggest that levels of chemical efficiency that lead to successful molecular biology are often inadequate for successful chemistry, and that each protocol be examined carefully and not followed blindly. Likewise, it can be useful to ignore enzyme optima, and concentrate on conditions that ensure that the desired DNA complex is present in the experiment.

These points remain correct. It is often tempting to prepare a sample in a series of buffers, and then to expect that putting the samples into a gel with a different running buffer will yield a proper analysis. This is false—the running buffer will shortly become the buffer in which the sample is sitting, and the DNA will equilibrate to that condition. When dealing with problems and system optimization, it is really key to test the system under the exact conditions in which the experiment will be run. Thus, a system expected to require high concentrations for a physical experiment may yield multimers when examined on nondenaturing gels; this is a key precaution to take. Likewise, a system that will be used at 4°*C should not be tested at room temperature.*

3 New Points

[17] Crude DNA Is Impure Material

DNA should almost always be purified. In small systems, such as DX arrays, poor array formation will result from impure DNA. DNA origami seems somewhat resistant to the need for purification, but no detailed study has been conducted to see the impact of using pure staple strands.

[18] DX Cohesion Is More Effective than Simple Sticky-Ended Cohesion

In recent work, we have found that motifs designed to form lattices but that seem somewhat intractable to crystal formation are much more likely to do so when there are two sticky ends than a single one [33].

[19] Robust Devices are Necessary for a Nanorobotics That Emulates Macro-scale Robotics

When designing DNA-based nanomechanical devices, it is important to make sure that they are robust devices. A robust device is a system that under operating conditions behaves like a macroscopic device. This means that it neither multimerizes nor dissociates during transitions, so that there are discrete end points.

[20] Two Contexts That Are Not Identical Are Different

It is extremely important to recognize that favored structures are often found in a particular context. For example, we found that a four-stranded PX system required

6–8 nucleotide pairs in the major groove and 5 nucleotide pairs in the minor groove [30]; at low concentrations, 9 nucleotide pairs could also occupy the major groove. Shih et al. [34] were able to use the PX motif for cohesion with 6 nucleotide pairs in the major groove and only four nucleotide pairs in the minor groove when a pair of hairpins were used. We have found recently that within the context of a single plasmid it is possible to extend the major groove size to 10 or even 11 nucleotide pairs. Each context was different, and had to be judged from that perspective only.

4 Concluding Comments

In this article, I have tried to annotate the recommendations that I made in the previous article, based on the experience of the intervening decade. In addition, I have included a few new points that have arisen. One other point that I mentioned in the original article should be reiterated: The newcomer to experimental nucleic acid chemistry should first try to learn the basics from an expert. At the time the first article was written, ours was the only laboratory involved in experimental structural DNA nanotechnology. Today, this is easier, because there are more than forty laboratories worldwide engaged in this enterprise.

Thus, experimental knowledge is accumulating at an astonishingly fast rate. It is impossible to detail all the technical lessons gained in those laboratories, or even to be aware of them all. The growth of this field in combination with, and stimulated by, DNA-based computation has been remarkably satisfying. However, the overall effort has also become so unwieldy that no individual can be aware of all the technical details now known that enable us to perform successful experiments in that area. Furthermore, while trying to remain general, I have discussed only those features that apply to traditional "vanilla" DNA or RNA. In addition to vanilla DNA, there are hundreds of derivatives, each of which is likely to evince a series of idiosyncrasies. This is a huge field, and the excitement is intense.

Acknowledgements I wish to thank all of my colleagues, both within and beyond our laboratory, who have aided in working out the principles of structural DNA nanotechnology. This research has been supported by grants GM-29554 from NIGMS, CCF-0523290, CTS-0608889, CCF-0726378 from the NSF, 48681-EL and MURI W911NF-07-1-0439 from ARO, DE-FG02-06ER64281 from DOE (Subcontract from the Research Foundation of SUNY), and a grant from the W.M. Keck Foundation.

References

1. Adleman LM (1994) Molecular computation of solutions to combinatorial problems. Science 266:1021–1024
2. Seeman NC (1982) Nucleic acid junctions and lattices. J Theor Biol 99:237–247
3. Seeman NC, Lukeman PS (2005) Nucleic acid nanostructures. Rep Prog Phys 68:237–270
4. Fu T-J, Seeman NC (1993) DNA double crossover structures. Biochemistry 32:3211–3220
5. He Y, Tian Y, Chen Y, Deng ZX, Ribbe AE, Mao CD (2005) Sequence symmetry as a tool for designing DNA nanostructures. Angew Chem Int Ed 44:6694–6696

6. Lu Y, Ke YG, Yan H (2005) Self-assembly of symmetric finite-size DNA nanoarrays. J Am Chem Soc 127:17140–17141
7. Rothemund PWK (2006) Folding DNA to create nanoscale shapes and patterns. Nature 440:297–302
8. Voet D, Rich A (1970) The crystal structures of purines, pyrimidines and their intermolecular complexes. Prog Nucl Acid Res Mol Biol 10:183–265
9. Fahlman RP, Sen D (1999) Synapsable DNA double helices: self-selective modules for assembling DNA superstructures. J Am Chem Soc 121:11079–11085
10. Wang WX, Liu HJ, Liu DS, Xu YR, Yang Y, Zhou DJ (2007) The use of the interparticle i-motif for the controlled assembly of gold nanoparticles. Langmuir 23:11956–11959
11. Chworos A, Jaeger L (2004) Building programmable jigsaw puzzles with RNA. Science 306:2068–2072
12. Paukstelis P, Nowakowski J, Birktoft JJ, Seeman NC (2004) The crystal structure of a continuous three-dimensional DNA lattice. Chem Biol 11:1119–1126
13. Sherman WB (2007) Disentangling kinetics and energetics in DNA nanostructure assembly: forming parallel double crossover molecules. In: Proceedings of the fourth conference on foundations of nanoscience, Snowbird, Utah, April 18–21, 2007. Science Technica Inc, pp 65–65
14. Birac JJ, Sherman WB, Kopatsch J, Constantinou PE, Seeman NC (2006) GIDEON, A program for design in structural DNA nanotechnology. J Mol Graph Model 25:470–480
15. Sherman WB, Seeman NC (2006) Design of low-stress nucleic acid nanotubes. Biophys J 90:4546–4557
16. Seeman NC (1985) The interactive manipulation and design of macromolecular architecture utilizing nucleic acid junctions. J Mol Graph 3:34–39
17. Fu T-J, Tse-Dinh Y-C, Seeman NC (1994) Holliday junction crossover topology. J Mol Biol 236:91–105
18. Park SH, Barish R, Li HY, Reif JH, Finkelstein G, Yan H, LaBean TH (2005) Three-helix bundle DNA tiles assemble into 2D lattice or 1D templates for silver nanowires. Nano Lett 5:693–696
19. Liu D, Wang MS, Deng ZX, Walulu R, Mao CD (2004) Tensegrity: construction of rigid DNA triangles with flexible four-arm DNA junctions. J Am Chem Soc 126:2324–2325
20. Churchill MEA, Tullius TD, Kallenbach NR, Seeman NC (1988) A Holliday recombination intermediate is twofold symmetric. Proc Natl Acad Sci (USA) 85:4653–4656
21. Kim SH, Suddath FL, Quigley GJ, McPherson A, Sussman JL, Wang AH-J, Seeman NC, Rich A (1974) The three dimensional tertiary structure of transfer RNA. Science 185:435–440
22. Hsieh P, Panyutin IG (1995) DNA branch migration. In: Eckstein F, Lilley DMJ (eds) Nucleic acids and molecular biology, vol 9. Springer, Berlin, pp 42–65
23. Yan H, Zhang X, Shen Z, Seeman NC (2002) A robust DNA mechanical device controlled by hybridization topology. Nature 415:62–65
24. Zhang DY, Turberfield AJ, Yurke B, Winfree E (2007) Engineering entropy-driven reactions and networks catalyzed by DNA. Science 318:1121–1125
25. Yin P, Choi HMT, Calvert CR, Pierce NA (2008) Programming biomolecular self-assembly pathways. Nature 451:318–322
26. Yurke B, Turberfield AJ, Mills AP Jr, Simmel FC, Newmann JL (2000) A DNA-fuelled molecular machine made of DNA. Nature 406:605–608
27. Depew RE, Wang J (1975) Conformational fluctuations of DNA helix. Proc Natl Acad Sci (USA) 72:4275–4279
28. Wu G, Jonoska N, Seeman NC (2008) Self-assembly of a DNA nano-object demonstrates natural computation (submitted for publication)
29. Wang H, Di Gate RJ, Seeman NC (1998) The construction of an RNA knot and its role in demonstrating that *E. coli* DNA topoisomerase III is an RNA topoisomerase. In: Sarma RH, Sarma MH (eds) Structure, motion, interaction and expression of biological macromolecules. Adenine Press, New York, pp 103–116
30. Shen Z, Yan H, Wang T, Seeman NC (2004) Paranemic crossover DNA: a generalized holliday structure with applications in nanotechnology. J Am Chem Soc 126:1666–1674

31. Du SM, Zhang S, Seeman NC (1992) DNA junctions, antijunctions and mesojunctions. Biochemistry 31:10955–10963
32. Sambrook J, Fritsch EF, Maniatis T (1989) Molecular Cloning, 2nd edn. Cold Spring Harbor Laboratory Press, Cold Spring Harbor
33. Constantinou PE, Wang T, Kopatsch J, Israel LB, Zhang X, Ding B, Sherman WB, Wang X, Zheng J, Sha R, Seeman NC (2006) Double cohesion in structural DNA nanotechnology. Org Biomol Chem 4:3414–3419
34. Shih WM, Quispe JD, Joyce GF (2004) DNA that folds into a nanoscale octahedron. Nature 427:618–621

Algorithmic Control: The Assembly and Operation of DNA Nanostructures and Molecular Machinery

Andrew J. Turberfield

Abstract It gives me great pleasure to contribute to this celebration of Grzegorz Rozenberg's contribution to the field of natural computing. I am grateful to Grzegorz for fostering this remarkably interdisciplinary community which has provided me with so much interest and enjoyment.

The theme of this symposium is 'algorithmic bioprocesses': this paper is concerned with the creation of artificial structures by algorithmic assembly of a biomolecule, DNA. I will survey different strategies for encoding assembly and operation algorithms in the design of DNA nanostructures, using examples that my colleagues and I have worked on.

1 Algorithmic Assembly

An algorithm is a process or set of rules [1]. DNA self-assembly is dependent on the Watson–Crick base pairing rule [2]: two single strands of DNA will bind antiparallel to each other to form a double helix if their nucleotide sequences are complementary (A binds to T, C to G). This rule is supplemented by rules that prescribe how to design more complicated structural motifs such as branched junctions, single-stranded loops, G-quadruplexes, i-motifs, etc. [3] This set of structural rules allows the product of assembly of an interacting set of DNA strands to be controlled through design of their nucleotide sequences [4].

Strand design is algorithmic: it usually begins with decisions on how to position strands in the final product and ends with automated selection of nucleotide sequences to satisfy the resulting complementarity constraints and to reduce the strength of competing interactions [5–7]. The process of assembly itself is also algorithmic: the design of the component molecules embodies the assembly program and controls the physical interactions that determine the product.

Figure 1 shows how a DNA tetrahedron can be assembled from four strands of DNA [8]. Each strand is designed to run around one face, hybridizing to each of the three other strands to form three double-helical edges. The edges consist of 20 base pairs (two helical turns) and are approximately 7 nm in length. This is much shorter than the persistence length of DNA (50 nm [9]), so to a good approximation

A.J. Turberfield (✉)
Department of Physics, University of Oxford, Parks Road, Oxford OX1 3PU, UK
e-mail: a.turberfield1@physics.ox.ac.uk

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_13, © Springer-Verlag Berlin Heidelberg 2009

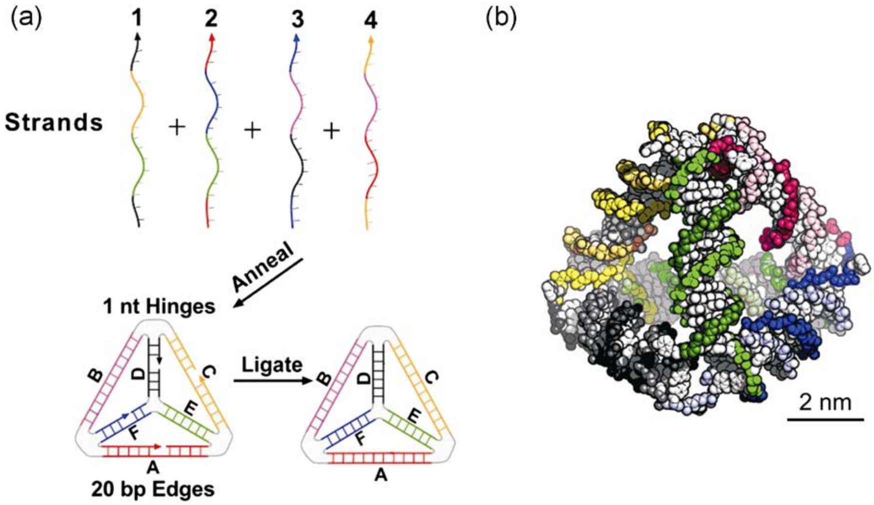


Fig. 1 Algorithmic assembly of a DNA tetrahedron [8]. (**a**) Design of the four component strands of DNA, whose nucleotide sequences are designed such that the tetrahedron is the stable product of self-assembly. Complementary subsequences that hybridize to form each edge are identified by colour. (**b**) Space filling representation of a tetrahedron with 20-base pair edges

the edges can be considered to be stiff and straight. The tetrahedron, constructed of rigid triangles, has a well-defined three-dimensional structure. The information contained in the design of the four strands—i.e., the regions of complementarity indicated by colour in Fig. 1, and the designed weakness of all other interactions—is sufficient to define this structure. To assemble tetrahedra in high yield, all that is necessary is to mix the four strands in hybridization buffer at high temperature and cool through the range of temperatures over which assembly occurs.

The products of DNA self-assembly often look like organic molecules scaled up by one or two orders of magnitude in linear dimension, with double helices for bonds and branch junctions for atoms. DNA self-assembly differs fundamentally from organic synthesis, however: the enormous number of distinct nucleotide sequences (4^{10} for each 10-basepair helix period) means that each junction (atom) and each helix (bond) can be unique (i.e., the corresponding sequences can be designed to be orthogonal), and the stabilities of off-target assemblies can be greatly reduced. In the simplest case, the designed product corresponds to a deep free energy minimum and its formation is not dependent on the details of the assembly process. DNA self-assembly usually needs no catalyst to favour a particular pathway and can often occur in a single reaction. Examples are Shih and co-workers' octahedron [10], Rothemund's origami [11], and the DNA tetrahedron shown in Fig. 1 [8].

In some experiments, assembly takes place in a carefully controlled sequence of steps, even though the product is designed to be very stable. Examples include the first DNA polyhedra, made by the Seeman group [12, 13]. Stepwise synthesis is necessary to achieve a high yield of a single product only when the loss in enthalpy resulting from deviations from the design is insufficient to compensate for the corresponding increase in entropy (for example, if off-target multimeric products could satisfy most base-pairing interactions), or when the time required to achieve equilibrium becomes impractically long. When it is necessary to adopt sequential synthesis rather than a one-pot reaction, and when it is simply more convenient, has not been thoroughly explored.

A strategy often used to create extended DNA structures, particularly two-dimensional arrays, is to build an assembly hierarchy into the strand designs. Arrays are typically built from DNA tiles, which are robust building blocks consisting of a small number of strands held together by hybridization. The most popular type is the double-crossover tile [14, 15] consisting of two double helices held together by strand exchange at two points; several other designs have been demonstrated [16–21]. Interactions between tiles are by hybridization of single-stranded 'sticky ends' (typically four to a tile). The structure of the array is determined by the tile geometry and the pattern of complementarity between the sticky ends. Extended arrays are usually made by mixing components at high temperature and cooling slowly through the temperature range in which assembly occurs (typically 80∼20°C) [15]. The interactions between the strands that make up a tile are, by design, significantly stronger than the interactions between sticky ends that bind the tiles together. When the system is heated above the melting temperature of all complexes, then cooled, the sequence of assembly is controlled by this hierarchy of interaction strengths: isolated tiles form first then, at lower temperatures, tiles assemble by sticky-end cohesion. Arrays formed in this way are typically indistinguishable from those formed by annealing pre-formed tiles [15]. In either case, the assembly algorithm involves two steps: tile formation, followed by assembly into an extended array.

Figure 2 shows two examples of arrays formed in this way [19]. The tile has the same DNA components in each case: it is formed by hybridization of four short strands to create a four-arm junction with six-nucleotide sticky ends. In a hybridization buffer containing magnesium ions, this motif folds into a compact configuration in which two pairs of arms stack coaxially to form two double helices that are joined by exchange of two strands where they cross [22, 23]. Hybridization of the sticky ends joins these tiles to create a woven structure resembling kagome basketwork [24]. The same tiles can be reprogrammed to form the square lattice structure shown in Fig. 2 by adding a protein (RuvA) that binds the junctions and unfolds them into a square planar configuration [25].

The arrays shown in Fig. 2 are periodic. Information-rich, aperiodic arrays can be created by a process that explicitly maps tile assembly onto the (algorithmic) operation of a 1D cellular automaton [26]: successive rows of tiles correspond to successive states of the automaton. The initial row of tiles corresponds to the automaton's program, and the pattern of sticky ends on the set of assembling tiles embodies its rule table. This strategy has been used to assemble complex tile patterns including a cumulative XOR on the input [27] which can generate a fractal pattern (a Sierpinski triangle) [28], and a binary counter [29]. This form of assembly takes place close to, but not quite at equilibrium: array growth must take place in a defined direction—outward from the seed—corresponding to the operating sequence of the automaton. The initial row of tiles acts as a crystallization nucleus to seed array growth. It is important that assembly is not too far from equilibrium, however, to maintain effective

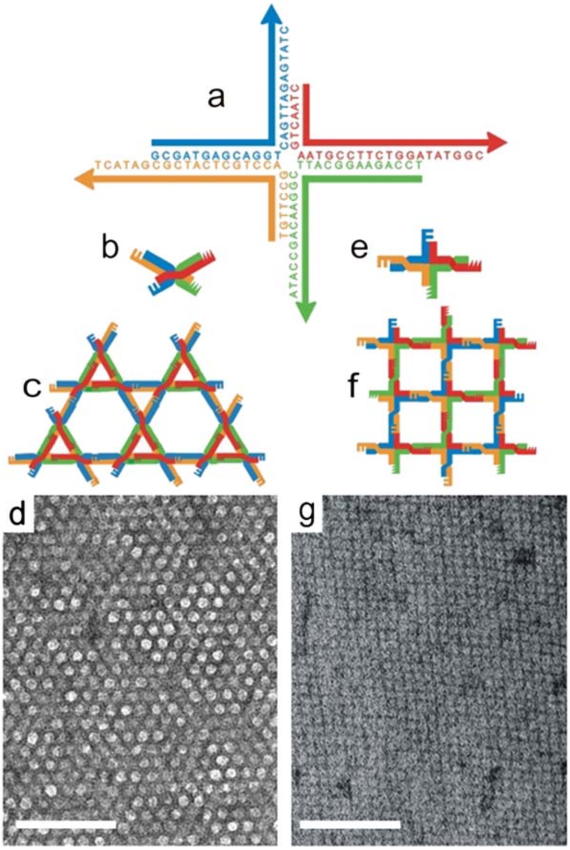


Fig. 2 Reprogrammable two-dimensional DNA array [19]. (**a**) The common structural unit—four oligonucleotides hybridize to form a four-arm tile with two pairs of complementary sticky ends. (**b**) Tertiary structure of the tile in the presence of Mg^{2+}: sticky ends are represented by lock and key symbols. (**c**) Kagome lattice formed by assembly of tiles in the presence of Mg^{2+} (for clarity, half a helical turn is shown between junctions that are, in fact, separated by 2.5 turns). The structure of the array is programmed by the spatial distribution of sticky ends on a tile. (**d**) Transmission electron micrograph of the kagome lattice (DNA is positively stained (*dark*); scale bar: 100 nm). (**e**) Square-planar tile produced by binding of a tetramer of protein RuvA to the four-arm junction. In this configuration, the ordering of sticky ends around the periphery of the tile is different. (**f**) Square lattice formed from tiles held in a square-planar configuration. By changing the tertiary structure of the tile and repositioning the sticky ends, the protein has reprogrammed array formation. (**g**) Transmission electron micrograph of the RuvA lattice (negatively stained: protein is lighter than background; scale bar: 100 nm)

discrimination between an incoming tile that correctly forms bonds to two tiles in the preceding row and one that forms only one bond. Error-correcting strategies that (at the cost of requiring a larger tile set) ensure that incorporation of one imperfectly bonded tile always leads to incorporation of another [30], doubling the free energy penalty, are only effective if the tile off-rate is sufficiently high to allow removal of both incorrect tiles before the front of the growing array has propagated so far that the error is locked in.

Algorithmic control of assembly can also be achieved in reactions that are far from equilibrium by designing secondary structure to control reaction rates. A further set of sequence design rules is being developed that ensures that the target structure is the one reached most rapidly. Kinetically controlled assembly, in general, avoids equilibrium—the assembly product corresponds to a local, not a global, free energy minimum. The idea that DNA hybridization rates could be controlled by a specific DNA signal was introduced by Turberfield and co-workers [31], and has been developed by groups at Caltech, Lucent Technologies, and Oxford [32–34]. The fundamental idea is that secondary structure, including loop structures, can be used to inhibit hybridization of complementary DNA strands. Hybridization of nucleotides in single-stranded loop domains can provide the free energy to drive a reaction, but is inhibited topologically (unlinked loops of DNA cannot be wound round each other to form a double helix without building in compensating counterturns) and energetically (a double helix forming a loop that is much shorter than the persistence length is highly strained). Hybridization can be facilitated by a strand invasion reaction with a third strand that disrupts secondary structure and opens a loop. If the subsequent reaction of the opened loop with its complement displaces the loop-opening strand then this strand is a catalyst that can initiate multiple reactions [31]. A reaction rate enhancement of three orders of magnitude has been achieved by a hybridization catalyst [33]. In other implementations, an initial loop-opening strand initiates a cascade of further loop-opening reactions to create a programmed polymeric or dendritic product [35, 36]. Control of non-equilibrium hybridization reactions can also be used to create logic gates whose inputs and outputs are DNA strands [37, 38]. It is intriguing to speculate that error rates in algorithmic tile assembly could be reduced by running the reaction further from equilibrium and using kinetic control to make the rapid incorporation of an incoming tile conditional on the correct placement of preceding tiles [31].

2 Algorithmic Control of Molecular Machinery

DNA hybridization can be used as a source of energy to drive changes in the conformation of a DNA nanostructure. This concept was introduced by the demonstration of DNA ‘tweezers’ [39] that could be opened and closed by adding components of a DNA fuel. DNA hybridization and strand displacement reactions have been used to drive rotary devices [40], to open a cage [41, 42], to control a chemical reaction [43–46] and to create ‘walkers’ that step along a track [47, 48]. Other nanostructure actuation strategies include conformational changes induced by changes in the buffer [49–51]. Each of these devices operates under algorithmic control and, for each, part of the control loop is outside the self-assembled biomolecular system. Usually control is exerted manually by a researcher who controls experimental parameters such as strand concentrations or pH, though control has been delegated to independent chemical reactions [52, 53]. The DNA walkers are good examples of the advantages and limitations of external control: each step requires addition of two signal strands of DNA, first to free a foot from an anchorage on the track by a strand displacement reaction, then to create a link to the next anchorage; the device is very tightly controlled but incapable of autonomous motion.

To create a DNA device (such as a molecular motor) whose control algorithm is encoded entirely in DNA, such that it can operate autonomously, is an experimental challenge [54]. A molecular motor requires an energy source: if the motor itself (or its track) is to avoid irreversible degradation then it must act as a catalyst for the reaction of an external fuel. Hybridization catalysis [31] was developed in order to enable the use of DNA hybridization as an energy source for autonomous molecular machinery: formation of ten base pairs provides approximately the same free energy as hydrolysis of a molecule of ATP under typical cellular conditions [55]. Other energy sources that have been explored are hydrolysis of adenosine triphosphate (ATP) [56] and of the DNA (or RNA) backbone [57, 58].

Unidirectional motion powered by DNA hydrolysis has been achieved by progressively destroying the track behind the motor (a 'burnt bridges' mechanism). The motor (cargo) consists of a single strand that can hybridize to any one of an array of single-stranded anchorages bound to a rigid track. In one implementation, the motor incorporates a DNA domain that directly catalyzes hydrolysis of a ribonucleotide incorporated in the anchorage [58]; in another, hybridization of motor to anchorage enables an auxiliary restriction enzyme to cut the anchorage [57]. In both cases, the motor migrates from the damaged anchorage to the next, intact, anchorage by a branch migration reaction to initiate another cycle of cleavage and motion. Undirected motion through a three-dimensional matrix of anchorages has also been demonstrated [59].

Figure 3 illustrates the operation of the enzyme-assisted burnt bridges motor [57]. The motor-anchorage duplex contains a non-palindromic recognition site for a restriction enzyme which has been modified to cut only one of the strands—the anchorage [60]. The enzyme cannot cut a single-stranded anchorage, so both the motor and the enzyme have catalytic functions. The fragment of the anchorage released by the enzyme is short enough to melt and diffuse away: the motor then transfers to the next anchorage by a process of branch migration [61] which is initiated by hybridization of the single-stranded domain, at the top of the motor, revealed by the loss of the cut fragment [62].

A DNA hybridization catalyst that is localized at the growing end of a polymer created by reactions between hairpin loops [63] can be considered as a motor, analogous to the bacterium *Listeria* which moves by catalyzing the polymerization of actin: this directional motion is similar to that of the 'burnt bridges' motors in that it causes an irreversible change (in this case, irreversible creation) of a track.

A motor that does not destroy its track has been created by using repeated enzymatic ligation (joining) and restriction (cutting) to move a DNA fragment from anchorage to anchorage [56]. Ligation of the fragment to the end of an anchorage enables further ligation to form a covalently bonded bridge with the next anchorage: this creates a recognition site for a restriction enzyme that cuts the duplex in such a way that the fragment is transferred between anchorages. By using two restriction

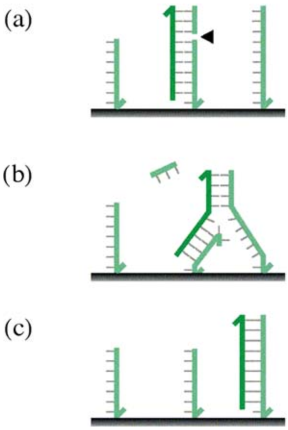


Fig. 3 Algorithmic operation of a linear motor powered by a nicking enzyme [57]. The track is an array of periodically spaced single-stranded anchorages. The motor (*dark green*) is an oligonucleotide that can bind to any one of the anchorages. (**a**) The motor is bound to anchorage S_i, enabling enzyme N.BbvC IB to cut the anchorage to release a short fragment. The motor is left with a single-stranded overhang which is free to bind to the adjacent anchorage S_{i+1}. The motor cannot bind to S_{i-1} which was cut in the previous step. (**b**) The motor then steps onto S_{i+1} by a simple branch-migration reaction. For each base pair broken between motor and S_i, a base pair can be formed between motor and S_{i+1}. (**c**) Upon completion of the branch migration reaction, the motor is bound to S_{i+1} and the step is complete

enzymes and a repeated set of four anchorages, it is possible for this device to operate unidirectionally and indefinitely [64]: its energy source is hydrolysis of ATP, coupled to DNA ligation by the ligase (an ATPase).

A design for a DNA motor that couples catalysis of the reaction of a pair of complementary hairpin loops to a step along a reusable track [54] is shown in Fig. 4. As with the burnt bridges motor shown in Fig. 3, the track consists of an array of single-stranded anchorages, to any of which the motor can hybridize. The reaction between a complementary pair of hairpin loops is catalyzed by the motor-anchorage complex and by a neighbouring empty anchorage: these structures react with complementary loops to create intermediate complexes in which the loops are open. The opened loops can then react rapidly with each other to form a stable duplex waste product. During each loop-loop reaction the motor is lifted from one anchorage and deposited on the next. This motor is processive: the motor remains securely attached to the track. It is not intrinsically directional, but directional motion could be achieved by adopting a burnt bridges mechanism: if one of the hairpin species were removed after forming complexes with each empty anchorage, then ‘used’ anchorages would remain empty, and the motor could only move forward. (The track could be regenerated by resupplying fuel.) Work on molecular motors that use biomimetic design principles to coordinate catalysis of the reaction of the fuel with mechanical motion is in progress [65, 66].

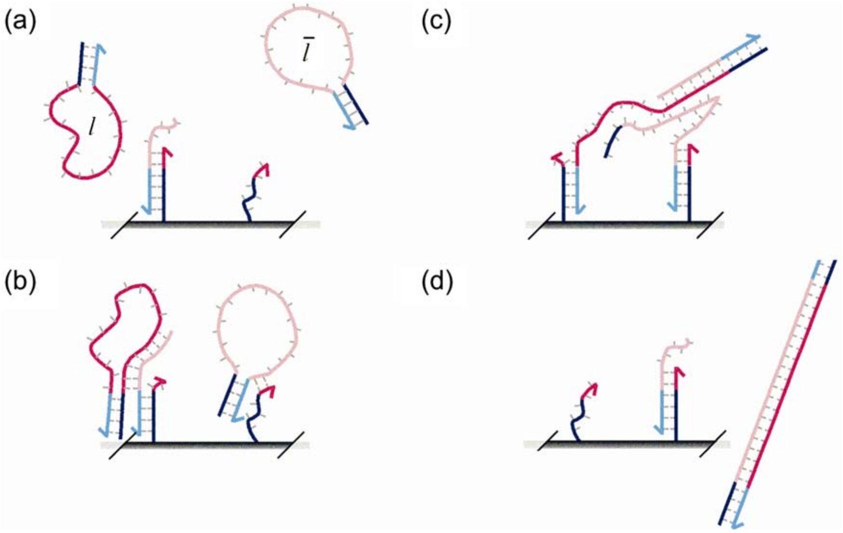


Fig. 4 Algorithmic operation of a hybridization-powered molecular motor [54]. (**a**) The motor is shown bound to one of an array of single-stranded anchorages attached to a rigid track. Movement of the motor is coupled to hybridization of complementary hairpin loops l and $\bar{l}$. (**b**) An empty anchorage is designed to bind to the first few bases of the loop domain of the $\bar{l}$ hairpin (*pink*) and to open the hairpin's neck (*blue*) by a strand-displacement reaction. The motor is complementary to the anchorage sequence and has an extended toehold region designed to interact with the loop domain of the complementary hairpin l. The track-bound motor can open a l hairpin in a strand-exchange reaction in which the blue domains in the hairpin neck, the motor and the anchorage exchange partners, displacing the motor from its initial anchorage but keeping it securely attached to the track. (**c**) By forcing open the necks of complementary hairpins l and $\bar{l}$ the track-bound motor and adjacent empty anchorage catalyze their hybridization. (**d**) When the open loops hybridize to produce an l–$\bar{l}$ duplex (a waste product), the motor is deposited on the next anchorage by strand exchange. One loop–loop reaction is thus coupled to one step down the track. This motor could be made directional by preparing all anchorages with bound (open) $\bar{l}$ hairpins then removing excess $\bar{l}$: motion of the motor would leave empty anchorages behind, preventing backward steps

3 Summary

DNA self-assembly is an algorithmic process. The algorithm that controls assembly is embodied in the nucleotide sequences of the component strands of DNA. Different control strategies are possible: assembly can occur in a single reaction or in a controlled sequence of reactions; strands may be designed to ensure that at each stage the desired product is the most stable or the most rapidly formed. Active devices can be controlled by externally supplied signals or by embedding a control algorithm in the design of interacting components and DNA fuels.

Acknowledgements I am grateful to Jonathan Bath for help with the preparation of this manuscript. I wish to acknowledge my colleagues and students at Oxford, and collaborations with Lucent Technologies, Duke, and Osaka Universities, Caltech and the Vrije Universiteit Amsterdam.

References

1. The Oxford English Dictionary (1989) 2nd edn. Oxford University Press
2. Watson JD, Crick FHC (1953) A structure for deoxyribose nucleic acid. Nature 171:737–738
3. Gilbert DE, Feigon J (1999) Multistranded DNA structures. Curr Opin Struct Biol 9:305–314
4. Seeman NC (2003) DNA in a material world. Nature 421:427–431
5. Seeman NC (1990) De novo design of sequences for nucleic-acid structural engineering. J Biomol Struc Dyn 8:573–581
6. Dirks RM, Lin M, Winfree E, Pierce NA (2004) Paradigms for computational nucleic acid design. Nucleic Acids Res 32:1392–1403
7. Goodman RP (2005) NANEV: a program employing evolutionary methods for the design of nucleic acid nanostructures. Biotechniques 38:548–550
8. Goodman RP, Schaap IAT, Tardin CF, Erben CM, Berry RM, Schmidt CF, Turberfield AJ (2005) Rapid chiral assembly of rigid DNA building blocks for molecular nanofabrication. Science 310:1661–1665
9. Smith SB, Finzi L, Bustamante C (1992) Direct mechanical measurements of the elasticity of single DNA molecules by using magnetic beads. Science 258:1122–1126
10. Shih WM, Quispe JD, Joyce GF (2004) A 1.7-kilobase single-stranded DNA that folds into a nanoscale octahedron. Nature 427:618–621
11. Rothemund PWK (2006) Folding DNA to create nanoscale shapes and patterns. Nature 440:298–302
12. Chen JH, Seeman NC (1991) Synthesis from DNA of a molecule with the connectivity of a cube. Nature 350:631–633
13. Zhang YW, Seeman NC (1994) Construction of a DNA-truncated octahedron. J Am Chem Soc 116:1661–1669
14. Fu TJ, Seeman NC (1993) DNA double-crossover molecules. Biochemistry 32:3211–3220
15. Winfree E, Liu FR, Wenzler LA, Seeman NC (1998) Design and self-assembly of two-dimensional DNA crystals. Nature 394:539–544
16. Mao CD, Sun WQ, Seeman NC (1999) Designed two-dimensional DNA Holliday junction arrays visualized by atomic force microscopy. J Am Chem Soc 121:5437–5443
17. LaBean TH, Yan H, Kopatsch J, Liu FR, Winfree E, Reif JH, Seeman NC (2000) Construction, analysis, ligation, and self-assembly of DNA triple crossover complexes. J Am Chem Soc 122:1848–1860
18. Yan H, Park SH, Finkelstein G, Reif JH, LaBean TH (2003) DNA-templated self-assembly of protein arrays and highly conductive nanowires. Science 301:1882–1884
19. Malo J, Mitchell JC, Venien-Bryan C, Harris JR, Wille H, Sherratt DJ, Turberfield AJ (2005) Engineering a 2D protein-DNA crystal. Angew Chem Int Ed 44:3057–3061
20. He Y, Chen Y, Liu HP, Ribbe AE, Mao CD (2005) Self-assembly of hexagonal DNA two-dimensional (2D) arrays. J Am Chem Soc 127:12202–12203
21. He Y, Tian Y, Ribbe AE, Mao CD (2006) Highly connected two-dimensional crystals of DNA six-point-stars. J Am Chem Soc 128:15978–15979
22. Ortiz-Lombardia M, Gonzalez A, Eritja R, Aymami J, Azorin F, Coll M (1999) Crystal structure of a DNA Holliday junction. Nat Struct Biol 6:913–917
23. Eichman BF, Vargason JM, Mooers BH, Ho PS (2000) The Holliday junction in an inverted repeat DNA sequence: sequence effects on the structure of four-way junctions. Proc Natl Acad Sci USA 97:3971–3976
24. Syôzi I (1951) Statistics of Kagomé lattice. Prog Theor Phys 6:306–308
25. Zerbib D, Mezard C, George H, West SC (1998) Coordinated actions of RuvABC in holliday junction processing. J Mol Biol 281:621–630
26. Winfree E (1996) On the computational power of DNA annealing and ligation. In: Lipton RJ, Baum EB (eds) DNA based computers, vol 27. American Mathematical Society, Providence, pp 199–221
27. Mao C, LaBean TH, Reif JH, Seeman NC (2000) Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. Nature 407:493–496

28. Rothemund PWK, Papadakis N, Winfree E (2004) Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biol 2:2041–2053
29. Barish RD, Rothemund PWK, Winfree E (2005) Two computational primitives for algorithmic self-assembly: copying and counting. Nano Lett 12:2586–2592
30. Winfree E, Bekbolatov R (2004) Proofreading tile sets: error correction for algorithmic self-assembly. DNA Comput 2943:126–144
31. Turberfield AJ, Mitchell JC, Yurke B, Mills AP, Blakey MI, Simmel FC (2003) DNA fuel for free-running nanomachines. Phys Rev Lett 90:118102
32. Bois JS, Venkataraman S, Choi HM, Spakowitz AJ, Wang ZG, Pierce NA (2005) Topological constraints in nucleic acid hybridization kinetics. Nucleic Acids Res 33:4090–4095
33. Seelig G, Yurke B, Winfree E (2006) Catalysed relaxation of a metastable fuel. J Am Chem Soc 128:12211–12220
34. Green SJ, Lubrich D, Turberfield AJ (2006) DNA hairpins: fuel for autonomous DNA devices. Biophys J 91:2966–2975
35. Dirks RM, Pierce NA (2004) Triggered amplification by hybridization chain reaction. Proc Natl Acad Sci USA 101:15275–15278
36. Yin P, Choi HMT, Calvert CR, Pierce NA (2008) Programming biomolecular self-assembly pathways. Nature 451:318–323
37. Seelig G, Soloveichik D, Zhang DY, Winfree E (2006) Enzyme-free nucleic acid logic circuits. Science 314:1585–1588
38. Zhang DY, Turberfield AJ, Yurke B, Winfree E (2007) Engineering entropy-driven reactions and networks catalyzed by DNA. Science 318:1121–1125
39. Yurke B, Turberfield AJ, Mills AP, Simmel FC, Neumann JL (2000) A DNA-fuelled molecular machine made of DNA. Nature 406:605–608
40. Yan H, Zhang X, Shen Z, Seeman NC (2002) A robust DNA mechanical device controlled by hybridization topology. Nature 415:62–65
41. Goodman RP, Heilemann M, Doose S, Erben CM, Kapanidis AN, Turberfield AJ (2008) Reconfigurable, braced, three-dimensional DNA nanostructures. Nat Nanotechnol 3:93–96
42. Erben CM, Goodman RP, Turberfield AJ (2006) Single-molecule protein encapsulation in a rigid DNA cage. Angew Chem Int Ed 45:7414–7417
43. Liao S, Seeman NC (2004) Translation of DNA signals into polymer assembly instructions. Science 306:2072–2074
44. Chen Y, Mao C (2004) Reprogramming DNA-directed reactions on the basis of a DNA conformational change. J Am Chem Soc 126:13240–13241
45. Snyder TM, Liu DR (2005) Ordered multistep synthesis in a single solution directed by DNA templates. Angew Chem Int Ed 44:7379–7382
46. Chhabra R, Sharma J, Liu Y, Yan H (2006) Addressable molecular tweezers for DNA-templated coupling reactions. Nano Lett 6:978–983
47. Sherman WB, Seeman NC (2004) A precisely controlled DNA biped walking device. Nano Lett 4:1203–1207
48. Shin J-S, Pierce NA (2004) A synthetic DNA walker for molecular transport. J Am Chem Soc 126:10834–10835
49. Mao C, Sun W, Shen Z, Seeman NC (1999) A nanomechanical device based on the B–Z transition of DNA. Nature 397:144–146
50. Liu D, Balasubramanian S (2003) A proton-fuelled DNA nanomachine. Angew Chem Int Ed 42:5734–5736
51. Alberti P, Mergny J-L (2003) DNA duplex–quadruplex exchange as the basis for a nanomolecular machine. Proc Natl Acad Sci USA 100:1569–1573
52. Liedl T, Simmel FC (2005) Switching the conformation of a DNA molecule with a chemical oscillator. Nano Lett 5:1894–1898
53. Dittmer WU, Simmel FC (2004) Transcriptional control of DNA-based nanomachines. Nano Lett 4:689–691
54. Bath J, Turberfield AJ (2007) DNA nanomachines. Nat Nanotechnol 2:275–284
55. SantaLucia J (1998) A unified view of polymer, dumbell, and oligonucleotide nearest neighbour thermodynamics. Proc Natl Acad Sci USA 95:1460–1465

56. Yin P, Yan H, Daniell XG, Turberfield AJ, Reif JH (2004) A unidirectional DNA walker that moves autonomously along a DNA track. Angew Chem Int Ed 43:4906–4911
57. Bath J, Green SJ, Turberfield AJ (2005) A free-running DNA motor powered by a nicking enzyme. Angew Chem Int Ed 44:4358–4361
58. Tian Y, He Y, Peng Y, Mao C (2005) A DNA enzyme that walks processively and autonomously along a one-dimensional track. Angew Chem Int Ed 44:4355–4358
59. Pei R, Taylor SK, Stefanovic D, Rudchenko S, Mitchell TE, Stojanovic MN (2006) Behavior of polycatalytic assemblies in a substrate-displaying matrix. J Am Chem Soc 128:12693–12699
60. Heiter DF, Lunnen KD, Wilson GG (2005) Site-specific DNA-nicking mutants of the heterodimeric restriction endonuclease R. BbvCI J Mol Biol 348:631–640
61. Lee CS, Davis RW, Davidson N (1970) A physical study by electron microscopy of the terminally repetitious, circularly permuted DNA from the coliphage particles of Escherichia coli 15. J Mol Biol 48:1–8
62. Yurke B, Mills AP (2003) Using DNA to power nanostructures. Genet Program Evol Mach 4:111–122
63. Venkataraman S, Dirks RM, Rothemund PWK, Winfree E, Pierce NA (2007) An autonomous polymerization motor powered by DNA hybridization. Nat Nanotechnol 2:490–494
64. Yin P, Turberfield AJ, Reif JH (2005) Designs of autonomous unidirectional walking DNA devices. DNA Comput 3384:410–425
65. Green SJ, Bath J, Turberfield AJ (2008) Coordinated chemomechanical cycles: a mechanism for autonomous molecular motion. Phys Rev Lett 101:238101
66. Bath J, Green SJ, Allen KE, Turberfield AJ (2009) Mechanism for a directional, processive, and reversible DNA motor. Small (in press)

Part V Membrane Computing

On Nonuniversal Symport/Antiport P Systems

Oscar H. Ibarra and Sara Woodworth

Abstract We examine restricted SA P system models and analyze minimal systems with regard to the size of the alphabet and the number of membranes. We study the precise power of SA P systems with either 1, 2, or 3 symbols and less than 5, 4, and 3 membranes, respectively, improving the previous results. The question of whether using only a single symbol with any number of membranes is universal remains open.

We define and examine restricted forms of SA P systems (called *bounded SA P systems* and *special SA P systems*) finding infinite hierarchies with respect to the both the size of the alphabet and the number of membranes. We also analyze the role of determinism versus nondeterminism and find that over a unary input alphabet, these systems are equivalent if and only if deterministic and nondeterministic linear-bounded automata (over an arbitrary input alphabet) are equivalent.

Finally, we introduce restricted SA P system models which characterize semilinear sets. We also show "slight" extensions of the models allow them to accept (respectively, generate) nonsemilinear sets. In fact, for these extensions, the emptiness problem is undecidable.

1 Introduction

A very simple membrane computing model known as *Symport/Antiport P system* (SA P system) was introduced by Gheorghe and Andrei Păun in [18] and quickly rose in popularity. The model is *purely communicative* and based on the biochemical idea that certain pairs of chemicals allow transport between membranes. In SA P systems, rules only allow the transport of objects between membranes. No objects are created, no objects are deleted, and the membrane structure is fixed. However, the model is quite powerful and is known to be computationally complete.

The SA P system model is defined formally as $\Pi = \langle V, \mu, w_1, \ldots, w_m, w_e, R_1, \ldots, R_m, i_o \rangle$ where V is the alphabet of objects allowed within the system. The environment contains an unlimited number of each object which can be brought into the system during the computation. μ is the initial membrane hierarchy with m membranes where each membrane is given a distinct label. w_i is the initial multiset of objects initially located in membrane i. w_e is the initial multiset of objects initially

This work was supported in part by NSF Grants CCF-0430945 and CCF-052416.

O.H. Ibarra (✉)
Department of Computer Science, University of California, Santa Barbara, CA 93106, USA
e-mail: ibarra@cs.ucsb.edu

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_14, © Springer-Verlag Berlin Heidelberg 2009

located in the environment. If an object occurs once in w_e, it must occur an infinite number of times. This means that the environment has its own alphabet $E \subseteq V$ such that each object in E occurs in unlimited quantities in the environment. R_i is a set of rules for membrane i of the form (u, out), (u, in), or $(u, out; v, in)$ where $u, v \in V^*$.

The rule types (u, out) and (u, in) are known as symport rules (uniport if $|u| = 1$) while the rule type $(u, out; v, in)$ is known as an antiport rule. A rule of the form (u, out) in membrane i sends the elements of u from membrane i out to the membrane (directly) containing i. A rule of the form (u, in) in membrane i transports the elements of u into membrane i from the membrane enclosing i. Hence, this rule can only be used when the elements of u exist in the outer membrane. A rule of the form $(u, out; v, in)$ simultaneously sends u out of the membrane i while transporting v into membrane i. Hence, this rule cannot be applied unless membrane i contains the elements in u and the membrane surrounding i contains the elements in v. These systems can be used as generators or acceptors of sets of numbers.

This model was initially found to be computationally complete for systems with five or more membranes [18]. Since this original paper, more restricted systems have been studied in terms of the number of objects in the alphabet, the number of membranes, and the size of the rules. Many computationally complete bounds have been found. These results can be found in various papers found in [1].

Here, we examine additionally restricted SA P system models. We analyze minimal systems with regard to the size of the alphabet and the number of membranes. We study the precise power of very simple SA P systems with either 1, 2, or 3 symbols and less than 5, 4, and 3 membranes, respectively. This improves the previous work in [2] and [19]. However, the question of whether using only a single symbol with any number of membranes is computationally complete remains open.

We then look at restricted forms of SA P systems (called *bounded SA P systems* and *special SA P systems*) where we find an infinite hierarchies with respect to the both the size of the alphabet and the number of membranes. We also analyze the role of determinism versus nondeterminism and find that over a unary input alphabet, these systems are equivalent if and only if deterministic and nondeterministic linear-bounded automata (over an arbitrary input alphabet) are equivalent. Similar results above have been shown for other types of restricted P systems (that are not symport/antiport) [12]. However, these previous results do not easily translate for the models of SA P systems considered here.

Finally, we introduce some restricted SA P system models which characterize semilinear sets. A set $Q \subseteq \mathbb{N}^k$ is a *linear set* if there exist vectors $v_0, v_1, \ldots, v_t$ in $\mathbb{N}^k$ such that $Q = \{v \mid v = v_0 + m_1 v_1 + \cdots + m_t v_t, m_i \in \mathbb{N}\}$. The vectors v_0 (referred to as the *constant vector*) and $v_1, v_2, \ldots, v_t$ (referred to as the *periods*) are called the *generators* of the linear set Q. A set $Q \subseteq \mathbb{N}^k$ is *semilinear* if it is a finite union of linear sets [7]. The empty set is a trivial semilinear set, where the set of generators is empty. Every finite subset of $\mathbb{N}^k$ is semilinear—it is a finite union of linear sets whose generators are constant vectors. It is also clear that the semilinear sets are closed under (finite) union. It is also known that they are closed under complementation, intersection, and projection. A semilinear subset of $\mathbb{N}^1$ (i.e., 1-tuples) is sometimes referred to as regular. Examples include $L = \{(n, 2n) \mid n \geq 1\}$

and $L = \{(r, s, t) \mid r \leq s \leq t \text{ and } r, s, t \geq 1\}$. We also show that "slight" extensions of the models will allow them to accept (respectively, generate) nonsemilinear sets. In fact, for these extensions, the emptiness problem is undecidable (i.e., no algorithm exists which decides the following question: given an arbitrary SA P system Π in the class, is the language accepted by Π empty?).

2 SA P Systems with a Small Number of Objects

SA P system completeness results try to determine how "minimal" of a system is needed for completeness to hold. Minimal systems restrict many aspects of the system including how many objects are used, how many membranes are used, and the size of the rules which are used. The best known previous results (before the results we present here) can be found in [2] where the number of symbols can be limited to 3 and the number of membranes to 3 and computational completeness is still achieved. This result was expanded to show 2 objects and 4 membranes, 4 objects and 2 membranes, and 5 objects and 1 membrane are computationally complete.

In [2], some partial results were given for minimal systems which are not known to be computationally complete. These results are stated here:

1. The languages $L_k = \{ki \mid i \in \mathbb{N}\}$ can be accepted by 1-symbol 1-membrane SA P systems.
2. The family of all finite sets of natural numbers is contained in the family of 1-symbol 1-membrane SA P system acceptors.
3. A set $Q \subset \mathbb{N}$ can be generated by a 1-symbol 1-membrane SA P system acceptor if and only if Q is finite.
4. The family of all regular sets of natural numbers is contained in the family of 1-symbol 2-membrane SA P system generators.
5. The family of all regular sets of natural numbers is contained in the family of 2-symbol 1-membrane SA P system generators.

(A *i-symbol j-membrane SA P system* is a SA P system which uses at most i symbols and at most m membranes.) Here, we improve the above results. We also find that 1-symbol multimembrane SA P systems (acceptors and generators) are surprisingly powerful.

2.1 2-*Symbol* 1-*Membrane SA P System Acceptors*

In this section, we consider SA P system acceptors which are restricted to two symbols and one membrane. Previous results in [2] show that all finite sets of natural numbers can be accepted by this type of restricted SA P system. These results, however, can be improved.

Theorem 1 *The family of all finite sets of natural numbers is properly contained in the family of* 2*-symbol* 1*-membrane SA P system acceptors.*

Proof The result that the family of all finite sets of natural numbers is contained in the family of 2-symbol 1-membrane SA P system acceptors was shown in [2]. To see that this containment is proper, we show that some nonfinite set of natural numbers can be accepted by a SA P systems with two symbols and one membrane. Consider the set $L = \{2^n \mid n \geq 0\}$. The following SA P system with only 2 symbols and 1 membrane accepts L, and hence proves that the inclusion is strict. Define $\Pi = \langle \Sigma, \mu, w_1, R_1, i_o \rangle$ where $\Sigma = \{o, t\}$, $\mu = [_1]_1$, $w_1 = \lambda$, $R_1 = \{(oo, out; o, in), (o, out; t, in), (tt, out; tt, in)\}$, $i_o = 1$.

Informally, this machine has the input o^i in the skin membrane initially. During each step, the input is reduced by half. If there is an odd number of o's, at least one t will be drawn into the membrane. Eventually, the number of o's will be reduced to one. This will draw a single t into the membrane during the next step. If the computation ever draws in more than one t, the computation will never halt. An input is accepted if the number of o's is always even (until the last step) and always uses the first rule to apply to each pair of o's. This will only occur if the input given is in L. □

Since L is not a semilinear set, the above proof also gives us the following result.

Corollary 1 *2-Symbol 1-membrane SA P system acceptors* $\neq$ *semilinear sets over* $\mathbb{N}$.

We can, however, show that all regular sets (and hence all semilinear sets) can be accepted by SA P systems with two symbols and one membrane. This improves the above result.

Theorem 2 *The family of all regular sets of natural numbers is properly contained in the family of 2-symbol 1-membrane SA P system acceptors.*

Proof From Corollary 1, we need only show that the family of all regular sets of natural numbers is contained in the family of 2-symbol 1-membrane SA P system acceptors. We give a construction such that given a regular set of natural numbers creates a SA P system with two symbols and one membrane accepting the same set. Any unary regular set M can be defined in terms of some sets M_0 and M_1 such that $M = M_0 \cup \{i + jk \mid i \in M_1, j \in \mathbb{N}\}$ for some k. From M, we create the SA P system $\Pi = \langle \{o, t\}, [_1]_1, \lambda, R_1, 1 \rangle$ where $R_1 = \{(o^i, out; t^2, in) \mid i \in M_0\} \cup \{(o^i, out; t^3, in) \mid i \in M_1\} \cup \{(o^k t^3, out; t^3, in)\} \cup \{(t^4, out; t^4, in), (o, out; o, in)\}$.

Informally, the resulting SA operates by using a trap symbol t to guarantee that no wrong path of the computation is taken. This is done by capping the number of t symbols to three in a correct computation. If four or more t symbols are brought into the membrane, the rule $(t^4, out; t^4, in)$ will guarantee that the computation will never halt. The rule $(o, out; o, in)$ is used to make sure the computation never halts if all the o's are not used in other rules (i.e., the input was not in accepted by $\mathcal{M}$).

The computation works by first choosing to either check that the o's make up an element in M_0 or they make up an element in $\{i + jk \mid i \in M_1, j \in \mathbb{N}\}$. This is done by using one of the first two rules in the first step. (The remaining o's will use the rule $(o, out; o, in)$ to remain unchanged. Also, it is important to note that if more than one instance of one of the first two rules is applied, at least four t symbols will be brought into the system.) If the second rule type was applied, the computation now is able to remove o^k in each prior step until the number of remaining o's is less than k. If the input is accepted, the computation will reach a step where there are no more o's in the membrane and less than four t's resulting in no applicable rules. □

2.2 1-*Symbol* 3-*Membrane SA P System Acceptors*

Very little so far is known about the power of SA P systems with only one symbol. Symport/antiport P system generators are known to be able to generate at least regular sets if they are allowed at least two membranes [2], but what about acceptors? Here, we find they are able to accept some nonsemilinear sets.

Theorem 3 *The family of languages accepted by* 1-*symbol* 3-*membrane SA P systems contain some nonsemilinear sets.*

Proof The following 1-symbol 3-membrane SA P system accepts the nonsemilinear set $L = \{2^n \mid n \geq 0\}$. Let $\Pi = \langle \Sigma, \mu, w_1, w_2, w_3, R_1, R_2, R_3, i_o \rangle$ where $\Sigma = \{o\}$, $\mu = [_1[_2[_3]_3]_2]_1$, $w_1, w_2, w_3 = \lambda$, $R_1 = \{(oo, out; o, in)\}$, $R_2 = \{(o, in)\}$, $R_3 = \{(oo, out), (oo, in)\}$, $i_o = 1$.

This system works by dividing the number of o's in membrane 1 in half until only a single o is left. If the input was of the form o^{2^n}, each step will leave an even number of o's in membrane 1 until the last step which will draw the final o into membrane 2 and the computation will halt. If the input was not of the form o^{2^n}, some step will have an odd number of o's (>1) in membrane 1. Since the computation must work in a maximally parallel manner, at least one o will be drawn into membrane 2 before the final step. If this occurs, at least two o's will be located in membrane 2 after no o's remain in membrane 1. These o's will travel between membrane 2 and membrane 3 forever causing an infinite loop. (This loop will also occur if the rule in membrane 1 is not exhausted before the rule in membrane 2 is applied. This type of computation signifies a wrong guess in the nondeterministic branching.) □

2.3 1-*Symbol Multimembrane SA P System Acceptors*

In the previous section, we saw that SA P system acceptors with a single symbol can accept nonsemilinear sets. So, a natural question is: What is the computing power of a multimembrane acceptor which uses only a single symbol? Although it is still open whether such systems are universal, we can show that they are more powerful than partially blind multicounter machines.

A *partially blind multicounter machine* (PBCM) [8] is a restricted type of counter machine which consists of a finite number (call the number m) of counters which can add one and subtract one, but cannot test for zero. If there is an attempt to decrement a zero counter, the system aborts and does not accept. A PBCM can be used as an acceptor of tuples. In this case, the first k counters (for some $k \le m$) are input counters. The system is started with some nonnegative integers $(n_1, \dots, n_k)$ in the input counters and the other counters set to zero. The input tuple is accepted if the system reaches a halting state and all the counters are zero. Hence, the language accepted by a PBCM is the set of k-tuples of nonnegative integers accepted by the system.

A PBCM can also be used as a generator of numbers. In this case, all m counters are initially zero. When the system halts, the computation is only considered valid if the nonoutput counters are all zero. If this is the case, the k-tuple created from the values of the k-output counters is said to be generated. Without loss of generality, it can be assumed that the k-output counters are strictly increasing counters.

Formally, a PBCM is defined as $\mathcal{M} = \langle m, B, l_0, l_h, R\rangle$ where m is the number of partially blind counters in the system, B is the set of instruction labels, l_0 is the starting instruction, l_h is the halting instruction, and R is the set of instructions. The instructions in R are of the forms: $l_i : (\mathtt{ADD}(r), l_j, l_k)$; $l_i : (\mathtt{SUB}(r), l_j)$; $l_i : (\mathtt{HALT})$ where l_i, l_j, and l_k are instruction labels and r is the counter that should be added to or subtracted from. If the instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$ is executed, either instruction l_j or l_k will execute next (picked nondeterministically). If the instruction $l_i : (\mathtt{SUB}(r), l_j)$ is executed, instruction l_j is executed next (assuming the computation did not abort). When the instruction $l_i : (\mathtt{HALT})$ is executed, the system halts. If the PBCM system is used as an acceptor, a deterministic model can also be considered. A deterministic PBCM is defined in the same manner except that the instruction $l_i : (\mathtt{ADD}(r), l_j, l_k)$ becomes $l_i : (\mathtt{ADD}(r), l_j)$.

For some $k \ge 2$, PBCMs can accept nonsemilinear sets over $\mathbb{N}^k$. When $k = 1$, PBCMs accept exactly the semilinear sets over $\mathbb{N}$. It is known that k-output PBCMs can be simulated by vector addition systems, and vice versa [8]. (Hence, such counter machines are not universal.) In particular, a k-output PBCM can generate the reachability set of a vector addition system. However, k-output PBCMs are strictly less powerful than unrestricted multicounter machines (which are equivalent to Turing Machines).

It is also known that the family of k-tuples generated by PCBMs is closed under union and intersection, but not under complementation. The membership, emptiness, infiniteness, disjointness, and reachability problems are decidable for PBCMs; but containment and equivalence are undecidable.

A related model called *blind multicounter machine* [8] is a multicounter machine that can add one and subtract one from a counter, but cannot test a counter for zero. The difference between this model and a partially blind counter machine is that a blind counter machine does not abort when a zero counter is decremented. Thus, the counter stores a negative number. Again, an input is accepted if the computation reaches an accept state and all the counters are zero. We note that a blind counter machine is equivalent in power to reversal bounded counter machines [8] which are equivalent to semilinear sets [13].

The next theorem is an improvement of a result in [19] where it was shown that a partially blind counter machine with m-counters (operating as an acceptor) can be simulated by a SA P system with two symbols and $3(m+1)$ membranes. Our construction is a rather intricate modification of the construction in [19].

Theorem 4 *Any partially blind counter machine $\mathcal{M}$ acceptor with m counters can be simulated by some* 1*-symbol* $(2m+3)$*-membrane SA P system acceptor* Π.

Proof Let $\mathcal{M} = \langle m, B, l_o, l_h, R\rangle$ where $\mathcal{M}$ is a partially blind counter machine with m counters. Assume that the start instruction (l_o) is labeled l_1 and each additional instruction is labeled sequentially as $l_2, l_3, \ldots, l_{|R|}$. Also, assume the last labeled instruction, $l_{|R|}$ is the HALT instruction (l_h) and no other HALT instructions exist.

The instructions of $\mathcal{M}$ are encoded with distinct numbers. These numbers have useful properties much like the encoding in the construction of [19]. In [19], the main property of the encoding guarantees that for a set of encoded numbers, each element of the set is distinct. In other words, given an encoded number o^i, there exists no set of additionally encoded numbers such that $o^{j_1} + \cdots + o^{j_n} = o^i$ (where the o^j's do not need to be distinct, but are not equal to o^i). In our construction, we use this idea and extend it so that we can encode two sets of numbers (call them set A and set B) such that given two encoded numbers $o^i \in A$ and $o^j \in B$, the encoding is distinct. This means that there does not exist a set of additionally encoded numbers such that $o^{k_1} + \cdots + o^{k_n} = o^i + o^j$ (where the o^k's do not need to be distinct, but are not equal to either o^i or o^j). This codification is done in two parts.

First, for each element in the set $B \cup R$ we associate a distinct natural number from the set $\{1, 2, \ldots, n\}$ where $n = |B \cup R|$. The mapping is given here with our way of denoting each distinct natural number along with the natural number itself.

1. $\forall l_i \in B$ (the label associated with some instruction), $l_i = 2i - 1$.
2. $\forall l_i : (\texttt{ADD}(r), l_j) \in R$ (an addition instruction itself), $A_{ir} = 2i$.
3. $\forall l_i : (\texttt{SUB}(r), l_j) \in R$ (a subtraction instruction itself), $S_{ir} = 2i$.

The $l_i : (\texttt{HALT})$ instruction requires two distinct natural numbers for each counter in the system. These numbers will be encoded over the set $\{(n+1), \ldots, (n+2m)\}$. This association is created as follows.

1. $\forall\, i (1 \le i \le m)$, $H_i = (2i-1) + n$ and $H_{i'} = 2i + n$

The second part of the codification is done by encoding each of the previously defined natural numbers. This encoding will guarantee that the set of applicable rules at each step has only one distinct 'correct' path. (A "noncorrect" path traps at least some objects causing an infinite loop.) This encoding is done with the function $v_\alpha(i) = 12mn' + 12mi$ where $n' = n + 2m + 1$.

The second set of encoded numbers denote internal "commands." These commands deal with the actual addition and subtraction of objects in the counter membranes. The set $\{ADD_j, SUB_j, REMOVE_j, CHECK_j\}$ for $1 \le j \le m$ denotes these commands. Each of these elements is encoded by the function v_β as follows.

1. $v_\beta(ADD_j) = 24mn' - 4(j-1) - 1$ (adds a number to counter j).
2. $v_\beta(REMOVE_j) = 24mn' - 4(j-1) - 2$ (signifies the instruction is completed).
3. $v_\beta(SUB_j) = 24mn' - 4(j-1) - 3$ (subtracts a number from counter j).
4. $v_\beta(CHECK_j) = 24mn' - 4(j-1) - 4$ (makes sure the counters halt in the appropriate configuration).

Some important properties of the encoding created by the functions v_α and v_β follow:

- Each $v_\alpha(i)$ is strictly greater than half of any other $v_\alpha(j)$.
- Each $v_\beta(i)$ is a distinct value between $24mn' - (24m)/2$ and $24mn'$.
- Each $v_\beta(i) - 24m(n'-1)$ is strictly greater than half of any other $v_\beta(j) - 24m(n'-1)$.
- The addition of any $v_\alpha(i)$ with some $v_\beta(j)$ is distinct and cannot be created using any other combination of $v_\alpha(i')$'s and $v_\beta(j')$'s. (This is due to the fact that every $v_\beta(j')$ is larger than any $v_\alpha(i')$. It is also due to the fact that every $v_\alpha(i')$ is a multiple of m and every $v_\beta(j')$ is not a multiple of m.)

Using the defined encoding of the instructions, we now create a SA P system Π to simulate $\mathcal{M}$ with $2m+3$ membranes. For ease of understanding, we will label these membranes with the labels $\{s, c_{i1}, c_{i2}, t_1, t_2 \mid 1 \le i \le m\}$. We use s to denote the skin membrane which controls the execution of each instruction. We use c_{i1} and c_{i2} $(1 \le i \le m)$ to denote membranes used to store the count of the counter i. We use the labels t_1 and t_2 to denote membranes that are used to "trap" incorrect moves and force an infinite looping of rules. Formally, we define the system as follows:

$$\Pi = \langle \Sigma, \mu, w_1, \ldots, w_{2m+3}, R_1, \ldots, R_{2m+3}, i_o \rangle$$

where $\Sigma = \{o\}$, $\mu = [_s[_{t_1}[_{t_2}]_{t_2}]_{t_1}[_{c_{11}}[_{c_{12}}]_{c_{12}}]_{c_{11}}[_{c_{m1}}[_{c_{m2}}]_{c_{m2}}]_{c_{m1}}]_s$, $w_s = o^{v_\alpha(l_1)}$, $w_{c_{i1}} = o^{v_\beta(REMOVE_i)}$ for $1 \le i \le m$, $w_{c_{i2}} = o^{v_\beta(SUB_i)}$ for $1 \le i \le m$, and $w_{t_1}, w_{t_2} = \lambda$ for $1 \le i \le m$. The rules are created as follows:

For each instruction of the form $l_i = (\mathtt{ADD}(r), l_j)$ add the following rules to membrane s:

1. $(o^{v_\alpha(l_i)}, out; o^{v_\alpha(A_{ir})} o^{v_\beta(ADD_r)}, in)$.
2. $(o^{v_\alpha(A_{ir})}, out; o^{v_\alpha(l_j)}, in)$.

For each instruction of the form $l_i = (\mathtt{SUB}(r), l_j)$ add the following rules to membrane s:

3. $(o^{v_\alpha(l_i)}, out; o^{v_\alpha(S_{ir})} o^{v_\beta(SUB_r)}, in)$.
4. $(o^{v_\alpha(S_{ir})}, out; o^{v_\alpha(l_j)}, in)$.

For the instruction of the form $l_i = (\mathtt{HALT})$ add the following rules to membrane s:

5. $(o^{v_\alpha(l_i)}, out; o^{v_\alpha(H_1)}, in)$.
6. $(o^{v_\alpha(H_i)}, out; o^{v_\alpha(H_{i'})} o^{v_\beta(CHECK_i)}, in)$ for $1 \le i \le m$.
7. $(o^{v_\alpha(H_{i'})}, out; o^{v_\alpha(H_{i+1})}, in)$ for $1 \le i < m$.
8. $(o^{v_\alpha(H_{m'})}, out)$.

The following additional rules must be added to the system:

9. In membrane s, for each counter r where $1 \le r \le m$ add the rule $(o^{v_\beta(REMOVE_r)}, out)$.
10. Add the following rules to membrane c_{r1} for $1 \le r \le m$:
    (a) $(o^{v_\beta(REMOVE_r)}, out; o^{v_\beta(ADD_r)}, in)$,
    (b) $(o^{v_\beta(REMOVE_r)}, out; o^{v_\beta(SUB_r)}, in)$,
    (c) $(o^{v_\beta(REMOVE_r)}, out; o^{v_\beta(CHECK_r)}, in)$.
11. Add the following rule to membrane c_{r2} for $1 \le r \le m$ $(o^{v_\beta(SUB_r)}, out; o^{v_\beta(SUB_r)}, in)$.
12. Add the rule (o, in) to membrane t_1.
13. Add the rules (o, in) and (o, out) to membranes t_2.

The basic structure containing only the rules not associated with specific instructions is given in Fig. 1. Here, we see the two trap membranes and their associated rules. Also, we see the two membranes for each counter along with their operational rules. Finally, the skin membrane surrounds all these membranes.

The fundamental idea necessary for the simulation is due to the manner in which the instructions and commands are encoded using the functions $v_\alpha()$ and $v_\beta()$. Because of the encoding, any single $v_\alpha(i)$ and $v_\beta(j)$ can be added to each other and these numbers are still distinctly distinguishable. Since the construction never has more than a single $v_\alpha(i)$ and a single $v_\beta(j)$ in any membrane during the same step, the multiset of o's can only be broken up as $v_\alpha(i)$ and $v_\beta(j)$ without leaving remaining o's to be drawn into the trap membranes. If the wrong rule set is picked, at

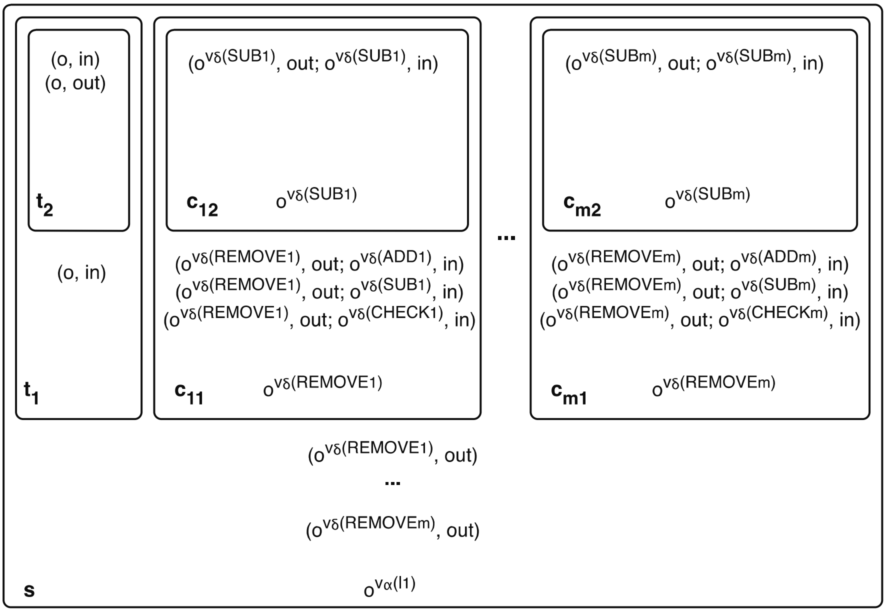

(o, in)
(o, out)
t_2
(o, in)
t_1
$(o^{v\delta(SUB1)}, out; o^{v\delta(SUB1)}, in)$
c_{12}
$o^{v\delta(SUB1)}$
$(o^{v\delta(REMOVE1)}, out; o^{v\delta(ADD1)}, in)$
$(o^{v\delta(REMOVE1)}, out; o^{v\delta(SUB1)}, in)$
$(o^{v\delta(REMOVE1)}, out; o^{v\delta(CHECK1)}, in)$
c_{11}
$o^{v\delta(REMOVE1)}$
...
$(o^{v\delta(SUBm)}, out; o^{v\delta(SUBm)}, in)$
c_{m2}
$o^{v\delta(SUBm)}$
$(o^{v\delta(REMOVEm)}, out; o^{v\delta(ADDm)}, in)$
$(o^{v\delta(REMOVEm)}, out; o^{v\delta(SUBm)}, in)$
$(o^{v\delta(REMOVEm)}, out; o^{v\delta(CHECKm)}, in)$
c_{m1}
$o^{v\delta(REMOVEm)}$
$(o^{v\delta(REMOVE1)}, out)$
...
$(o^{v\delta(REMOVEm)}, out)$
s
$o^{v\alpha(l1)}$

Fig. 1 Basic membrane structure used to simulate a PBCM

least one extra o will remain in the membrane and the trap membranes will draw in the o causing an infinite looping of rules to occur.

Informally, the process of Π occurs as a 2-part system. The set of numbers encoded by the function $v_\alpha()$ control the sequence of instructions which are executed and the set of numbers encoded by the function $v_\beta()$ control the actual processing of increment and decrement instructions.

An instruction of the form $l_i : (\mathtt{ADD}(r), l_j)$ begins with the encoded multiplicity for the instruction label within membrane s. This brings in both the multiplicity of o associated with the instruction itself along with $o^{v_\beta(ADD_r)}$ (using rule 1). The encoded ADD_r command adds a single o to membrane c_{r1} (by swapping $o^{v_\beta(ADD_r)}$ with $o^{v_\beta(REMOVE_r)}$ where $ADD_r = REMOVE_r + 1$ using rule 10(a)). Simultaneously, rule 2 is used in membrane s to throw out $o^{v_\alpha(A_{ir})}$ and draw in $o^{v_\alpha(l_j)}$, leaving $o^{v_\alpha(l_j)} + o^{v_\beta(REMOVE_r)}$ in membrane s. To complete the addition process, $o^{v_\beta(REMOVE_r)}$ is thrown out of membrane s (rule 9). Simultaneously, $o^{v_\alpha(l_j)}$ are thrown out of membrane s and the appropriate o's are drawn in to execute the next instruction.

An instruction of the form $l_i : (\mathtt{SUB}(r), l_j, l_k)$ operates almost identically to that of the $l_i : (\mathtt{ADD}(r), l_j)$ instruction except that $o^{v_\beta(ADD_r)}$ is replaced with $o^{v_\beta(SUB_r)}$.

If the wrong maximal multiset of rules is ever chosen in a given step, some number of o's will not be used in instruction/command steps (due to the fact that no $v_\alpha(i) + v_\beta(j)$ can be evenly broken up into any other set of $v_\alpha()$'s and $v_\beta()$'s). These additional o's will be drawn into trap membrane t_1 using rule 12. Once an o is within this trap membrane, the computation will never halt since the o will constantly pass between the outer trap membrane and the inner trap membrane using the rules in 13.

Handling the simulation of a subtract instruction from a zero counter operates in a slightly convoluted manner. The subtract instruction itself is allowed to process, but the system "blocks" (i.e., enters a state where an infinite loop of applicable instructions is guaranteed) when the next instruction is processed. (Since $l_i : (\mathtt{HALT})$ is processed as if it is an instruction, we are guaranteed to have an additional instruction following a "subtract from zero" instruction.) When a subtract from zero occurs, the contents of c_{i1} will be left with just $o^{v_\beta(SUB_i)}$. All instructions require that $o^{v_\beta(REMOVE_i)}$ be removed from c_{i1} (rules 10(a), 10(b), and 10(c)) in order to process. Since $o^{v_\beta(REMOVE_i)}$ is $o^{v_\beta(SUB_i)} + 1$, this rule is not applicable if a subtract from zero occurs and the extra o's in membrane s must be drawn into membrane t_1 where they will be shuttled between t_1 and t_2 forever.

To check that all of the counters halt containing a count of zero, the encoding of the set $\{H_i \mid 1 \leq i \leq m\}$ is used to stall the computation one step by bringing in the appropriate encoding of the set $\{H_{i'} \mid 1 \leq i \leq m\}$ (using rule 6). When $o^{v_\alpha(H_i')}$ is brought in, so is the encoded command $CHECK_j$. Each $CHECK_j$ is processed after the encoded $CHECK_{(j-1)}$ has been processed using rules 6 and 7. In the next step, $o^{v_\beta(CHECK_j)}$ is exchanged with $o^{v_\beta(REMOVE_j)}$ in membrane c_{j1}. If membrane c_{j1} previously contained a zero count, it now contains only $o^{v_\beta(CHECK_j)}$. This will cause rule 11 to no longer be applicable. If c_{j1} previously contained a count greater than zero, rule 11 in membrane c_{j2} to be applicable forever. Finally, membrane s

removes objects $o^{v_\beta(REMOVE_j)}$ while the encoded halt instruction for counter $j+1$ is drawn in.

Nondeterminism in this construction is essential. Clearly, a symport/antiport system with one symbol must operate nondeterministically if more than one rule of either type (u, in) or (u, out) occurs in any membrane. A deterministic system would require most rules to be antiport rules and consist of a rather tricky construction. In the above construction, each step often has many possible applicable rules, but only one set of "valid" rules using all of the objects in the membrane. If not all the objects are used in "valid" rules, the remaining objects are trapped. Nondeterminism picks the rule set which uses all of the current o objects without using a trap rule. □

In the above proof, the definition of a SA P acceptor was loosened, initially allowing a fixed number of input symbols in the input membrane along with the input itself. This relaxed definition is not necessary. If we require the input membrane to initially contain only the input itself (the strict SA definition), then we only need to make the following changes to our construction.

1. Initially $w_{c_{i1}} = \lambda$; $w_{c_{i2}} = o^{v(\beta_{iSUB})+v(\beta_{iREMOVE})}$.
2. Add the rule $(o^{v(\beta_{iREMOVE})}, out)$ to membrane c_{i2}.

Now, our simulation operates identically to the previous simulation after the first step of computation.

It can be noted that our simulation does not use an unbounded number of maximally parallel steps in a halting computation. At any point during the simulation, at most two rules are applied in any membrane during a single step (except within the trap membranes if a computation takes a wrong path). While the number of rules which may be applicable in the trap membranes could be large, we only need one rule to actually execute at each step to guarantee an infinite loop. Hence, the rules in these membranes are not affected by limiting the number of maximally parallel steps. We will denote a restricted model of a symport/antiport P system k-tuple acceptor with m objects and n membranes which restricts the number of rules per membrane per step to x as an *i-symbol j-membrane x-parallel SA P system acceptor*.

Corollary 2 *Any partially blind counter machine $\mathcal{M}$ acceptor with m counters can be simulated by some* 1*-symbol* $(2m+3)$*-membrane* 2*-parallel SA P system acceptor Π.*

Proof This follows from Theorems 4 and 3. □

These results can also be used to give an upper bound on the number of membranes needed for a single object SA P system to simulate any regular set. It is easy to see that a 1-counter PBCM can accept any regular set by initially starting with the input in the counter, decrementing the counter, and simulating the finite automaton (FA) accepting the regular set. The 1-counter PBCM accepts (guessing that the counter is zero) if the FA accepts. Since Theorem 4 gives a construction to simulate an m-counter PBCM with $2m+3$ membranes, we can simulate a 1-PBCM with a 1-symbol 5-membrane SA P system acceptor. (Again the inclusion is strict due to Theorem 3.)

Corollary 3 *The family of all regular sets of natural numbers is properly contained in the family of* 1*-symbol* 5*-membrane SA P system acceptors.*

We will show that, in fact, 1-symbol multimembrane SA P system acceptors are more powerful than partially blind counter machines. Consider a SA acceptor $\mathcal{P}$ with only one symbol o and r membranes $m_1, \dots, m_r$. Without loss of generality (by relabeling the membranes), assume that the first k membranes $m_1, \dots, m_k$ are the input membranes. There are fixed multisets $o^{s_1}, \dots, o^{s_r}$ such that at the start of the computation, the input membranes are given $o^{s_1+n_1}, \dots, o^{s_k+n_k}$ for some nonnegative integers $n_1, \dots, n_k$ and the other membranes are given o^{s_j} for $j = k+1, \dots, r$. If the system halts, then we say that the k-tuple $(n_1, \dots, n_k)$ is accepted. The set of all such k-tuples is the set accepted by $\mathcal{P}$. Then the following corollary follows from Theorem 4.

Corollary 4 *If $L \subseteq \mathbb{N}^k$ is accepted by a partially blind counter machine with $m \geq k$ counters* (*note that the first k counters are input counters*), *then L can be accepted by a* 1*-symbol*, $(2m+3)$*-membrane SA acceptor.*

Corollary 5 *Let G be a k-dimensional vector addition system with states* (*VASS*), *and $R(G) \subseteq \mathbb{N}^k$ be the reachability set of G. Then $R(G)$ can be accepted by a* 1*-symbol multimembrane SA P system acceptor.*

Proof Let $G = \langle x, W, T, p_0 \rangle$ be a VASS. We need only show that $R(G)$ can be accepted by a partially blind counter machine $\mathcal{M}$. The construction is straightforward. $\mathcal{M}$ has $2k$ counters, where the first k counters are the input counters. Given input $y = (n_1, \dots, n_k)$ in the first k counters and zero in the last k counters, $\mathcal{M}$ first stores the k-tuple x in the last k counters. Then $\mathcal{M}$ simulates the computation of G (i.e., applies the vectors in W) on the last k counters. At some point nondeterministically chosen, $\mathcal{M}$ guesses that the k-tuple w in the last k counters is equal to the k-tuple y in the input counters. Then for each i, $\mathcal{M}$ decrements counter i and $n+i$ by 1 simultaneously a nondeterministic number of times in state q_i after which $\mathcal{M}$ jumps to state q_{i+1}. After processing all the counters, $\mathcal{M}$ enters an accepting state. □

We can now show that 1-symbol multimembrane SA P system acceptors are more powerful than partially blind counter machines.

Theorem 5 1*-Symbol multimembrane SA P system acceptors are more powerful than partially blind counter machines.*

Proof From Theorems 3 and 4, we need only show that $L = \{2^n \mid n \geq 0\}$ cannot be accepted by a partially blind counter machine.

The idea is the following. Given a partially blind counter machine $\mathcal{M}$ with $k+1$ counters with the first counter being the input counter, accepting $L(\mathcal{M}) \subseteq \mathbb{N}$, we construct a $(k+3)$-dimensional VASS $G = \langle x, W, T, p_0 \rangle$ (for some k), such that $R(G) \cap (\mathbb{N} \times \{0\}^{k+2}) = \{(n, 0, \ldots, 0) \mid n \in L(M)\}$. Let $L(G)$ be the projection of $R(G) \cap (\mathbb{N} \times \{0\}^{k+2})$ on the first coordinate. Then from the results in [9], $L(G)$ is semilinear. Hence, $L(\mathcal{M})$ is also semilinear (since $L(G) = L(\mathcal{M})$).

We now describe the construction of G from $\mathcal{M}$. Initially starting with $x = (0, 1, 0, \ldots, 0)$, G increments coordinates 1 and 3 by 1 simultaneously a nondeterministic number of times, say, n. At this point, the coordinates will have values $(n, 1, n, 0, \ldots, 0)$. Then G simulates the computation of $\mathcal{M}$ on the last $k+1$ coordinates $(n, 0, \ldots, 0)$ until $\mathcal{M}$ enters an accepting state, guessing that its $k+1$ counters are zero. G then decrements the second coordinate by 1.

Clearly, the projection of $R(G) \cap (\mathbb{N} \times \{0\}^{k+2})$ on the first coordinate is $L(\mathcal{M})$, and the result follows. □

2.4 1-*Symbol Multimembrane SA P System Generators*

In the previous section, we defined both PBCM acceptors and PBCM generators. The following theorem shows that these two PBCM models are equivalent.

Theorem 6 *Let $k \geq 1$ and $L \subseteq \mathbb{N}^k$.* (1) *If $\mathcal{M}$ is a PBCM generator with m counters generating L, then we can construct a PBCM acceptor $\mathcal{M}'$ with $m+k$ counters accepting L.* (2) *If $\mathcal{M}$ is a PBCM acceptor with m counters accepting L, then we can construct a PBCM generator $\mathcal{M}'$ with $m+k$ counters generating L.*

Proof Part 1: With input $(n_1, \ldots, n_k)$ on its first k counters, $\mathcal{M}'$ simulates $\mathcal{M}$ on the remaining m counters. When $\mathcal{M}$ enters a final state, then for each i, $\mathcal{M}'$ decrements counter i and $k+i$ by 1 simultaneously a nondeterministic number of times in state q_i after which $\mathcal{M}'$ jumps to state q_{i+1}. After processing all the counters, $\mathcal{M}'$ enters an accepting state.

Part 2: $\mathcal{M}'$ first nondeterministically generates a $2k$-tuple $(n_1, n_2, \ldots, n_k, n_1, n_2, \ldots, n_k)$ in the first two-counters. Then $\mathcal{M}'$ simulates $\mathcal{M}$ on the last m counters and halts in a final state if $\mathcal{M}$ accepts. □

Now consider a 1-symbol r-membrane SA generator $\mathcal{P}$. The system has only one object o and r membranes $m_1, \ldots, m_r$, where the first $k \leq r$ membranes $m_1, \ldots, m_k$ are output membranes. There are fixed multisets $o^{s_1}, \ldots, o^{s_r}$ such that at the start of the computation, each membrane m_j is set to o^{s_j} for $1 \leq j \leq m$. If the system halts with $o^{s_1+n_1}, \ldots, o^{s_k+n_k}$ in membranes $m_1, \ldots, m_k$, then we say that the k-tuple $(n_1, \ldots, n_k)$ is generated. The set of all such k-tuples is the set generated by $\mathcal{P}$. Then as with the 1-symbol multimembrane SA P system acceptors, we can show the following.

Theorem 7 *Let $L \subseteq \mathbb{N}^k$. (1) If L is generated by a PBCM with $m \geq k$ counters, then L can be generated by a 1-symbol $(2m+3)$-membrane SA P system generator. (2) If L is the reachability set of a VASS, then L can be generated by a PBCM generator, and hence also by a 1-symbol multimembrane SA P system generator.*

Proof Part (1) can be proved by a simple modification to the proof of Theorem 4. The definition of a PBCM acceptor specifies the counters must contain zero after the computation halts for a given input to be accepted. The definition of a PBCM generator requires the nonoutput counters be zero after halting. Therefore, the simulation of the instruction HALT must be changed in order to simulate a generator rather than an acceptor. The basic idea is to reencode the HALT instruction by removing the instructions which test the output counters end with a count of zero. The membrane c_{r2} and the rule 11 for each r where $1 \leq r \leq k$ must be removed so that a positive counter value does not cause the system to infinitely loop. (Note that this infinite loop is necessary in the case of an acceptor, since all the counters must end with zero for the input tuple to be accepted.)

Also, since an additional instruction is needed to guarantee that a subtract from zero instruction was not executed during the last steps, the halt instruction simulates adding and immediately subtracting one from each output counter. (This requires a change to the encoding to account for $2k$ more instructions. We omit the details.) Part (2) follows from Part (1) and Theorem 6. □

We can also bound the maximal parallelism for this system to two rules per step per membrane just like we did for the acceptor version giving us the following result.

Corollary 6 *Any partially blind counter machine $\mathcal{M}$ generator with m counters can be simulated by some 1-symbol $(2m+3)$-membrane 2-parallel SA P system generator Π.*

It is easy to see that a PBCM generator with only one counter can generate any regular set by incrementing the counter and simulating the state transitions of the finite automaton (FA) accepting this regular set. If the FA accepts, the 1-counter PBCM generator enters the final state. Thus, from Theorem 7, we can see the family of all regular sets of natural numbers is contained in the family of 1-symbol 5-membrane SA P system generators. However, this is not minimal since it was shown in [2] that the family of all regular sets of natural numbers is contained in the family of 1-symbol 2-membrane SA P system generators. It is open whether these inclusions are proper (even if we allow the 1-symbol SA generator to have any number of membranes).

Remark 1 It is also open whether the family of languages generated by 1-symbol multimembrane SA P systems is able to generate any languages which are not in the class of languages generated by PBCMs. If they cannot, this would means that $L = \{o^{2^n} \mid n \geq 0\}$ cannot be generated by 1-symbol multimembrane SA P systems. This would also mean 1-symbol multimembrane SA P systems *generators* are not equal to 1-symbol multimembrane SA P systems *acceptors*. Whether or not these statements are true is still open.

2.5 Prioritized 1-*Symbol Multimembrane SA P System Acceptors*

In order to extend the construction given in Theorem 4 to simulate traditional counter machines (hence accepting all recursively enumerable sets of natural numbers), we need to be able to test the counters for zero. The ability to do this without additional rule restrictions is still an open question. However, augmenting 1-symbol multimembrane SA P system acceptors with additional rule restrictions allows universality. Here, we show that allowing the use of priority relations is enough.

A SA P system with priority relations is defined identically to a regular SA P system with the addition of a priority relation associated with all the rules in the system. Each element of the priority relation is defined as a 2-tuple (r_1, r_2) where r_1 and r_2 are rules such that $r_1 > r_2$. This affects the application of the rules by not allowing rule r_2 to be applied unless rule r_1 is no longer applicable.

To show universality, we give a construction which simulates the operation of any counter machine (CM) with m-counters by a 1-symbol $(2m + 3)$-membrane SA P system acceptor with priority. Since CMs are known to accept all recursively enumerable sets of natural numbers, this shows that these systems are able to accept all recursively enumerable sets of natural numbers (obviously the reverse is trivial).

Theorem 8 *1-Symbol $(2m + 3)$-membrane SA P system acceptors with priority relations are able to accept all recursively enumerable sets of natural numbers.*

Proof For this proof, we give a construction to convert any m-counter CM $\mathcal{M}$ into a 1-symbol $(2m + 3)$-membrane SA P system acceptor with priority Π. This construction follows the construction given in the proof of Theorem 4. Therefore, we only give the changes that need to be made to the previous construction.

The primary difference between a CM system and a PBCM system is the operation of the subtract instruction. To correctly simulate the new type of subtraction instruction, we must add additional encoded instructions to deal with testing for zero. This means our encoding for Theorem 4 must be modified. Our new encoding is done as follows.

Again, we first associate some number of distinct natural numbers to each instruction we want encoded. Here, for each instruction (or instruction label), we give our way of denoting each specific associated natural number followed by the natural number itself. Note that each subtraction instruction has 4 distinct natural numbers associated to it.

1. $\forall l_i \in B : l_i = 5i - 4$.
2. $\forall l_i : (\texttt{ADD}(r), l_j) \in R : A_{ri} = 5i - 3$.
3. $\forall l_i : (\texttt{SUB}(r), l_j, l_k) \in R : S_{ir} = 5i - 3$; $S'_{ir} = 5i - 2$; $S''_{ir} = 5i - 1$; $S'''_{ir} = 5i$.

Let $n = 5|B|$ which is the maximum number of instructions that must be encoded. (This case occurs if each instruction is a subtraction instruction. Note some numbers might not be used in this mapping if the instruction set contains addition instructions.) For the halt instruction, we need two distinct natural numbers for each counter (just like the proof of Theorem 4) with the following mapping.

1. $\forall\, i (1 \leq i \leq m) : H_i = 2i - 1 + n$ and $H_{i'} = 2i + n$.

Let $n' = n + 2m + 1$. The encoding of these distinct natural numbers (corresponding to the instructions and instruction labels) is done with the function $v_\alpha(i) = 16mn' + 16mi$.

Previously, to simulate a PBCM, we encoded the commands ADD_i, SUB_i, $CHECK_i$, and $REMOVE_i$ for each counter $(1 \leq i \leq m)$. To simulate a CM, we must also encode two additional commands for each counter. We will call these $ZERO_i$ and $Z - CORR_i$ for $1 \leq i \leq m$. These commands signify checking that a counter contains a zero count and concluding the assumption of a zero counter is correct. The addition of these commands requires our encoding of the commands to also be changed slightly.

These commands are encoded with the function $v_\beta(j)$ defined as follows:

1. $v_\beta(ADD_j) = 32mn' - 6(j-1) - 1$ (adds a number to counter j).
2. $v_\beta(REMOVE_j) = 32mn' - 6(j-1) - 2$ (signifies the instruction is completed).
3. $v_\beta(SUB_j) = 32mn' - 6(j-1) - 3$ (subtracts a number from counter j).
4. $v_\beta(CHECK_j) = 32mn' - 6(j-1) - 4$ (guarantees the counters halt in the appropriate configuration).
5. $v_\beta(Z - CORR_j) = 32mn' - 6(j-1) - 5$ (makes sure counter j contains zero).
6. $v_\beta(ZERO_j) = 32mn' - 6(j-1) - 6$ (signifies our zero counter guess was correct).

The $l_i : (\mathtt{ADD}(r), l_j)$ instructions and $l_i : (\mathtt{HALT})$ instruction create identical SA rules to those created in Theorem 4. However, subtract instructions are simulated differently. Formally, the SA rules needed to implement a subtract instruction follow.

1. For all $l_i = (\mathtt{SUB}(r), l_j, l_k) \in R$, add the following rules to membrane s:
   (a) $(o^{v_\alpha(l_i)}, out; o^{v_\alpha(S_{ir})} o^{v_\beta(SUB_r)}, in)$,
   (b) $(o^{v_\alpha(S_{ir})}, out; o^{v_\alpha(l_j)}, in)$,
   (c) $(o^{v_\alpha(l_i)}, out; o^{v_\alpha(S'_{ir})} o^{v_\beta(ZERO_r)}, in)$,
   (d) $(o^{v_\alpha(S'_{ir})}, out; o^{v_\alpha(S''_{ir})}, in)$,
   (e) $(o^{v_\alpha(S''_{ir})} o^{v_\beta(Z-CORR_r)}, out; o^{v_\alpha(S'''_{ir})}) o^{v_\beta(SUB_r)}, in)$,
   (f) $(o^{v_\alpha(S_{ir})'''}, out; o^{v_\alpha(l_k)}, in)$.
2. Add the following rules to membrane c_{r1}. (This is in addition to the c_{r1} rules used in Theorem 4.):
   (a) $(o^{v_\beta(REM_r)} o, out; o^{v_\beta(ZERO_r)}, in)$,
   (b) $(o^{v_\beta(Z-CORR_r)}, out; o^{v_\beta(ZERO_r)}, in)$.
   Add the associated priority relation 2(a) > 2(b).

Informally, the subtract instruction begins by nondeterministically guessing whether the current counter contains a zero count or a nonzero count. This is done by nondeterministically choosing to execute a rule of type 1(a) or a rule of type 1(c). If a nonzero count in guessed, the encoded *SUB* command for counter r is brought into the system along with the encoded instruction $l_i : (\mathtt{SUB}(r), l_j)$. These objects decrement counter r in the same manner as decrementing a PBCM counter. (If the guess was incorrect—meaning the counter initially contained zero—this process will cause some objects to be trapped, guaranteeing the computation to infinitely loop.) If a zero-count is guessed, the encoded $ZERO_r$ command is drawn into the membrane along with the encoded instruction S'_{ir}. If the guess is correct, the encoded $ZERO_r$ command is swapped with the encoded $Z - CORR_r$ command in membrane c_{r1} (representing membrane r). This will cause counter r to contain a count of 1 (since $ZERO_r = Z - CORR_r + 1$). To complete the instruction, 1 is subtracted from counter r to return it to zero. If the guess was wrong (the counter contained one or more), the encoded $ZERO_r$ command is swapped with the encoding of $REMOVE_r + 1$. In the next step the instruction S''_{ir} will be unable to be removed causing it to become trapped.

The priority relation guarantees that instruction 2(b) is only executed if the counter contains zero. If instruction 2(a) is ever executed, there will be too many objects to correctly execute instruction 1(e). This guarantees that a potentially halting computation can only occur if the counter contains a zero count.

The remainder of the construction directly follows the proof of Theorem 4. The previous construction correctly simulates the $l_i : (\mathtt{ADD}(r), l_j)$ and $l_i : (\mathtt{HALT})$ instructions of a counter machine. Therefore, the entire computation of CM $\mathcal{M}$ can be simulated by the constructed SA P system Π which consists of only a single object, but allowing priorities. □

In Sect. 2.4, it was also shown that the simulation in Theorem 4 for PBCM acceptors could be modified to also simulate PBCM generators. This result can also be applied to the above proof allowing us to simulate a CM generator. The modifications needed are the same as those used in the proof of Theorem 7.

Theorem 9 1-*Symbol* (2m + 3)-*membrane SA P system generators with priority relations are able to accept all recursively enumerable sets of natural numbers.*

Proof This follows from the proof of Theorem 7 and the proof of Theorem 8. □

All recursively enumerable sets of natural numbers can be accepted (and generated) by a 3-CM acceptor (generator). It can also be noted that for each counter we need only one element in the priority relation. Therefore, to simulate a 3-CM, we only need a priority relation of size 3. The following corollary clearly holds.

Corollary 7 1-*Symbol* 9-*membrane SA P system acceptors* (*or generators*) *with* 3 *priority relations are able to accept all recursively enumerable sets of natural numbers.*

Since the proof of Theorem 8 utilizes the same basic structure as the PBCM acceptor/PBCM generator theorems, the maximal parallelism can be bounded in the same manner as before with a maximum of two rules being applied per membrane per step. This leads to the following corollary.

Corollary 8 1-*Symbol* 9-*membrane* 2-*parallel SA P system acceptors* (*or generators*) *with* 3 *priority relations are able to accept all recursively enumerable sets of natural numbers*.

3 Bounded SA P Systems

Initially, membrane systems were designed to be nondeterministic systems. When multiple, maximal sets of rules are applicable, nondeterminism decides which maximal set to apply. Recently, deterministic versions of some membrane models have been studied to determine whether they are as computationally powerful as the nondeterministic versions [6, 12]. Deterministic models guarantee that each step of the computation consists of only one maximal multiset of applicable rules. In some cases, both the nondeterministic and deterministic versions are equivalent in power to Turing Machines (see, e.g., [6]). In some nonuniversal P systems, the deterministic versus the nondeterministic question has been shown to be equivalent to the long-standing open problem of whether deterministic and nondeterministic linear-bounded automata are equivalent [12]; for another very simple class of systems, deterministic systems are strictly weaker than nondeterministic systems [12]. However, these two latter results do not easily translate for SA P systems.

In this section, we look at restricted models of symport/antiport P systems. Two models, called *bounded SA P systems* and *special SA P systems*, are acceptors of multisets with the restriction that the multiplicity of each object in the system does not change during the computation. These models differ in whether they also bound the number of membranes within the system or bound the number of distinct objects that can occur abundantly in the environment. Another model, called *bounded string SA P system*, is an acceptor of string languages. This model has the property that at any time during the computation, the number of objects in the system is equal to the number of input symbols that have been read so far (in addition to a fixed number of objects given to the system at the start of the computation). We study the computing power of these models. In particular, we investigate questions concerning hierarchies (with respect to the number of distinct objects used in the system or number of membranes in the system) and whether determinism is strictly weaker than nondeterminism.

3.1 **1**-*Membrane Bounded SA P Systems*

Let Π be a 1-membrane symport/antiport P system over an alphabet V, and let $\Sigma = \{a_1, \ldots, a_k\} \subseteq V$ be the *input* alphabet. Π is restricted in that all rules are of the form $(u, out; v, in)$, where $u, v \in V^*$ with $|u| = |v| \geq 1$. Thus, the number of objects in the system at any time during the computation remains the same. Note that all the rules are antiport rules.

There is a fixed string (multiset) w in $(V - \Sigma)^*$ such that initially, the system is given a string $wa_1^{n_1} \cdots a_k^{n_k}$ for some nonnegative integers $n_1, \ldots, n_k$ (thus, the input multiset is $a_1^{n_1} \cdots a_k^{n_k}$). If the system halts, then we say that the string $a_1^{n_1} \cdots a_k^{n_k}$ is accepted. The set of all such strings is the language $L(\Pi)$ accepted by Π. We call this system a *bounded SA P system*. Π is *deterministic* if the maximally parallel multiset of rules applicable at each step in the computation is unique. We will show the following:

1. A language $L \subseteq a_1^* \cdots a_k^*$ is accepted by a deterministic (nondeterministic) bounded SA P system if and only if it is accepted by a deterministic (nondeterministic) $\log n$ space-bounded Turing machine (with a two-way read-only input with left and right end markers).
2. For every r, there is an $s > r$ and a unary language L (i.e., $L \subseteq o^*$) accepted by a bounded SA P system with an alphabet of s symbols that cannot be accepted by any bounded SA P system with an alphabet of r symbols. This result holds for both deterministic and nondeterministic versions.
3. Deterministic and nondeterministic bounded SA P systems over a unary input alphabet are equivalent if and only if deterministic and nondeterministic linear-bounded automata (over an arbitrary alphabet) are equivalent. This later problem is a long-standing open problem in complexity theory [20].

The restriction $|u| = |v| \geq 1$ in rule $(u, out; v, in)$ can be relaxed to $|u| \geq |v| \geq 1$, but the latter is equivalent since we can always introduce a dummy symbol d and add $d^{|u|-|v|}$ to v to make the lengths the same and not use symbol d in any rule. We note a similar system, called bounded P system (BPS) with cooperative rules of the form $u \to v$ where $|u| \geq |v| \geq 1$, was also studied in [5] for their model-checking properties.

For ease in exposition, we first consider the case when the input alphabet is unary, i.e., $\Sigma = \{o\}$. Thus, the bounded SA P system Π has initial configuration wo^n (for some n). The idea is to relate the computation of Π to a restricted type of multicounter machine, called *linear-bounded multicounter machine*.

A *deterministic* multicounter machine $\mathcal{M}$ is linear-bounded if, when given an input n in one of its counters (called the input counter) and zeros in the other counters, it computes in such a way that the sum of the values of the counters at any time during the computation is at most n. One can easily normalize the computation so that every increment is preceded by a decrement (i.e., if $\mathcal{M}$ wants to increment a counter C_j, it first decrements some counter C_i and then increments C_j) and every decrement is followed by an increment. Thus, we can assume that every instruction of $\mathcal{M}$, which is not “Halt,” is of the form:

p: If $C_i \neq 0$, decrement C_i by 1, increment C_j by 1, and go to state k else go to state l,

where p, k, l are labels (states). We do not require the contents of the counters are zero when the machine halts.

If in the above instruction, there is a “choice” for states k and/or l, the machine is *nondeterministic*. We will show that we can construct a deterministic (nondeterministic) bounded SA P system Π which uses a fixed multiset w such that when Π is started with multiset wo^n, it simulates $\mathcal{M}$ and has a halting computation if and only if $\mathcal{M}$ halts on input n. Moreover, the rules of Π are of the form $u \to v$, where $|u| = |v| = 1$ or 2.

It is convenient to use an intermediate P system, called SCPS, which is a restricted version of the CPS (communicating P system) introduced in [22]. A SCPS (“S” for simple) is a restricted CPS which has only rules of the form $a \to a_x$ or $ab \to a_x b_y$. Moreover, if the skin membrane has these types of rules, then $x, y \neq out$ (i.e., no objects are transported to the environment).

Lemma 1 *If a language $L \subseteq o^*$ is accepted by a deterministic* (*nondeterministic*) *linear-bounded multicounter machine $\mathcal{M}$, then it is accepted by a deterministic* (*nondeterministic*) *SCPS Π.*

Proof We only prove the case when $\mathcal{M}$ is deterministic, the nondeterministic case being similar. The construction of Π is a simple modification of the construction in [22]. Assume $\mathcal{M}$ has m counters $C_1, \dots, C_m$. Π has the same membrane structure as in [22]. In particular, the skin membrane contains membranes $E_1, \dots, E_m$ to simulate the counters, where the multiplicity of the distinguished (input) symbol o in membrane E_i represents the value of counter C_i. There are other membranes within the skin membrane that are used to simulate the instructions of $\mathcal{M}$ (see [22]). All the sets of rules $R_1, \dots,$ are the same as in [22], except the instruction

p: If $C_i \neq 0$, decrement C_i by 1, increment C_j by 1, and goto l else goto k

of $\mathcal{M}$ is simulated as in [22], but the symbol o is not thrown out (from the skin membrane) into the environment but added to membrane E_j. It follows from the construction in [22] that Π will not have any instruction of the form $ab \to a_x b_y c_{come}$ and if instructions of the form $a \to a_x$ or $ab \to a_x b_y$ appear in the skin membrane, then $x, y \neq out$. Hence, Π is a deterministic SCPS. □

Lemma 2 *If a language $L \subseteq o^*$ is accepted by a deterministic* (*nondeterministic*) *linear-bounded multicounter machine, then it is accepted by a deterministic* (*nondeterministic*) *bounded SA P system.*

Proof We show how to convert the multimembrane SCPS Π of Lemma 1 to a 1-membrane bounded SA P system Π'. The construction is similar to the one given in [5]. Suppose that Π has membranes $1, \dots, m$. For each object a in V, Π' will have symbols $a_1, \dots, a_m$. In particular, for the distinguished input symbol o in V, Π' will have $o_1, \dots, o_m$. Hence, the distinguished input symbol in Π' is o_{i_0}, where i_0 is the index of the input membrane in Π. We can convert Π to a bounded SA P system Π' as follows:

1. If $a \to a_x$ is a rule in membrane i of Π, then $(a_i, out; a_j, in)$ is a rule in Π', where j is the index of the membrane into which a is transported to, as specified by x.

2. If $ab \to a_x a_y$ is a rule in membrane i of Π, then $(a_i b_i, out; a_j b_k, in)$ is a rule in Π', where j and k are the indices of the membranes into which a and b are transported to, as specified by x and y.

Thus, corresponding to the initial configuration wo^n of Π, where o^n is in the input membrane i_0 and w represents the configuration denoting all the other symbols (different from o) in the other membranes, Π' will have initial configuration $w'o^n_{i_0}$, where w' are symbols in w renamed to identify their locations in Π.

Clearly, Π' accepts $o^n_{i_0}$ if and only if Π accepts o^n, and Π' is a deterministic (nondeterministic) bounded SA P system. □

We will prove the converse of Lemma 2 indirectly. A k-head two-way finite automaton (k-2FA) is a finite automaton with k two-way read-only heads operating on an input (with left and right end markers) [17]. A multihead 2FA is a k-2FA for some k.

Lemma 3 *If Π is a deterministic* (*nondeterministic*) *bounded SA P system with an alphabet V of m symbols* (*note that V contains the distinguished input symbol o*), *then Π can be simulated by a deterministic* (*nondeterministic*) *$m(m+1)$-2FA $\mathcal{M}$.*

Proof Suppose Π is a deterministic bounded SA P system accepting a language $L(\Pi) \subseteq o^*$. Assume that its alphabet is $V = \{a_1, \dots, a_m\}$, where $a_1 = o$ (the input symbol). We construct a deterministic multihead FA $\mathcal{M}$ to accept $L(G)$. The input to $\mathcal{M}$ (not including the left and right end markers) is o^n for some n. We will need the following heads to keep track of the multiplicities of the symbols in the membrane during the computation (note that the bounded SA P system Π is given wo^n initially):

1. K_i for $1 \le i \le m$. Head K_i will keep track of the current number of a_i's. Initially, K_1 will point to the right end marker (indicating that there are n o's in the input) while all other K_i will point to the appropriate position on the input corresponding to the multiplicity of symbol a_i in the fixed string w.
2. $K_{i,j}$ for $1 \le i, j \le m$. These heads keep track of how many a_i's are replaced by a_j's during the next step of Π.

One step of Π is simulated by a (possibly unbounded) number of steps of $\mathcal{M}$. At the beginning of the simulation of every step of Π, $\mathcal{M}$ resets all $K_{i,j}$'s to the left end marker. To determine the next configuration of Π, $\mathcal{M}$ processes the rules as follows:

Let $R_1, R_2, \dots, R_s$ be the rules in the membrane. By using $K_1, \dots, K_m$ (note each K_i represents the number of a_i's in the membrane), $\mathcal{M}$ applies rule R_1 sequentially a maximal number of times storing the "results" (i.e., the number of a_i's that are converted by the applications of rule R_1) to a_j in head $K_{i,j}$. Thus, each application of R_1 may involve decrementing the K_i's and incrementing some of the $K_{i,j}$'s. (By definition, the sequential application of R_1 has reached its maximum at some point, if further application of the rule is no longer applicable.)

The process just described is repeated for the other rules $R_2, \ldots, R_s$. When all the rules have been processed, $\mathcal{M}$ updates each head K_j using the values stored in $K_{i,j}$, $1 \leq i \leq m$. This completes the simulation of the unique (because Π is deterministic) maximally parallel step of Π.

It follows from the above description that a deterministic bounded SA P system can be simulated by a deterministic $m(m+1)$-2FA.

If Π is nondeterministic, the construction of the nondeterministic multihead 2FA Π is simpler. Π just sequentially guesses the rule to apply each time (i.e., any of $R_1, R_2, \ldots, R_s$) until no more rule is applicable. Note that $\mathcal{M}$ does not need the heads $K_{i,j}$'s. □

For the proof of the next theorem, we need a definition. Define a *generalized linear-bounded multicounter machine* as follows. As before, at the start of the computation, the input counter is set to a value n (for some n), and all other counters are set to zero. Now, we only require that there is a positive integer c such that at any time during the computation, the value of any counter is at most cn. (Thus, we no longer require that the sum of the values of the counters is at most n.) In [5], it was shown that a generalized linear-bounded multicounter machine can be converted to a linear-bounded multicounter machine. For completeness, we describe the construction.

Suppose that $\mathcal{M}$ is a generalized linear-bounded multicounter machine with counters $C_1, \ldots, C_m$, where C_1 is the input counter. Construct another machine $\mathcal{M}'$ with counters $D, C_1, \ldots, C_m$, where D is now the input counter. $\mathcal{M}'$ with input n in counter D, first moves n from D to C_1 (by decrementing D and incrementing C_1). Then $\mathcal{M}'$ simulates $\mathcal{M}$ on counters $C_1, \ldots, C_m$ (counter D is no longer active).

Let d be any positive integer. We modify $\mathcal{M}'$ to another machine $\mathcal{M}''$ which uses, for each counter C_i, a buffer of size d in its finite control to simulate $\mathcal{M}'$, and $\mathcal{M}''$ increments and decrements each counter modulo d. $\mathcal{M}''$ does not alter the action of $\mathcal{M}'$ on counter D.

By choosing a large enough D, it follows that the computation of $\mathcal{M}''$ is such that when given input n in counter D and zeros in counters $C_1, \ldots, C_m$, the sum of the values of counters $D, C_1, \ldots, C_m$ at any time is at most n. It follows that given a generalized linear-bounded multicounter, we can construct an equivalent linear-bounded multicounter machine.

The next theorem is similar to a result in [5] concerning BPS.

Theorem 10 *Let $L \subseteq o^*$. Then the following statements are equivalent*: (1) *L is accepted by a bounded SA P system*, (2) *L is accepted by a linear-bounded multicounter machine*, (3) *L is accepted by a $\log\ n$ space-bounded Turing machine*, (4) *L is accepted by a multihead 2FA These equivalences hold for both the deterministic and nondeterministic versions.*

Proof The equivalence of (3) and (4) is well known. By Lemmas 2 and 3, we need only show the equivalence of (2) and (4). That a linear-bounded multicounter machine can be simulated by a multihead 2FA is obvious. Thus, (2) implies (4). We now show the converse. Let Π be a two-way multihead FA Π with m heads $H_1, \ldots, H_m$. From the discussion above, it is sufficient to construct a generalized multicounter machine $\mathcal{M}$ equivalent to Π. $\mathcal{M}$ has $2m + 1$ counters, $D, C_1, \ldots, C_m, E_1, \ldots, E_m$. $\mathcal{M}$ with input n in counter D, and zero in the other counters first decrements D and stores n in counters $C_1, \ldots, C_m$. Then $\mathcal{M}$ simulates the actions of head H_i of Π using the counters C_i and E_i. □

Lemmas 2 and 3 and Theorem 10 can be generalized to nonunary inputs, i.e., inputs of the form $a_1^{n_1} \cdots a_k^{n_k}$, where $a_1, \ldots, a_k$ are distinct symbols. The constructions are straightforward generalizations of the ideas above. Thus, we have the following corollary.

Corollary 9 *Let $L \subseteq a_1^* \cdots a_k^*$. Then the following statements are equivalent*: (1) *L is accepted by a bounded SA P system*, (2) *L is accepted by a linear-bounded multicounter machine*, (3) *L is accepted by a $\log n$ space-bounded Turing machine*, (4) *L is accepted by a multihead* 2FA. *These equivalences hold for both the deterministic and nondeterministic versions.*

We now proceed to show that the number of symbols in the alphabet V of a bounded SA P system induces an infinite hierarchy. This is an interesting contrast to a result in [19] stating an unbounded SA P system with three objects is universal. The proof follows the ideas in [11], which showed an infinite hierarchy for a variant of SCPS, called RCPS.

We will need the following result from [17].

Theorem 11 *For every k, there is a unary language L that can be accepted by a $(k+1)$-2FA but not by any k-2FA. The result holds for both deterministic and nondeterministic versions.*

Theorem 12 *For every r, there exist an $s > r$ and a unary language L (i.e., $L \subseteq o^*$) accepted by a bounded SA P system with an alphabet of s symbols that cannot be accepted by any bounded SA P system with an alphabet of r symbols. This result holds for both deterministic and nondeterministic versions.*

Proof Suppose there is an r such that any unary language accepted by any bounded SA P system with an arbitrary alphabet can be accepted by a bounded SA P system with an alphabet of r symbols. Let $k = r(r+1)$. From Theorem 11, there is a unary language L that can be accepted by a $(k+1)$-2FA but not by any k-2FA. By Theorem 10, this language can be accepted by a bounded SA P system. Then by hypothesis, L can also be accepted by a bounded SA P system with an alphabet of r symbols. Then from Lemma 3, we can construct from this bounded SA P system an $r(r+1)$-2FA accepting L. Hence, L can be accepted by a k-2FA, a contradiction. □

For our next result, we need the following theorem from [21].

Theorem 13 *Nondeterministic and deterministic multihead 2FAs over a unary input alphabet are equivalent if and only if nondeterministic and deterministic linear bounded automata* (*over an arbitrary input alphabet*) *are equivalent.*

From Theorems 10 and 13, we have:

Theorem 14 *Nondeterministic and deterministic bounded SA P systems over a unary input alphabet are equivalent if and only if nondeterministic and deterministic linear bounded automata* (*over an arbitrary input alphabet*) *are equivalent.*

3.2 Multimembrane Special SA P Systems

Let Π be a multimembrane SA P system, which is restricted in that only rules of the form $(u, out; v, in)$, where $|u| = |v| \geq 1$, can appear in the skin membrane. There are no restrictions on the weights of the rules in the other membranes. Clearly, the number of objects in the system at any time during the computation remains the same. We denote by E_t the alphabet of t symbols (for some t) in the environment. There may be other symbols in the membranes that remain in the system during the computation and are not transported to/from the environment, and they are not part of E_t. Note that E_0 means that the environment alphabet is empty (i.e., there are no symbols in the environment at any time). As before, we consider the case where the input alphabet is unary (i.e., $\Sigma = \{o\}$). Π's initial configuration contains o^n in the input membrane (for some n) and a *fixed* distribution of some *non-o* symbols in the membranes. The string o^n is accepted if the system eventually halts. We call the system just described a *special SA P system*.

Theorem 15 *Let $L \subseteq o^*$. Then the following statements are equivalent*: (1) *L is accepted by a multimembrane special SA P system with* **no** *symbols in the environment, i.e., has environment alphabet E_0* (*= empty set*), (2) *L is accepted by a bounded SA P system*, (3) *L is accepted by a linear-bounded multicounter machine*, (4) *L is accepted by a $\log n$ space-bounded Turing machine*, (5) *L is accepted by a multihead 2FA. These equivalences hold for both the deterministic and nondeterministic versions.*

Proof As in Lemma 3, it is easy to show a deterministic (nondeterministic) m-membrane special SA P system with no symbols in the environment can be simulated by a deterministic (nondeterministic) two-way FA with $2m$ heads.

By Theorem 10, to complete the proof, we need only show that a linear-space bounded multicounter machine can be simulated by a multimembrane special SA P system with no symbols in the environment. For notational convenience, we will assume the multicounter machine is controlled by a program with instructions of the type $l_i : (\texttt{ADD}(r), l_j)$, $l_i : (\texttt{SUB}(r), l_j, l_k)$, and $l_i : (\texttt{HALT})$ where l_i is the label for the current instruction being executed and r is the counter which is either being incremented or decremented. If the current instruction is an add instruction, the next instruction to execute will be l_j. If the current instruction is a subtract instruction, the next instruction depends on the value of r. If $r \neq 0$, the next instruction is denoted by l_j otherwise the next instruction is denoted by l_k.

The special SA P system simulating a linear-space bounded multicounter machine will use one membrane to simulate each counter of the multicounter machine. These membranes will be placed within a "program" membrane where the current instruction is brought in, implemented, and then expelled. This entire system is enclosed within a dummy membrane (the skin membrane) containing no rules and a single copy of each instruction object along with a few auxiliary objects. So, the overall system uses $m + 2$ membranes. Obviously, if the skin membrane of the special SA P system contains no rules, no object can ever be brought into the system or expelled from the system. Hence, since the system initially contains $|wo^n|$ symbols, the system will continue to contain $|wo^n|$ symbols after each step of the computation.

To show how any linear-space bounded multicounter machine can be simulated, we give a formal transformation to a special SA P system. Our transformation is similar to the transformation in [19] except that our transformation yields a deterministic (nondeterministic) special SA P system if the original linear-space bounded multicounter machine is deterministic (nondeterministic). (The transformation in [19] only produces a nondeterministic SA P system.) The transformation is done as follows. Consider a multicounter machine $\mathcal{M}$ with m counters. Construct a SA P system Π which simulates $\mathcal{M}$ as follows: $\Pi = \langle V, H, \mu, w_1, w_2, \ldots, w_{m+2}, E_0, R_1, R_2, \ldots, R_{m+2}, i_o \rangle$ where $H = \{1, 2, \ldots, m+2\}$; $\mu = [_1[_2[_3]_3[_4]_4 \ldots [_{m+2}]_{m+2}]_2]_1$; $w_1 =$ one copy of each element in V except o and l_{01} (we assume $\mathcal{M}$'s program begins with the instruction l_0); $w_2 = l_{01}$; $w_3 = o^n$; $w_i = \lambda$, for all $i = 4, \ldots, m+2$; $E_0 = \emptyset$ (the environment, E_t, is empty because $t = 0$); No need to specify i_0, since our system is an acceptor.

The elements of V are as follows:

1. o—The symbol o is used as the counting object for the system. The multiplicity of o's in each counter membrane signifies the count of that counter.
2. $d_1, d_2, d_3, d_4, d_5, d_6$—These objects are used to delay various objects from being used for a number of steps. The objects d_1 and d_2 are used to delay an action for 1 step. The remaining objects are used to delay an action for 3 steps.
3. c_1, c_2, c_3—These objects are called check objects and are used to guarantee a subtract instruction expels at most one o object from the appropriate counter membrane.
4. l_{i1}, l_{i2} for each instruction $l_i : (\mathtt{ADD}(r), l_j)$. The object l_{i1} signifies the instruction executed next is l_i. The object l_{i2} is used in executing instruction l_i.
5. $l_{i1}, l_{i2}, l_{i3}, l_{i4}$ for each instruction $l_i : (\mathtt{SUB}(r), l_j, l_k)$. The object l_{i1} signifies that the next instruction we will execute is l_i. The objects l_{i2}, l_{i3}, and l_{i4} are used in executing instruction l_i and are used to signify which branch of l_i will determine the next instruction.
6. l_{i1} for each instruction $l_i : (\mathtt{HALT})$.

The sets of rules for the R_i's are created as follows:

1. $R_1 = \emptyset$.
2. R_2 contains the rules $(d_1, out; d_2, in)$, $(d_3, out; d_4, in)$, $(d_4, out; d_5, in)$, $(d_5, out; d_6, in)$ referred to as delay rules.
3. For each instruction $l_i : (\texttt{ADD}(r), l_j)$ in $\mathcal{M}$:
   - R_2 contains the rules $(l_{i1}, out; l_{i2}d_1o, in)$, $(l_{i2}d_2, out; l_{j1}, in)$.
   - R_{r+2} contains the rule $(l_{i2}o, in)$, (l_{i2}, out).
4. For each instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ in $\mathcal{M}$:
   - R_2 contains the rules $(l_{i1}, out; l_{i2}c_1d_3, in)$, $(l_{i2}o, out; c_2l_{i3}, in)$, $(c_1c_2d_6l_{i3}, out; l_{j1}, in)$, $(l_{i2}d_6, out; c_3l_{i4}, in)$, $(c_1c_3l_{i4}, out; l_{k1}, in)$.
   - R_{r+2} contains the rules $(l_{i2}c_1, in)$, $(l_{i2}o, out)$, $(c_1, out; c_2, in)$, (c_2, out), $(l_{i2}, out; d_6, in)$, (d_6, out), $(c_1, out; c_3, in)$, (c_3, out).
5. For each instruction $l_i : (\texttt{HALT})$ no rules are added.

Informally, these special SA P system rules work using the following ideas. Initially, the system is started with the first instruction label object l_{01} in the program membrane and the input o^n within membrane 3 (corresponding to counter 1). To execute an add instruction, the initial instruction object is replaced with the objects needed to execute the instruction—l_{i2}, d_1, and o. If the instruction is a subtract instruction the instruction l_{i1} is replaced with l_{i2} along with a delay object d_3 and a check object c_1. Once the appropriate objects are in the program membrane, a o object is appropriately moved into or out of the counter membrane corresponding to the current instruction. In the case where the current instruction tries to decrement a zero counter, the check objects cooperate with the delay objects to detect the situation and bring the appropriate objects into and out of the active membranes. Finally, the instruction executing objects are expelled from the program membrane and the correct next instruction object is brought into the program membrane.

Note that when a counter is decremented, an o object is removed from the corresponding membrane and moved into the skin membrane. When a counter is incremented, an o object is brought into the corresponding membrane from the skin membrane. Since the multicounter machine being simulated is, by definition, guaranteed to always decrement before incrementing, we are guaranteed to have thrown an o object into membrane 1 before we ever try bringing an o object from membrane 1 to membrane 2. This guarantees that the special SA P system will operate through the multicounter machine's program instructions correctly. □

Corollary 10 *Let t be any positive integer. Then multimembrane special SA P systems with an environment alphabet of t symbols are equivalent to multimembrane special SA P systems with no symbols in the environment. This holds for deterministic and nondeterministic versions.*

Proof This follows from the above theorem and the observation that a system with an environment of t symbols can be simulated by a two-way FA with $2m(t + 1)$ heads. □

The proof of the next result is similar to that of Theorem 12.

Theorem 16 *For every r, there exist an $s > r$ and a unary language L (i.e., subset of o^*) accepted by an s-membrane special SA P system that cannot be accepted by any r-membrane special SA P system. This result holds for both deterministic and nondeterministic versions.*

3.3 1-*Membrane Bounded SA P Systems Which Accept String Languages*

Let Π be a 1-membrane SA P system with alphabet V and input alphabet $\Sigma \subseteq V$. Assume Σ contains a distinguished symbol \$, called the (right) end marker. The rules are restricted to the form $(u, out; v, in)$ or $(u, out; vc, in)$ where u is in V^+, v is in $(V - \Sigma)^+$, $|u| = |v| \geq 1$, and c is in Σ. Thus, the system can export any symbol in V to the environment, but can only import symbols in Σ from the environment via rules of type 2 (referred to as *read-rules*). We call Π a *bounded string SA P system*. There is an abundance of symbols from $V - \Sigma$ in the environment. The only symbols from Σ available in the environment are in the input string $z = a_1 \cdots a_n$ (where a_i is in $\Sigma - \{\$\}$ for $1 \leq i < n$, and $a_n = \$$), which is provided online externally.

There is a fixed string w in $(V - \Sigma)^*$, which is the initial configuration of Π. Maximal parallelism in the application of the rules is assumed as usual. Hence, in general, the size of the multiset of rules applicable at each step is unbounded. In particular, the number of instances of read-rules (i.e., rules of the form $(u, out; vc, in)$) applicable in a step is unbounded. However, if a step calls for reading k input symbols (for some k), these symbols must be consistent with the next k symbols of the input string z that have not yet been processed. Note that rules of type 1 do not consume any input symbol from z.

The input string $z = a_1 \cdots a_n$ (with $a_n = \$$) is accepted if, after reading all the input symbols, Π eventually halts. The language accepted is $L(\Pi) = \{a_1 \cdots a_{n-1} \mid a_1 \cdots a_n$ is accepted by $\Pi\}$ (we do not include the end marker).

We have two versions of the system described above: deterministic and nondeterministic bounded string SA P systems. Again, in the deterministic case, the maximally parallel multiset of rules applicable at each step of the computation is unique. We will show the deterministic version is strictly weaker than the nondeterministic version. The proof uses some recent results in [12] concerning a simple model of a CPS, called SCPA.

An SCPA Π has multiple membranes, with the skin membrane labeled 1. The symbols in the initial configuration (distributed in the membranes) are *not* from Σ (the input alphabet). The rules (similar to those of a CPS) are of the form: (1) $a \to a_x$, (2) $ab \to a_x b_y$, (3) $ab \to a_x b_y c_{come}$. The input to Π is a string $z = a_1 \cdots a_n$ (with $a_n = \$$, the end marker), provided externally online. The restrictions on the operation of Π are the following:

1. There are no rules in membrane 1 with a_{out} or b_{out} on the right-hand side of the rule (i.e., no symbol can be expelled from membrane 1 into the environment).
2. A rule of type 3 (called a read-rule) can only appear in membrane 1. This brings in c if the next symbol in the input string $z = a_1 \cdots a_n$ that has not yet been processed (read) is c; otherwise, the rule is not applicable.
3. Again, in general, the size of the maximally parallel multiset of rules applicable at each step is unbounded. In particular, the number of instances of read-rules (i.e., rules of the form $ab \to a_x b_x c_{come}$) applicable in a step is unbounded. However, if a step calls for reading k input symbols (for some k), these symbols must be consistent with the next k symbols of the input string z that have not yet been processed (by the semantics of the read-rule described in the previous item).

The system starts with an initial configuration of symbols from $V - \Sigma$ distributed in the membranes. The input string $z = a_1 \cdots a_n$ is accepted if, after reading all the input symbols, the SCPA eventually halts. The language accepted by Π is $L(\Pi) = \{a_1 \cdots a_{n-1} \mid a_1 \cdots a_n$ is accepted by $\Pi\}$ (we do not include the end marker).

A *restricted* 1*-way linear-space DCM* (*NCM*) Π is a deterministic (nondeterministic) finite automaton with a one-way read-only input tape with right delimiter (end marker) \$ and a number of counters. As usual, each counter can be tested for zero and can be incremented/decremented by 1 or unchanged. The counters are restricted in that there is a positive integer c such that at any time during the computation, the amount of space used in any counter (i.e., the count) is at most ck, where k is the number of symbols of the input that have been read so far. Note that the machine need not read an input symbol at every step. An input $w = a_1 \cdots a_n$ (where a_n is the end marker, \$, which only occurs at the end) is accepted if, when Π is started in its initial state with all counters zero, it eventually enters an accepting state while on \$.

We note that although the machines are restricted, they can accept fairly complex languages. For example, $\{a^n b^n c^n \mid n \geq 1\}$ and $\{a^{2^n} \mid n \geq 0\}$ can both be accepted by restricted 1-way linear-space DCMs. (We usually do not include the end marker, which is part of the input, when we talk about strings/languages accepted.) It can be shown that a restricted 1-way linear-space DCM (NCM) is equivalent to a restricted 1-way $\log n$-space deterministic (nondeterministic) Turing machine that was studied in [3].

We will need the following result that was recently shown in [12].

Theorem 17 *A language L is accepted by a restricted* 1*-way linear-space DCM* (*NCM*) *if and only if it is accepted by a deterministic SCPA* (*nondeterministic SCPA*).

Theorem 18 *Deterministic* (*nondeterministic*) *bounded string SA P systems are equivalent to deterministic* (*nondeterministic*) *SCPAs.*

Proof First we show that a deterministic (nondeterministic) SCPA Π can be simulated by a deterministic (nondeterministic) bounded string SA P system Π', which has only one membrane. Suppose Π has membranes $1, \ldots, m$, with index 1 representing the skin membrane. For every symbol a in the system and membrane i, create a new symbol a_i. We construct Π' by converting the rules to 1-membrane rules as described in the proof of Lemma 2, except that now we have to handle rules of the form $ab \to a_x b_y c_{come}$ in membrane 1. We transform such a rule to $(a_1 b_1, out; a_j b_k c_1, in)$, where j and k are the indices of the membranes into which a and b are transported to, as specified by x and y. After we have constructed Π', modify it slightly by deleting the subscripts of all symbols with subscript 1 (in the rules and initial configuration). Thus, unsubscripted symbols are associated with symbols in membrane 1 of the SCPA Π.

For the converse, we need only show (by Theorem 17) that a deterministic (nondeterministic) bounded string SA P system Π can be simulated by a restricted 1-way linear-space DCM (NCM) $\mathcal{M}$. The construction of $\mathcal{M}$ is similar to Lemma 3, except $\mathcal{M}$ uses counters (instead of heads), and in the maximally parallel step, the read-rules are the first ones to be processed. Define an atomic read-rule process as follows: $\mathcal{M}$ systematically cycles through the read-rules and finds (if it exists) the first one that is applicable (note that for a read-rule $(u, out; vc, in)$ to be applicable, the next input symbol to be processed must be c). $\mathcal{M}$ applies a sequence of these read-rules until no additional read-rule is applicable. Then all the other rules are processed. We omit the details. If Π is a nondeterministic SCPA, the construction of a nondeterministic $\mathcal{M}$ is similar, in fact, easier. □

From Theorem 17 and the fact that deterministic SCPAs are strictly weaker than nondeterministic SCPAs [12], we have the following theorem.

Theorem 19 *Deterministic bounded string SA P systems are strictly weaker than nondeterministic bounded string SA P systems.*

Let $L = \{x\#^p \mid x$ is a binary number with leading bit 1 and $p \neq 2\,val(x)\}$, where $val(x)$ is the value of x. It was shown in [12] that L can be accepted by a nondeterministic SCPA but not by any deterministic SCPA. Hence, L is an example of a language that can be accepted by a nondeterministic bounded string SA P system that cannot be accepted by any deterministic bounded string SA P system. The following follows from Theorem 18 and the fact that similar results hold for SCPAs [12].

Theorem 20 *Let NBSA* (*DBSA*) *be the class of languages accepted by nondeterministic* (*deterministic*) *bounded string SA P systems. Then*: (1) *NBSA is closed under union and intersection but not under complementation*, (2) *DBSA is closed under union*, *intersection*, *and complementation.*

4 SA P Systems and Semilinear Sets

A general problem of clear interest in the area of membrane computing or P systems is to find classes of nonuniversal P systems that correspond to (i.e., characterize) known families of languages or subsets of $\mathbb{N}^k$ (where $\mathbb{N}$ is the set of nonnegative integers, and k is a positive integer) and to investigate their closure and decidability properties. For example, P system characterizations of ET0L, bounded languages accepted by multihead finite automata, and context-sensitive languages are known (see, e.g., [3, 11, 14, 15]). Here, we give characterizations of semilinear sets in terms of restricted models of SA P systems.

We introduce some restricted models of SA P systems that are used as acceptors (respectively, generators) of sets of tuples of nonnegative integers and show that they characterize precisely the semilinear sets. Specifically, we prove that a set $R \subseteq \mathbb{N}^k$ is accepted (respectively, generated) by a restricted system if and only if R is a semilinear set. We also show that "slight" extensions of the models will allow them to accept (respectively, generate) nonsemilinear sets. In fact, for these extensions, the emptiness problem is undecidable.

4.1 Simple SA P Systems

We first introduce a restricted model of a SA P system [18] which is used as an acceptor of tuples of nonnegative integers. A *simple SA P system* Π is defined as follows:

1. The alphabet of objects is $V = F \cup \{o\}$, where F is a finite set and o is a distinguished object.
2. There are $k + 1$ membranes ($k \geq 1$) arranged in a 2-level structure: membranes $m_1, m_2, \ldots, m_k$ (the *input membranes*) are at the same level and enclosed in membrane m_{k+1} (the *skin membrane*).
3. At the start of the computation, the k input membranes are given the strings $o^{n_1}, \ldots, o^{n_k}$, respectively, for some nonnegative integers $n_1, \ldots, n_k$ (the skin membrane initially does not contain any o).
4. Also, at the start of the computation, there are fixed strings, i.e., multisets $w_1, \ldots, w_{k+1} \in F^*$ in membranes $m_1, \ldots, m_{k+1}$, respectively. Thus, the w_i's do not contain any o.
5. The environment initially contains a fixed (finite) multiset over F. Of course, symbols exported to the environment from the skin membrane during computation can be retrieved from the environment.
6. Each membrane has a set R_i of rules (some may be empty) The rules are of the form: (u, out), (v, in), or $(u, out; v, in)$ where $u, v \in V^+$. A rule of type (a) transports multiset u from the membrane containing the rule into the surrounding membrane (if the membrane contains u). Rule of type (b) imports multiset v from the surrounding membrane into the membrane containing the rule (if the surrounding membrane contains v). Rule of type (c) simultaneously transports u to the surrounding membrane and imports v from the surrounding membrane (if the membrane contains u and the surrounding membrane contains v).

   The restriction is: In the rules of types (b) and (c), v *does not* contain o's. This means the number of o's in any membrane can only be decreased and cannot be increased.

7. As usual in a P system, the rules are applied in a nondeterministic maximally parallel manner.

Notice that the fixed multisets over F given initially in the membranes as well as in the environment are part of the specification of the simple SA P system Π (which we do not always explicitly state). We say that a tuple $(n_1, \dots, n_k)$ is accepted by Π if, when the k input membranes are given $o^{n_1}, \dots, o^{n_k}$, respectively, the system halts (i.e., none of the rules is applicable). The set of all such tuples is denoted by $R(\Pi)$.

Simple SA P systems are intimately related to counter machines and we will compare them here to variations of reversal-bounded CMs. A *reversal-bounded counter machine* is a counter machine which restricts the number of times a counter changes (reverses) from a nondecreasing mode to a nonincreasing mode and vice versa during the course of a computation. Other than the bound on the number of reversals, the machine operates identically to a standard CM. We can incorporate the reversals into the finite control which will cause the machine to halt and reject if a counter ever tries to make more than k reversals.

We define here a special case of reversal-bounded CMs as a counter machine with only k counters (the input counters) each of whose counters can only be decremented. Moreover, at every step, the machine decrements exactly one counter. We call this machine a *decreasing counter machine*.

We can augment a reversal-bounded multicounter machine with an unrestricted counter, i.e., a free counter. This counter can make an unbounded number of reversals. We call such a machine a *reversal-bounded counter machine with a free-counter*.

Theorem 21 *Let $R \subseteq \mathbb{N}^k$. Then the following statements are equivalent*: (1) *R is a semilinear set*, (2) *R is accepted by a reversal-bounded counter machine with a free counter*, (3) *R is accepted by a reversal-bounded counter machine*, *and* (4) *R is accepted by a decreasing counter machine.*

Proof It is obvious that (4) implies (3) and (3) implies (2). From the definition of a semilinear set, it is easy to construct, given a semilinear set R, a decreasing counter machine $\mathcal{M}$ accepting R. Since $\mathcal{M}$ is nondeterministic, it sufficient to describe the construction of $\mathcal{M}$ when R is a linear set. So, suppose, $R = \{v \mid v = v_0 + m_1 v_1 + \cdots + m_t v_t,\ m_i \in \mathbb{N}^1\} \subseteq \mathbb{N}^k$, with $v_i = (v_{i1}, \dots, v_{ik})$ for $0 \leq i \leq t$. $\mathcal{M}$, when given $(n_1, \dots, n_k)$ in its counters, applies the constant vector v_0 to decrement the counters simultaneously by $v_{01}, \dots, v_{0k}$, respectively. Then $\mathcal{M}$ nondeterministically guesses the number of times m_i to apply v_i to the counters, again, decreasing the counters simultaneously by the amounts $m_i v_{i1}, \dots, m_i v_{ik}$, respectively, for $1 \leq i \leq t$. If all the counters become zero at the same time, $\mathcal{M}$ accepts. Thus, (1) implies (4). That (2) implies (1) is a trivial consequence of a result in [10], which showed that if a bounded language $L \subseteq a_1^* \cdots a_k^*$ (where $a_1, \dots, a_k$ are distinct symbols and $n_1, \dots, n_k$ are nonnegative integers) is accepted by a nondeterministic finite automaton augmented with reversal-bounded counters and one unrestricted counter, then the set $\{(n_1, \dots, n_k) \mid a_1^{n_1} \cdots a_k^{n_k} \in L\}$ is semilinear. □

Lemma 4 *Let Π be a simple SA P system. Then $R(\Pi)$ can be accepted by a reversal-bounded counter machine with a free counter $\mathcal{M}$.*

Proof We construct a counter machine $\mathcal{M}$ with $k+1$ counters to simulate Π. The intuitive idea behind the simulation is the following. The first k counters are reversal-bounded (the input counters) and the last is the free counter. Initially, the input counters are set to $n_1, \ldots, n_k$, respectively. The free counter will keep track of the current number of o's in the skin membrane (at the start, there is none). The initial configuration $(w_1, \ldots, w_k, w_{k+1})$ and the rules $(R_1, \ldots, R_{k+1})$ are stored in the finite-state control of $\mathcal{M}$. The finite-state control keeps track of the numbers of non-o symbols and their distributions within the membranes and the environment (this can be done since their total multiplicities remain the same (as ones initially given as fixed constants in the definition of Π) at any time, independent of the n_i's). $\mathcal{M}$ simulates each nondeterministic maximally parallel step of Π by several moves. Clearly, because of the restrictions on the rules, the counters keeping track of the multiplicities of o's in the input membranes are only decremented. Special care has to be taken when simulating a rule of type either (u, out) or $(u, out; v, in)$ when u contains multiple copies of o's. In order to tell whether such a rule is applicable or not, for each membrane we associate a finite buffer of size d (where d is the maximum number of o's that can be thrown out by a single rule in the membrane) to the finite control of M to keep track of the first d o's in the membrane while using the counter of M associated with the membrane to hold the number of the remaining o's. By doing so, checking whether the above rule is applicable can be done by examining the contents of the finite buffer associated with the membrane where the rule resides.

Now, in a maximally parallel step, some (possibly all) of the input membranes can transport o's to the skin membrane and the skin membrane itself can also transport some o's to the environment. However, the total number of o's transferred from the input membranes to the skin membrane and the total number of o's transferred from the skin membrane to the environment may have no relationship, so the free counter may be decremented and incremented an unbounded number of times during the computation. This is the reason why we need a free counter. It follows from the description that $\mathcal{M}$ can simulate the computation of Π. □

We now prove the converse of Lemma 4.

Lemma 5 *Let $\mathcal{M}$ be a reversal-bounded counter machine with a free counter. Then $R(\mathcal{M})$ can be accepted by a simple SA P system Π.*

Proof By the proof of Theorem 21, we may assume that $\mathcal{M}$ is a decreasing counter machine with k counters accepting $R(\mathcal{M}) \subseteq \mathbb{N}^k$. Thus $\mathcal{M}$, when started in its initial state with $n_1, \ldots, n_k$ in the counters halts in an accepting state if $(n_1, \ldots, n_k)$ is in $R(\mathcal{M})$. Moreover, at each step of the computation, before it halts, $\mathcal{M}$ decrements exactly one counter (there are no increments).

We will construct a simple SA P system Π simulating $\mathcal{M}$. As defined, Π will have a 2-level structure with k input membranes $m_1, \ldots, m_k$ (at the same level) enclosed by the skin membrane m_{k+1}. The k input membranes will keep track of the values of the counters. The construction of Π follows the construction in [19] where a two-level SA P system is shown to simulate a multicounter machine. In the construction, each of the inner membranes represents a counter and the multiplicity of the distinguished symbol o within each membrane represents the value of that counter. The rules associated with each subtract instruction in the construction adhere to the restrictions required by a simple SA P system. Since $\mathcal{M}$ has no increment instructions, the associated Π, by the construction in [19], is a simple SA P system. We omit the details. □

From Theorem 21 and Lemmas 4 and 5, we have the following theorem.

Theorem 22 *Let $R \subseteq \mathbb{N}^k$. Then the following statements are equivalent*: (1) *R is a semilinear set*, (2) *R is accepted by a reversal-bounded counter machine with a free counter*, (3) *R is accepted by a reversal-bounded counter machine*, (4) *R is accepted by a decreasing counter machine*, (5) *R is accepted by a simple SA P system.*

Note that in a simple SA P system, the number of o's in the membranes cannot be increased, since in the rules of the form (v, in) and $(u, out; v, in)$, we do not allow v to contain o's. We can generalize the model. The environment can have an infinite supply of o's, and in the rules of the forms (v, in) and $(u, out; v, in)$ in the skin membrane, v is in F^+o^*. Thus, v can contain o's but must contain at least one symbol in F. (We do not allow v to only contain o's since, otherwise, the system will not halt due to an infinite supply of o's in the environment.) Thus, the number of o's in the skin membrane can increase during the computation by importing o's from the environment. Call this model *simple$^+$ SA P system*. Clearly, the construction in Lemma 4 still works when Π is a simple$^+$ SA P system. The only modification is that in the simulation of a maximally parallel step of Π by $\mathcal{M}$, we also need to consider the o's that may be brought into the skin membrane from the environment by the (v, in) and $(u, out; v, in)$ rules. Thus, we have the following corollary.

Corollary 11 *Let $R \subseteq \mathbb{N}^k$. Then the following statements are equivalent*: *items* (1), (2), (3), (4), (5) *of Theorem* 22, *and* (6): *R is accepted by a simple$^+$ SA P system.*

The following corollary follows from known results concerning semilinear sets.

Corollary 12 *Let k be any positive integer. Then*:

1. *The class of subsets of $\mathbb{N}^k$ accepted by simple SA P systems is closed under union, intersection, and complementation.*
2. *The membership, disjointness, containment, and equivalence problems for simple SA P systems accepting subsets of $\mathbb{N}^k$ are decidable.*

4.2 Cascade SA P Systems

In this section, we show Theorem 22 does not generalize to the case when the simple SA P system has a 3-level structure. In particular, consider a simple SA P system with only three membranes m_1, m_2, m_3, where membrane m_1 is enclosed in m_2, and m_2 is enclosed in m_3 (the skin membrane). Initially, membrane m_1 contains the input o^n. The same restriction (i.e., in the rules of the forms (v, in) and $(u, out; v, in)$, v does contain o's) applies. We show such a system can accept a nonsemilinear set. In fact, the emptiness problem for such systems is undecidable. To facilitate the proofs, we first introduce the notion of cascade counter machines.

Cascade Counter Machines. A *k-counter cascade machine* $\mathcal{M}$ is a finite-state machine with k counters, $c_1, \ldots, c_k$. The instructions of $\mathcal{M}$ are of the following forms:

$s \rightarrow (s', c_i := c_i - 1; c_{i+1} := c_{i+1} + 1)$ (decrement c_i then increment c_{i+1}),
$s \rightarrow (s'$ if c_i is zero else $s'')$ (test if $c_i = 0$),
$s \rightarrow (s', c_k := c_k - 1)$ (counter c_k can be independently decremented).

Notice that in the above, it is implicit that $\mathcal{M}$ cannot increment c_1 (there is no such instruction). We say that a nonnegative integer n is accepted if $\mathcal{M}$, when started in its initial state with counter values $(n, 0, \ldots, 0)$ eventually enters an accepting state.

We first show the emptiness problem for deterministic 3-counter cascade machines is undecidable by showing a 3-counter cascade machine with initial counter values $(n, 0, 0)$ can simulate the computation of a deterministic (unrestricted) 2-counter machine with initial counter values $(0, 0)$. The former accepts some n if and only if the latter halts. The result then follows from the undecidability of the halting problem for 2-counter machines [16].

So, suppose that $\mathcal{M}$ is a deterministic (unrestricted) 2-counter machine. We show that $\mathcal{M}$ can be simulated by a deterministic 3-counter cascade machine $\mathcal{M}'$ with counters c_1, c_2, c_3. The two counters x_1 and x_2 of $\mathcal{M}$ are simulated by c_2 and c_3 of $\mathcal{M}'$, respectively. Clearly, testing if counter x_i is zero for $i = 1, 2$ can be directly simulated in $\mathcal{M}'$. Incrementing/decrementing counters x_1 and x_2 of $\mathcal{M}$ can also be simulated in $\mathcal{M}'$:

1. When $\mathcal{M}$ increments x_1, $\mathcal{M}'$ performs the following: Decrement c_1, increment c_2.
2. When $\mathcal{M}$ increments x_2, $\mathcal{M}'$ performs the following: Decrement c_1, increment c_2, decrement c_2, increment c_3.
3. When $\mathcal{M}$ decrements x_1, $\mathcal{M}'$ performs the following: Decrement c_2, increment c_3, decrement c_3.
4. When $\mathcal{M}$ decrements x_2, $\mathcal{M}'$ also decrements c_3.

During the simulation, if c_1 is zero when an instruction being simulated calls for decrementing c_1, $\mathcal{M}'$ rejects. Note all state transitions in $\mathcal{M}$ are simulated faithfully by $\mathcal{M}'$. It follows we can construct $\mathcal{M}'$ so it accepts the input n (initially given in c_1) if and only if n is "big" enough to allow the simulation of $\mathcal{M}$ to complete. If $\mathcal{M}$ does not halt or n is not big enough to carry out the simulation (at some point),

$\mathcal{M}'$ goes into an infinite loop or rejects. It follows that the emptiness problem for deterministic 3-counter cascade machines is undecidable.

Example We now give an example of a deterministic 3-counter cascade machine $\mathcal{M}$ accepting a nonsemilinear set. Starting with $c_1 = n, c_2 = 0, c_3 = 0$,

1. If c_1 is zero, $\mathcal{M}$ rejects.
2. $\mathcal{M}$ configures the counters to contain: $c_1 = n - 1, c_2 = 0, c_3 = 1$.
3. If c_1 is zero, $\mathcal{M}$ accepts.
4. Set $k = 1$.
5. Starting with values: $c_1 = n - (1 + 3 + \cdots + (2k - 1)), c_2 = 0, c_3 = (2k - 1)$, $(*)$ $\mathcal{M}$ iteratively decrements c_3 by 1 while decrementing c_1 by 1 and incrementing c_2 by 1 until $c_3 = 0$. Then $\mathcal{M}$ decrements c_1 by 2 and increments c_2 by 2 (this is done in two steps). After that, $\mathcal{M}$ iteratively decrements c_2 by 1 while incrementing c_3 by 1 until $c_2 = 0$.

   – If c_1 becomes zero before the completion of $(*)$, $\mathcal{M}$ rejects.
   – If $c_1 = 0$ after the completion of $(*)$, $\mathcal{M}$ accepts, else $\mathcal{M}$ sets $k := k + 1$ and goes back to $(*)$.

Clearly, the values of the counters when k becomes $k + 1$ are: $c_1 = n - (1 + 3 + \cdots + (2k - 1) + (2k + 1)) = n - (k + 1)^2, c_2 = 0, c_3 = (2k + 1)$. It follows that M can be constructed to accept the set $\{n^2 \mid n \geq 1\}$, which is not semilinear.

From the above discussion and example, we have the following theorem.

Theorem 23 *Deterministic* 3-*counter cascade machines can accept nonsemilinear sets. Moreover, their emptiness problem is undecidable.*

Remark 2 The construction of the deterministic 3-counter cascade machine in the example above can be modified to accept the set $R_1 = \{2n^2 \mid n \geq 1\}$. Now, define for each $k \geq 1$, the set $R_k = \{2n^{2^k} \mid n \geq 1\}$. One can show by essentially iterating the construction in the example that R_k can be accepted by a deterministic $(2 + k)$-counter cascade machine. We believe (but have no proof at this time), that the R_k's form an infinite hierarchy: R_{k+1} can be accepted by a deterministic $(2 + k)$-counter cascade machine but cannot be accepted by any deterministic or nondeterministic $(2 + (k - 1))$-counter cascade machine. Note that 1- and 2-counter cascade machines are equivalent—both accept exactly the semilinear sets.

It is interesting to observe that for a k-counter cascade machine $\mathcal{M}$, if counter c_1 cannot be tested for zero, then either $R(\mathcal{M}) = \emptyset$ (if $\mathcal{M}$ never enters an accepting state regardless of the input initially given in c_1) or there exists an $m \in \mathbb{N}^1$ such that $R(\mathcal{M}) = \{n \mid n \geq m, n \in \mathbb{N}^1\}$ (m is the smallest input for which $\mathcal{M}$ accepts). Hence, for cascade counter machines lacking the capability of testing counter c_1 for zero, they accept only semilinear sets. The emptiness problem, nevertheless, remains undecidable for such a restricted class of cascade counter machines, implying that the semilinear sets associated with such machines are not effective.

We conclude this section by noting Theorem 23 is not true for (deterministic or nondeterministic) 2-counter cascade machines. In fact, consider a nondeterministic machine $\mathcal{M}$ having $k+1$ counters, where the first k counters are initially set to input values $n_1, \dots, n_k$, respectively, and the last counter set to zero. The computation is restricted in that the first k counters can only be decremented, but the last counter can decrement/increment independently. It follows from Theorem 21 these machines accept exactly the semilinear sets.

Cascade SA P systems. A *cascade SA P system* consists of k membranes $m_1, \dots, m_k$ (for some k) that are nested: For $1 \le i \le k-1$, membrane m_i is enclosed in membrane m_{i+1}. The input membrane, m_1, initially contains o^n for some n. Again, in the rules of the forms (v, in) and $(u, out; v, in)$, v does not contain o's. There are fixed multisets $w_1, \dots, w_k$ not containing o's in membranes $m_1, \dots, m_k$ initially. The environment initially contains a fixed multiset of symbols.

The connection between cascade counter machines and cascade SA P systems is given by the following theorem.

Theorem 24 *Let $k \ge 1$ be a positive integer. A set $Q \subseteq \mathbb{N}^1$ is accepted by a k-membrane cascade SA P system if and only if it can be accepted by a k-counter cascade machine.*

Proof Let Π be a k-membrane cascade SA P system. We construct an equivalent k-counter cascade machine $\mathcal{M}$. We associate a counter c_i for every membrane m_i to keep track of the number of o's in membrane m_i during the computation. The construction of $\mathcal{M}$ simulating Π is straightforward, following the strategy in the proof of Lemma 4.

We now prove the converse. Let $\mathcal{M}$ be a k-counter cascade machine. For notational convenience, we will assume the program instructions for $\mathcal{M}$ are labeled $l_0, l_1, \dots, l_n$ and begin with instruction l_0. We also assume they are written in the form $l_i : (\mathtt{SUB}(r), l_s, l_t)$ meaning that when instruction l_i is executed, counter r is decremented. If counter r was initially positive (meaning it was able to be decremented), the machine will next execute the instruction l_s, otherwise it will execute the instruction l_t. Also, since $\mathcal{M}$ is a cascade counter machine, each decrement from counter r where $r < k$ must be followed by an instruction which increments the counter $r+1$. Hence, we can incorporate each increment instruction into its preceding decrement instruction. (In the case where we decrement counter r and $r = k$, no increment instruction follows since the decremented value is thrown out of the system.) In this way, we can consider the program for $\mathcal{M}$ to consist entirely of decrement instructions. We now construct an equivalent k-membrane cascade SA P system Π which simulates each decrement instruction of $\mathcal{M}$. The membrane structure of Π is a set of nested membranes which each correspond to a counter in $\mathcal{M}$. The skin membrane also acts as program control membrane. Formally, the simulation occurs by creating the following cascade SA P system from a given cascade counter machine: $\Pi = \langle V, H, \mu, w_{m_1}, \dots, w_{m_k}, E, R_{m_1}, \dots, R_{m_k} \rangle$ where $V = \{l_{i1}, l_{i2}, l_{i3}, l_{i4}, d_{ij} \mid l_i$ is an instruction label of the form $l_i : (SUB(r), l_s, l_t)$

where $r \neq k$ and $0 \leq j \leq 2(k-r)+1\} \cup \{l_{i1}, l_{i2}, l_{i3} \mid l_i$ is an instruction label of the form $l_i : (SUB(r), l_j, l_s)$ where $r = k\} \cup \{d_0, d_1\} \cup \{c, c', c_1, \ldots, c_m\} \cup \{o\}$. $H = \{m_1, m_2, \ldots, m_k\}$. $\mu = [_{m_k}[_{m_{k-1}} \cdots [_{m_1}]_{m_1} \cdots]_{m_{k-1}}]_{m_k}$. $w_{m_1} = c_1 o^n$. $w_{m_i} = c_i$ for all $1 < i < k$. $w_{m_k} = l_{01} c_k$ (since l_0 is the first instruction to execute). $E =$ one copy of each element in V except o and l_{01}.

Rule sets $(R_{m_1}, \ldots, R_{m_k})$ are created based on the cascade machine's program. Rule $(d_0, out; d_1, in)$ is initially created within R_{m_k}. For each rule of the form l_i : $(\mathtt{SUB}(r), l_s, l_t)$ where $r \neq k$ we add the rules:

1. R_{m_k} contains the rules $(l_{i1}, out; l_{i2} c d_0 d_{i0}, in)$, $(d_{ij}, out; d_{i(j+1)}, in)$ where $0 \leq j \leq 2(k-r)$, $(d_1 d_{i[2(k-r)+1]}, out; l_{i4} c', in)$, $(l_{i2} d_1, out; l_{i3}, in)$, $(l_{i3} c d_{i[2(k-r)+1]}, out; l_{s1}, in)$, $(l_{i2} l_{i4}, out; l_{t1}, in)$, (cc', out).
2. R_{m_n} where $k \geq n > r$ contains the rules $(l_{i2} c, in)$, $(l_{i2} c_r, out)$, $(l_{i3} c_r, in)$, $(l_{i3} c, out)$, $(l_{i4} c', in)$, $(l_{i2} l_{i4}, out)$, (cc', out).
3. R_{m_r} contains the rules $(c_r, out; l_{i2} c, in)$, $(l_{i2} o, out)$, $(l_{i3} c_r, in)$, $(l_{i3} c, out)$, $(l_{i2}, out; c_r c', in)$, (cc', out).

For a rule of the form $l_i : (SUB(r), l_s, l_t)$ where $r = k$, we create the following rules:

1. $R_{m_r} = R_{m_k}$ contains rules $(l_{i1}, out; l_{i2} d_0, in)$, $(l_{i2} o, out; l_{i3}, in)$, $(l_{i3} d_1, out; l_{s1}, in)$, $(l_{i2} d_1, out; l_{t1}, in)$.

Informally, the above simulation operates as follows. The process of simulating a single subtract instruction $l_i : (\mathtt{SUB}(r), l_s, l_t)$ if $r \neq k$ begins by the presence of the object l_{i1} within the outermost membrane (m_k). This object is used to bring in the necessary execution objects l_{i2}, c, d_0, and d_{i0} using rule $(l_{i1}, out; l_{i2} c d_0 d_{i0}, in)$. The objects l_{i2} and c are used cooperatively and are drawn deeper through the membrane hierarchy until they have reached the membrane m_{r+1}. Here, they are drawn into membrane m_r while the object c_r is thrown out.

If membrane m_r contains an o object (meaning counter r is not empty), the objects l_{i2} and o are thrown out into membrane m_{r+1}. This simulates both the current subtract instruction along with the add instruction we know must follow. Now, the objects l_{i2} and c_r are used cooperatively and are thrown out of each membrane until they located in the skin membrane.

While this has been occurring, the delay objects in the skin membrane have been being incremented. The d objects are delay objects and are used to delay certain execution steps. During each step of computation, their subscripts are incremented by one. The object d_0 only changes to d_1 to delay an action for a single step while the object d_{i0} increments to $d_{i[2(k-r)+1]}$. This number $(2(k-r)+1)$ corresponds to the number of steps plus one that l_{i2} will take to travel to membrane r and back if membrane r contains a o.

This allows us to determine whether the object l_{i2} is stuck in membrane r.

If the membrane m_r contains an o (meaning counter r is not zero), objects l_{i2} and c_r will return to the skin membrane in $2(k-r)$ steps and rule $(l_{i2} d_1, out; l_{i3}, in)$ is applicable before $d_{i[2(k-r)+1]}$ is brought into the membrane. So, l_{i2} and d_2 are thrown out into the environment and object l_{i3} is brought into the system. Now, the objects c and c_r must be swapped to their original positions. This occurs by having objects l_{i3} and c_r work cooperatively to move deeper through the membranes to membrane r. Then objects l_{i3} and c work cooperatively to be thrown out of each membrane until returning to the skin membrane. At this point, everything is complete and all objects are in the correct location. So, objects l_{i3}, c, and $d_{i[2(k-r)+1]}$ are thrown out into the environment while object l_{s1} is brought in. Now, instruction l_i is complete and instruction l_s will execute next.

If the objects l_{i2} and c_r have not returned to the skin membrane after $2(k-r)+1$ steps, then the membrane r must not have contained an o. At this point, the objects d_1 and $d_{i[2(k-r)+1]}$ are thrown out of the skin membrane and objects l_{i4} and c' are brought in. Now, objects l_{i4} and c' work cooperatively to move deeper through the membranes to membrane m_{r+1}. Object c' is drawn into membrane m_r while object l_{i2} is thrown out. At this point, membrane m_r contains the objects c and c' while membrane m_{r+1} contains the objects l_{i2} and l_{i4}. These pairs of objects work cooperatively to be thrown out of each membrane. The pair $l_{i2}l_{i4}$ will get to the skin membrane a step ahead of the pair cc'. The objects l_{i2} and l_{i4} are thrown out into the environment while bringing in the object l_{t1}. During the next step, the pair cc' will be thrown out into the environment. At this point, instruction l_i is complete and instruction l_t will execute next.

If the instruction to be simulated is of the form $l_i : (\mathtt{SUB}(r), l_s, l_t)$ where $r = k$, the simulation is much simpler. In this case, since the instruction is immediately placed within the counter membrane, only a single delay object is needed along with the instruction object l_{i2}. If membrane k contains an o, it is thrown out during the next step along with the object l_{i2} and the object l_{i3} is brought in allowing the final step to clean up and bring in the instruction object l_{s1}. If l_{i2} is still in membrane m after one step, the delay object can cooperate with object l_{i2} to bring in the next instruction object l_{t1}.

Consequently, these cascade SA P system rules simulate the operation of $\mathcal{M}$. □

From Theorems 23 and 24, we have the following corollary.

Corollary 13 3-*Membrane cascade SA P systems can accept nonsemilinear sets. Moreover, their emptiness problem is undecidable.*

A careful examination of the proof of Theorem 24 reveals that the degree of maximal parallelism for the constructed SA P system is finite (i.e., at every step of the computation, the size of the multiset of applicable rules is bounded by some fixed integer). Hence, Corollary 13 holds even if the 3-membrane cascade SA P systems have a bounded degree of maximal parallelism.

4.3 k-Membrane Extended Cascade SA P Systems

The k-membrane cascade SA P system of the previous section can be generalized. A *k-membrane extended cascade SA P system* has a set of objects $V = F \cup \Sigma_r$, where now the input alphabet is $\Sigma_r = \{a_1, \ldots, a_r\}$ ($r \geq 1$). Again, the rules are restricted in that in the rules of the forms (v, in) and $(u, out; v, in)$, v does not contain any symbol in Σ_r. The environment initially contains only F. There are fixed strings $w_i \in F^*$, such that the system starts with $w_1 a_1^{n_1} \ldots a_r^{n_r}$ in membrane m_1 (the input membrane) and w_i in membrane m_i for $2 \leq i \leq k$. If the system halts, then we say that the r-tuple $(n_1, \ldots, n_r)$ is accepted.

Now, consider a finite-state device $\mathcal{M}$ with a finite-state control and a "bag" containing a multiset of symbols. $\mathcal{M}$ starts in its initial state with the multiset $a_1^{n_1} \cdots a_r^{n_r}$. $\mathcal{M}$'s instructions of are of the following form:

$q \rightarrow (q'$ delete a_i from the bag if it is in the bag else $q'')$.

Thus, from state q, $\mathcal{M}$ removes a_i from the bag if it is in the bag and goes to state q'; otherwise, $\mathcal{M}$ goes to state q''. The initial multiset in the bag is accepted if $\mathcal{M}$ enters an accepting state. We call this device a *1-bag automaton*. A 1-bag automaton is like a multiset automaton studied in [4]. Although the notion is not the same, the idea is quite similar.

We can generalize the 1-bag automaton to a *k-bag automaton*, where now, a symbol is deleted from bag i if and only if it is exported into bag $i + 1$. A symbol can be deleted from the kth bag independently.

Lemma 6 *A set $R \subseteq \mathbb{N}^r$ is accepted by a 1-bag automaton if and only if it is accepted by a decreasing r-counter machine.*

Proof Clearly, deleting a_i from the bag corresponds to decrementing counter i ($1 \leq i \leq r$). □

Theorem 25 *Let $k \geq 1$. A set of tuples R is accepted by a k-membrane extended cascade SA P system if and only if R is accepted by a k-bag automaton.*

Proof The proof for the "only if" part is a straightforward generalization of the proof of the first part of Theorem 21 (which was for $k + 1$). For the second part, let $\mathcal{M}$ be a k-bag automaton. We construct a k-membrane extended cascade SA P system Π equivalent to $\mathcal{M}$, in the same manner as the construction of Theorem 24 where each membrane corresponds to a bag. The rules can be created by mapping each subtraction rule of the form $l_i : (SUB(r), l_s, l_t)$ to the bag rule of the form $q \rightarrow (q'$ delete a_i from the bag if it is in the bag else $q'')$ as follows. The instruction labels of a counter machine can also be viewed as states so we can say l_i corresponds to q, l_s corresponds to q', and l_t corresponds to q''. The bag associated with q corresponds to the counter r. The additional difference is that the bag also specifies the object (a_i) in Σ which should be thrown out of the bag. Hence, we can create Π to simulate $\mathcal{M}$ using the techniques in Theorem 24 and the above mapping along with the following changes. The set V will now additionally contain the set of objects $\{a_1, \ldots, a_r\}$ rather than the single object $\{o\}$. The set $w_{m_1} = c_1 a_1^{n_1} \cdots a_r^{n_r}$ rather than $c_1 o^n$. Also, the rules $(l_{i2}o, out)$ and $(l_{i2}o, out; l_{i3}, in)$ will be changed to $(q_2 a_i, out)$ and $(q_2 a_i, out; s_3, in)$, respectively. Clearly, this k-membrane extended cascade SA P system now simulates a k-bag automaton. □

Theorem 26 *A set $R \subseteq \mathbb{N}^r$ is accepted by a* 1*-membrane extended cascade SA P system if and only if it is a semilinear set.*

Proof Let Π be a 1-membrane extended cascade SA P system with input alphabet Σ_r. We can easily construct a decreasing r-counter machine $\mathcal{M}$ which, when the counters are initially given $n_1, \dots, n_r$, simulates the computation of Π on $wa_1^{n_1} \cdots a_r^{n_r}$. The simulation is straightforward, as in Lemma 4. It follows from Theorem 21 that $R(\Pi)$ is a semilinear set.

For the converse, by Lemma 6, we need only show that a 1-bag automaton can be simulated by a 1-membrane extended cascade SA P system. This follows from Theorem 25. □

Let $\Sigma_2 = \{a_1, a_2\}$. Using the ideas in the example of the previous section, we can easily construct a 2-bag automaton accepting the nonsemilinear set $\{(n_1, n_2) \mid n_1, n_2 \geq 0, n_1 + n_2 = m^2$ for some $m \geq 1\}$. It is also easy to construct a 2-bag automaton with input alphabet Σ_2 that simulates the computations of a 2-counter automaton. The values of the counters are represented in the second bag. The number of a_1's (resp., a_2's) in that bag denotes the value of the first (resp., second) counter. The a_1's and the a_2's in the first bag are the suppliers (sources) of the "increments" for the two counters in the second bag.

From the above discussion and Theorem 25, we have the following theorem.

Theorem 27 2*-bag automata* (*and*, *hence*, 2*-membrane extended cascade SA P systems*) *can accept nonsemilinear sets*. *Moreover*, *their emptiness problem is undecidable*.

4.4 Restricted SA P System Generators

In this section, we look at SA P systems used as generators of tuples. In the definition of a k-membrane cascade SA P system, the input o^n is initially given in m_1 (the innermost membrane) with no o's in the other membranes. The computation is such that the o's can only be exported from membrane m_i to membrane m_{i+1} (or to the environment in the case of m_k).

Now, consider a model Π which is a generator of tuples and the cascading (flow of o's) is from the environment to the innermost membrane. More precisely, let $m_1, \dots, m_k$ be the membranes of Π, where m_i is enclosed in m_{i+1} for $1 \leq i \leq k-1$ (m_1 is the innermost membrane and m_k is the skin membrane). Initially, there are no o's in the membranes, but there is an infinite supply of o's in the environment. There may also be a finite supply of other symbols in the environment initially. Rules of the forms (u, out) and $(u, out; v, in)$ are restricted in that u cannot contain o's. Thus, o's can only move from the environment to membrane m_k and from m_{i+1} to m_i for $1 \leq i \leq k-1$. (Note once o's reach membrane m_1, they remain there.) The set of numbers generated by Π is $G(\Pi) = \{n \mid \Pi$ halts with o^n in the skin membrane $m_k\}$.

It is important to note that the skin membrane is the output membrane. We call this new model a *k-membrane reverse-cascade SA P system.*

Theorem 28

1. 1-*Membrane and* 2-*membrane reverse-cascade SA P systems are equivalent*, *and they generate exactly the semilinear sets over* $\mathbb{N}^1$.
2. 3-*Membrane reverse-cascade SA P systems can generate nonsemilinear sets*. *In fact*, *for any recursively enumerable* (*RE*) *set* $R \subseteq \mathbb{N}^1$, *the set* $\{2^n \mid n \in R\}$ *can be generated by a* 3-*membrane reverse-cascade SA P system*. (*Hence*, *their emptiness problem is undecidable*.)
3. *Any RE set R can be generated by a* 4-*membrane reverse-cascade SA P system*.

The proof of Theorem 28 involves the use of a counter machine similar to the k-counter cascade machine in Sect. 3. Define a *k-counter reverse-cascade machine* $\mathcal{M}$ as a nondeterministic machine with k counters $c_1, \ldots, c_k$. $\mathcal{M}$ starts in its initial state with all counters zero. As usual, the counters can be incremented/decremented and tested for zero but with the following restrictions:

1. If counter c_{i+1} is decremented it must be followed by an increment of counter c_i for $1 \leq i \leq k-1$, and this is the only way counter c_i can be incremented.
2. Counter c_k can be incremented independently.
3. Counter c_1 cannot be decremented. (Thus, c_1 is nondecreasing, hence essentially useless. The reason is that once it becomes positive, it will remain positive, and can no longer affect the computation. We include this counter for convenience.)

We say that $\mathcal{M}$ generates a nonnegative integer n if it halts with value n in counter c_k, and the set of all such numbers generated is the set generated by $\mathcal{M}$.

It can be shown that for any $k \geq 1$, a set $R \subseteq \mathbb{N}^1$ is generated by a k-membrane reverse-cascade SA P system if and only if it can be generated by a k-counter reverse-cascade machine. Then to prove items (1), (2), and (3) of Theorem 28, we need only show that they hold for 1-counter/2-counter, 3-counter, and 4-counter reverse-cascade machines, respectively.

Remark 3 We believe that the 4 membranes in Theorem 28, item (3) is the best possible. We think that there are RE sets (even recursive sets) that cannot be generated by 3-counter reverse-cascade machines based on the following discussion.

By definition, in a 3-counter reverse-cascade machine $\mathcal{M}$, with three counters, c_1, c_2, c_3, counter c_1 cannot be decremented. So, in fact, the computation of $\mathcal{M}$ can be simulated by a machine $\mathcal{M}'$ with only two counters: d_1, d_2. Again, the only restriction is that if d_2 is decremented, it must be followed by an increment of d_1, and this is the only way d_1 can be incremented. But now, we allow d_1 to be decremented independently and, as before, d_2 can be incremented independently.

We conjecture that there is an RE set (even a recursive set) that cannot be generated by a 2-counter machine $\mathcal{M}'$ as defined above. (Note that by definition, the generated number is in counter d_2 when the machine halts.) However, we have no formal proof at this time.

We can generalize the reverse-cascade SA P system by using, instead of only one input symbol o, a set of symbols $\Sigma_r = \{a_1, \ldots, a_r\}$ as input symbols, again with the restriction that these symbols can only be moved from the environment to membrane m_k and from m_{i+1} to m_i for $1 \leq i \leq k-1$. Now the system generates a set of r-tuples of nonnegative integers in the skin membrane when it halts. We can prove the following theorem.

Theorem 29

1. 1-*Membrane reverse-cascade SA P systems with input alphabet* $\Sigma_r = \{a_1, \ldots, a_r\}$ *generate exactly the semilinear sets over* $\mathbb{N}^r$.
2. 2-*Membrane reverse-cascade SA P systems with input alphabet* $\Sigma_2 = \{a_1, a_2\}$ *can generate nonsemilinear sets over* $\mathbb{N}^2$. *In fact, for any RE set* R, *the set* $\{(2^n, 0) \mid n \in R\}$ *can be generated by a* 2-*membrane reverse-cascade SA P system with input alphabet* Σ_2.
3. *For any RE set* R, *the set* $\{(n, 0) \mid n \in R\}$ *can be generated by a* 3-*membrane reverse-cascade SA P system with input alphabet* Σ_2.
4. *For any RE set* R, *the set* $\{(n, 0, 0) \mid n \in R\}$ *can be generated by a* 2-*membrane reverse-cascade SA P system with input alphabet* Σ_3.

Remark 4 Again, as in Remark 3, we believe that Theorem 29, item (3) does not hold for 2-membrane reverse-cascade SA P systems with input alphabet Σ_2.

In the definition of a reverse-cascade SA P system, the skin membrane is the output membrane. We now consider the model where the output membrane is the innermost membrane m_1 (and not the skin membrane). Similar to Theorem 28, we can prove the following (but item (3) is weaker).

Theorem 30 *Under the assumption that the output membrane is the innermost membrane* m_1 (*and not the skin membrane*), *we have*:

1. 1-*Membrane and* 2-*membrane reverse-cascade SA P systems are equivalent, and they generate exactly the semilinear sets over* $\mathbb{N}$.
2. 3-*Membrane reverse-cascade SA P systems can generate nonsemilinear sets* (*e.g., the set* $\{n^2 \mid n \geq 1\}$). *Moreover, their emptiness problem is undecidable.*
3. *Any RE set* R *can be generated by a* 5-*membrane reverse-cascade SA P system.*

Remark 5 It does not seem that item (3) of the above theorem holds for a 4-membrane reverse-cascade SA P system, but we have no proof at this rime.

Finally, consider a 2-level SA P system Π which has membranes $m_1, m_1, \ldots, m_{k+1}$, where $m_1, \ldots, m_k$ are at the same level, and they are enclosed in the skin membrane m_{k+1}. The environment contains F initially and an infinite supply of o's. We require that for membranes $m_1, \ldots, m_k$, in the rules of the forms (u, out) and

(u, out, v, in), u does not contain o's. Note that there is no restriction on the rules in the skin membrane. For this system, we say that $(n_1, \ldots, n_k)$ is generated if, when Π is started with no o's in the system and fixed $w_i \in F^*$ in m_i $(1 \leq i \leq k+1)$, Π halts with $o^{n_1}, \ldots, o^{n_k}$ in membranes $m_1, \ldots, m_k$. Call the system just described a *simple SA P system generator*. We can show the following theorem.

Theorem 31 *A set $R \subseteq \mathbb{N}^k$ is generated by a simple SA P system generator if and only if R is semilinear.*

The theorem above no longer holds when the simple SA P system generator is extended to a 3-level structure, as Theorem 30, item (2) shows.

References

1. The P systems web page. http://psystems.disco.unimib.it
2. Alhazov A, Freund R, Oswald M (2005) Symbol/membrane complexity of P systems with symport/antiport rules. In: Proceedings of the 6th international workshop on membrane computing, pp 123–146
3. Csuhaj-Varju E, Ibarra OH, Vaszil G (2004) On the computational complexity of P automata. In: Proceedings of the 10th international meeting on DNA computing (DNA10). Lecture notes in computer science. Springer, Berlin, pp 97–106
4. Csuhaj-Varjú E, Martín-Vide C, Mitrana V (2000) Multiset automata. In: Proceedings of the workshop on multiset processing. Lectures notes in computer science, vol 2235. Springer, Berlin, pp 69–84
5. Dang Z, Ibarra OH, Li C, Xie G (2005) On model-checking of P systems. In: Proceedings of the 4th international conference on unconventional computation
6. Freund R, Păun G (2003) On deterministic P systems. In: [1]
7. Ginsburg S (1966) The mathematical theory of context-free languages. McGraw–Hill, New York
8. Greibach S (1978) Remarks on blind and partially blind one-way multicounter machines. Theor Comput Sci 7:311–324
9. Hauschildt D, Jantzen M (1994) Petri net algorithms in the theory of matrix grammars. Acta Inform (Hist Arch) 31(8):719–728
10. Ibarra OH (1978) Reversal-bounded multicounter machines and their decision problems. J ACM 25:116–133
11. Ibarra OH (2005) The number of membranes matters. Theor Comput Sci
12. Ibarra OH (2005) On determinism versus nondeterminism in P systems. Theor Comput Sci
13. Ibarra OH, Woodworth S, Yen H-C, Dang Z (2005) On symport/antiport P systems and semilinear sets. In: Proceedings of the 6th international workshop on membrane computing (WMC6). Lecture notes in computer science. Springer, Berlin, pp 253–271
14. Ito M, Martín-Vide C, Păun Gh (2001) A characterization of Parikh sets of ETOL languages in terms of P systems. In: Ito M, Păun Gh, Yu S (eds) Words, semigroups, and transductions. Festscrift in honor of Gabriel Thierrin. World Scientific, Singapore, pp 239–253
15. Martín-Vide C, Pazos J, Păun G, Rodríguez-Patón A (2003) Tissue P systems. Theor Comput Sci 296(2):295–326
16. Minsky M (1961) Recursive unsolvability of Post's problem of tag and other topics in the theory of Turing machines. Ann Math 74:437–455
17. Monien B (1980) Two-way multihead automata over a one-letter alphabet. RAIRO Inform Theor 14(1):67–82

18. Păun A, Păun G (2006) The power of communication: P systems with symport/antiport. New Gener Comput 20(3):295–306
19. Păun G, Pazos J, Pérez-Jiménez MJ, Rodríguez-Patón A (2005) Symport/antiport P systems with three objects are universal. Fundam Inform 64(1–4):353–367
20. Savitch W (1970) Relationships between nondeterministic and deterministic tape complexities. J Comput Syst Sci 4(2):177–192
21. Savitch W (1973) A note on multihead automata and context-sensitive languages. Acta Inform 2:249–252
22. Sosík P (2002) P systems versus register machines: two universality proofs. In: Proceedings of workshop on membrane computing (WMC-CdeA2002), pp 371–382

Spiking Neural P Systems. Recent Results, Research Topics

Gheorghe Păun and Mario J. Pérez-Jiménez

Abstract After a quick introduction of spiking neural P systems (a class of P systems inspired from the way neurons communicate by means of spikes, electrical impulses of identical shape), and presentation of typical results (in general equivalence with Turing machines as number computing devices, but also other issues, such as the possibility of handling strings or infinite sequences), we present a long list of open problems and research topics in this area, also mentioning recent attempts to address some of them. The bibliography completes the information offered to the reader interested in this research area.

1 Forecast

It is obvious that the (human) brain structure and functioning, from neurons, astrocytes, and other components to complex networks and complex (chemical, electrical, informational) processes taking place in it, should be—and only partially is—a major source of inspiration for informatics (we choose this more general term rather that the restrictive, but usual, "computer science", in order to stress that we have in mind both mathematical approaches, with intrinsic motivation, and practical approaches, both the theory of computability and the use of computing machineries). If biology is such a rich source of inspiration for informatics as natural computing proves, then the brain should be the "golden mine" of this intellectual enterprise. Risking a forecast, we believe that *if something really great is to appear in informatics in the near future*, *then this "something" will be suggested by the brain* (*and this will probably be placed at the level of "strategies" of computing*, *not at the "tactic" level*—just in balance with the two computing devices already learned from the brain activity and which can be considered the most central notions in informatics, the Turing machine, and the finite automaton).

The previous statements do not intend to suggest that spiking neural P systems are the answer to this learning-from-brain challenge, but only to call (once again) the

G. Păun (✉)
Institute of Mathematics of the Romanian Academy, PO Box 1-764, 014700 Bucureşti, Romania
e-mail: george.paun@imar.ro

G. Păun
Department of Computer Science and Artificial Intelligence, University of Sevilla, Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
e-mail: gpaun@us.es

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_15, © Springer-Verlag Berlin Heidelberg 2009

attention to this challenge. Becoming familiar with brain functioning, in whatever reductionistic framework (as spiking neural P systems investigation is), can however be useful. After all, “the road of one thousand miles starts with the first step”, Lao Tze said…

2 Some (Neural) Generalities

The neuron is a highly specialized cell, at the same time intricate and simple, robust and fragile, like any other cell, but having the particularity of being involved (in general) in huge networks by means of the synapses established with partner neurons. It is not at all the intention of these lines to give any biological information from this area, but only to point out some of the peculiarities related to neurons and the brain: the functioning of each neuron assumes chemical, electrical, and informational processing at the same time; the axon is not a simple transmitter of impulses, but an information processor; in the communication between neurons the spiking activity plays a central role (which means that the distance in time between consecutive spikes is used to carry information, that is, time is a support of information); the neurons are not cooperating only through synapses, but their relationships are also regulated through the calcium waves controlled by the astrocytes, “eavesdroppers” of axons playing an important role in the neural communication; the brain displays a general emergent behavior which, to the best of our knowledge, cannot be explained only in terms of neuron interrelationships (something is still missing in this picture, maybe of a quantum nature—as Penrose suggests, maybe related to the organization of parts, maybe of a still subtler or even unknown nature). Some of these ideas (especially spiking) are supposed to lead to “neural computing of the third generation”, which suggests that already computer scientists are aware of the possibility of major progresses to be made (soon) on the basis of progresses in neurobiology.

The bibliography of this note contains several titles, both from the general biology of the cell [1], general neurology [51], and from neural computing based on spiking [4, 17, 33–36], about the axon as an information processor [49], astrocytes and their role in the brain functioning [46, 50]. Of course, these titles are only meant to be initial “dendrites” to the huge bibliography related to (computer science approaches to) brain functioning.

3 Spiking Neural P Systems—An Informal Presentation

Spiking neural P systems (SN P systems, for short) were introduced in [26] in the precise (and modest: trying to learn a new “mathematical game” from neurology, not to provide models to it) aim of incorporating in membrane computing ideas specific to spiking neurons; the intuitive goal was to have (1) a tissue-like P system with (2) only one (type of) object(s) in the cells—the *spike*, with (3) specific rules for evolving populations of spikes, and (4) making use of the time as a support of information.

In what follows, we briefly describe several classes of SN P systems investigated so far, as well as some of the main types of results obtained in this area.

In short, an SN P system (of the basic form—later called a *standard* SN P system) consists of a set of *neurons* placed in the nodes of a directed graph and sending signals (*spikes*, denoted in what follows by the symbol a) along the arcs of the graph (these arcs are called *synapses*). The objects evolve by means of *spiking rules*, which are of the form $E/a^c \rightarrow a; d$, where E is a regular expression over $\{a\}$ and c, d are natural numbers, $c \geq 1, d \geq 0$. The meaning is that a neuron containing k spikes such that $a^k \in L(E), k \geq c$, can consume c spikes and produce one spike, after a delay of d steps. This spike is sent to all neurons to which a synapse exists outgoing from the neuron where the rule was applied. There also are *forgetting rules*, of the form $a^s \rightarrow \lambda$, with the meaning that $s \geq 1$ spikes are removed, provided that the neuron contains exactly s spikes.

An extension of theses type of rules was considered (with a mathematical motivation) in [14, 37]: rules of the form $E/a^c \rightarrow a^p; d$, with the meaning that when using the rule, c spikes are consumed and p spikes are produced (one assumes that $c \geq p$, not to produce more than consuming). Because p can be 0 or greater than 0, we obtain a generalization of both spiking and forgetting rules, while forgetting rules also have a regular expression associated with them.

An SN P system (with standard as well with extended rules) works in the following way. A global clock is assumed and in each time unit each neuron which can use a rule should do it (the system is synchronized), but the work of the system is sequential locally: only (at most) one rule is used in each neuron. One of the neurons is considered to be the *output neuron*, and its spikes are also sent to the environment. The moments of time when a spike is emitted by the output neuron are marked with 1, the other moments are marked with 0. This binary sequence is called the *spike train* of the system—it might be infinite if the computation does not stop.

With a spike train we can associate various numbers, which can be considered as *computed* (we also say *generated*) by an SN P system. For instance, in [26] only the distance between the first two spikes of a spike train was considered, then in [42] several extensions were examined: the distance between the first k spikes of a spike train, or the distances between all consecutive spikes, taking into account all intervals or only intervals that alternate, all computations or only halting computations, etc.

An SN P system can also work in the *accepting* mode: a neuron is designated as the *input neuron* and two spikes are introduced in it, at an interval of n steps; the number n is accepted if the computation halts.

Two main types of results were obtained: computational completeness in the case when no bound is imposed on the number of spikes present in the system, and a characterization of semi-linear sets of numbers in the case when a bound is imposed.

Another attractive possibility is to consider the spike trains themselves as the result of a computation, and then we obtain a device generating a (binary) language.

We can also consider both input and output neurons and then an SN P system can work as a transducer. Such possibilities were investigated in [43]. Languages—even on arbitrary (i.e., not only binary) alphabets—can be obtained also in other ways: following the path of a designated spike across neurons, as proposed in [12], or using rules of the extended form mentioned above. Specifically, with a step when the system sends out i spikes, we associate a symbol b_i, and thus we get a language over an alphabet with as many symbols as the number of spikes simultaneously produced. This case was investigated in [14], where representations or characterizations of various families of languages were obtained. (An essential difference was found between the case when zero spikes sent out is interpreted as a symbol b_0 and the case when this is interpreted as inserting λ, the empty string, in the result.)

Other extensions were proposed in [24] and [22], where several output neurons were considered, thus producing vectors of numbers, not only numbers. A detailed typology of systems (and of sets of vectors generated) is investigated in the two papers mentioned above, with classes of vectors found in between the semi-linear and the recursively enumerable ones.

The proofs of all computational completeness results known up to now in this area are based on simulating register machines. Starting the proofs from small universal register machines, as those produced in [29], one can find small universal SN P systems (working in the generating mode, as sketched above, or in the computing mode, i.e., having both an input and an output neuron and producing a number related to the input number). This idea was explored in [37] and the results are as follows: there are universal computing SN P systems with 84 neurons using standard rules and with only 49 neurons using extended rules. In the generative case, the best results are 79 and 50 neurons, respectively.

In the initial definition of SN P systems, several ingredients are used (delay, forgetting rules); some of them of a general form (unrestricted synapse graph, unrestricted regular expressions). As shown in [21], several normal forms can be found, in the sense that some ingredients can be removed or simplified without losing the computational completeness. For instance, the forgetting rules or the delay can be avoided, and the outdegree of the synapse graph can be bounded by 2, while the regular expressions from firing rules can be of very restricted forms. The dual problem, of the indegree bounding, was solved (affirmatively) in [44].

Besides using the rules of a neuron in the sequential mode introduced above, it is possible to also use the rules in a parallel way. A possibility was considered in [27]: when a rule is enabled, it is used as many times as possible, thus exhausting the spikes it can consume in that neuron. As proved in [27], SN P systems with the exhaustive use of rules are again universal, both in the accepting and the generative cases.

In the proof of these results, the synchronization plays a crucial role, but both from a mathematical point of view and from a neuro-biological point of view, it is rather natural to consider non-synchronized systems, where the use of rules is not obligatory: even if a neuron has a rule enabled in a given time unit, this rule is not obligatorily used, the neuron may remain still, maybe receiving spikes from the neighboring neurons; if the unused rule may be used later, it is used later, without any restriction on the interval when it has remained unused; if the new spikes made the rule non-applicable, then the computation continues in the new circumstances (maybe other rules are enabled now). This way of using the rules applies also to the output neuron, hence now the distance in time between the spikes sent out by the system is no longer relevant. That is why, for non-synchronized SN P systems we take as a result of a computation the total number of spikes sent out; this, in turn, makes necessary considering only halting computations (the computations never halting are ignored, they provide no output). Non-synchronized SN P systems were introduced and investigated in [8], where it is proved that SN P systems with extended rules are still equivalent with Turing machines (as generators of sets of natural numbers).

4 Some (More) Formal Definitions

To make clearer some of the subsequent formulations, we recall here the definition of central classes of SN P systems, but more details should be found in the papers mentioned in the bibliography. No general notions or notations from language or automata theory, computability, complexity, computer science in general or membrane computing, are recalled.

A *spiking neural P system* (in short, an SN P system), of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, out),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, where:
   (a) $n_i \geq 0$ is the *initial number of spikes* contained by the neuron;
   (b) R_i is a finite set of *rules* of the following general form:

$$E/a^c \rightarrow a^p; d,$$

   where E is a regular expression with a the only symbol used, $c \geq 1$, and $p, d \geq 0$, with $c \geq p$; if $p = 0$, then $d = 0$, too.
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses*);
4. $out \in \{1, 2, \ldots, m\}$ indicates the *output neuron*.

A rule $E/a^c \rightarrow a^p; d$ with $p \geq 1$ is called a *firing* (we also say *spiking*) *rule*; a rule $E/a^c \rightarrow a^p; d$ with $p = d = 0$ is written in the form $E/a^c \rightarrow \lambda$ and is called a *forgetting rule*. If $L(E) = \{a^c\}$, then the rules are written in the simplified form $a^c \rightarrow a^p; d$ and $a^c \rightarrow \lambda$. A system having only rules of the forms $E/a^c \rightarrow a; d$ and $a^c \rightarrow \lambda$ is said to be *restricted* (we also use to say that such a system is a *standard* one).

The rules are applied as follows: if the neuron σ_i contains k spikes, $a^k \in L(E)$ and $k \geq c$, then the rule $E/a^c \rightarrow a^p; d \in R_i$ (with $p \geq 1$) is enabled and it can be applied; applying it means that c spikes are consumed, only $k-c$ remain in the neuron, the neuron is fired, and it produces p spikes after d time units. If $d=0$, then the spikes are emitted immediately, if $d=1$, then the spikes are emitted in the next step, and so on. In the case $d \geq 1$, if the rule is used in step t, then in steps $t, t+1, t+2, \ldots, t+d-1$ the neuron is *closed*, and it cannot receive new spikes (if a neuron has a synapse to a closed neuron and sends spikes along it, then the spikes are lost). In step $t+d$, the neuron spikes and becomes again open, hence can receive spikes (which can be used in step $t+d+1$). The p spikes emitted by a neuron σ_i are replicated and they go to all neurons σ_j such that $(i,j) \in syn$ (each σ_j receives p spikes). If the rule is a forgetting one, hence with $p=0$, then no spike is emitted (and the neuron cannot be closed, because also $d=0$).

In the synchronized mode, considered up to now in all SN P systems investigations except [8], a global clock is assumed, marking the time for all neurons, and in each time unit, in each neuron which can use a rule, a rule must be used. Because two rules $E_1/a^{c_1} \to a^{p_1}; d_1$ and $E_2/a^{c_2} \to a^{p_2}; d_2$ can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules can be applied in a neuron, and then one of them is chosen non-deterministically. Note that the neurons work in parallel (synchronously), but each neuron processes sequentially its spikes, using only one rule in each time unit.

The initial configuration of the system is described by the numbers $n_1, n_2, \ldots, n_m$ of spikes present in each neuron. During the computation, a configuration is described by both the number of spikes present in each neuron and by the state of the neuron, more precisely, by the number of steps to count down until it becomes open (this number is zero if the neuron is already open). Thus, $\langle r_1/t_1, \ldots, r_m/t_m \rangle$ is the configuration where neuron $\sigma_i, i = 1, 2, \ldots, m$ contains $r_i \geq 0$ spikes and it will be open after $t_i \geq 0$ steps; with this notation, the initial configuration is $C_0 = \langle n_1/0, \ldots, n_m/0 \rangle$ (see an example in Fig. 2).

Using the rules as suggested above, we can define transitions among configurations. Any sequence of transitions starting in the initial configuration is called a *computation*. A computation halts if it reaches a configuration where all neurons are open and no rule can be used. With any computation, halting or not, we associate a *spike train*, a sequence of digits 0 and 1, with 1 appearing in positions which indicate the steps when the output neuron sends spikes out of the system (we also say that the system itself spikes at that time). With any spike train, we can associate various numbers, which are considered as computed (generated) by the system; in the spirit of spiking neural computing, the distance between certain spikes are usually taken as the result of a computation (e.g., the distance between the first two spikes). Because of the non-determinism in using the rules, a given system computes in this way a set of numbers. An SN P system can be also used in the accepting mode: a number n is introduced in the system in the form of the distance between two spikes entering a specified neuron, and this number is accepted if the computation eventually halts.

We denote by $N_{\mathrm{gen}}(\Pi)$ the set of numbers generated (in the synchronized way) by a system Π in the form of the number of steps elapsed between the first two spikes of a spike train. Then by $Spik_2SP_m(rule_k, cons_p, forg_q, del_d)$ we denote the

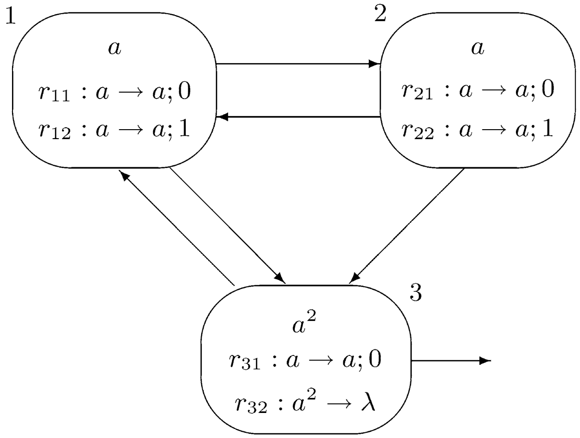


Fig. 1 The initial configuration of the SN P system Π

family of such sets of numbers generated by systems with at most m neurons, each of them containing at most k rules, all of them of the standard form, and each rule consuming at most p spikes, forgetting at most q spikes, and having the delay at most d. When using extended SN P systems, we use $Spik_2EP_m(rule_k, cons_p, prod_q, del_d)$ to denote the family of sets $N_{\text{gen}}(\Pi)$ generated by systems with at most m neurons, each of them containing at most k rules (of the extended form), each spiking rule consuming at most p spikes, producing at most q spikes, and having the delay at most d. When any of the parameters m, k, p, q, d is not bounded, it is replaced by $*$. When using the rules in the exhausting or the non-synchronized mode, we write $N_{\text{gen}}^{\text{ex}}(\Pi)$, $N_{\text{gen}}^{\text{nsyn}}(\Pi)$, respectively, and the superscripts *ex* and *nsyn* are also added to *Spik* in the families notation.

The notations should be changed when dealing with other sets of numbers than the distance between the first two spikes, with accepting systems, when generating or accepting languages, but we do not enter here into details. Instead, we close this section by introducing two important tools in presenting SN P systems, namely, the graphical representation and the transition diagram.

Figures 1 and 2 are recalled from [9]. The graphical representation of an SN P system is rather intuitive: the neurons are represented by membranes, placed in the nodes of a directed graph whose arrows represent the synapses; an arrow also exits from the output neuron, pointing to the environment; in each neuron, we specify the rules and the spikes present in the initial configuration.

Figure 1 represents the initial configuration of a system Π. We have three neurons, labeled with 1, 2, 3, with neuron σ_3 being the output one. Each neuron contains two rules, with neurons σ_1 and σ_2 having the same rules (firing rules which can be chosen in a non-deterministic way, the difference between them being in the delay from firing to spiking), and neuron σ_3 having one firing and one forgetting rule. In the figure, the rules are labeled, and these labels are useful below, in relation with Fig. 2.

This figure can be used for analyzing the evolution of the system Π: because the system is finite, the number of configurations reachable from the initial configura-

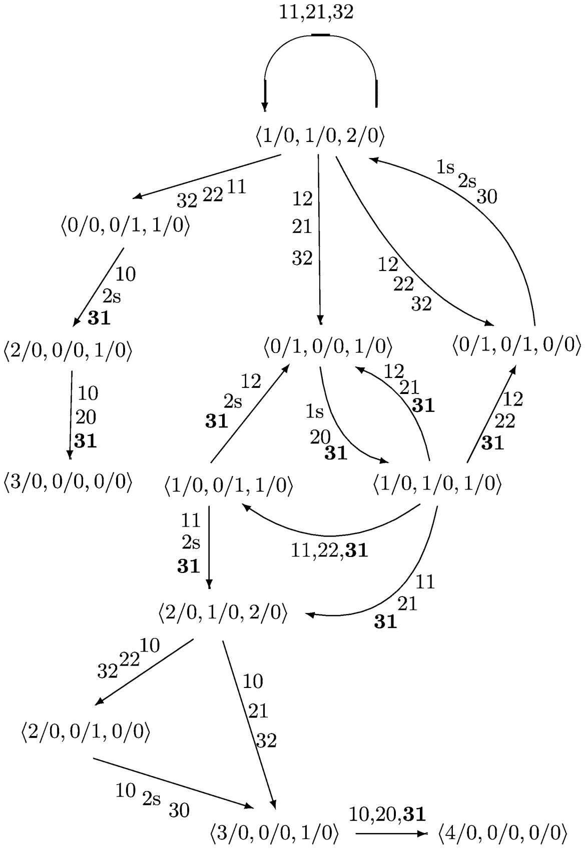


Fig. 2 The transition diagram of system Π from Fig. 1

tion is finite, too; hence, we can place them in the nodes of a graph, and between two nodes/configurations we draw an arrow if and only if a direct transition is possible between them. In Fig. 2, the rules are also indicated used in each neuron, with the following conventions: for each r_{jk} we have written only the subscript jk, with **31** being written in bold face, in order to indicate that a spike is sent out of the system at that step; when a neuron σ_j, $j = 1, 2, 3$ uses no rule, we have written $j0$, and when it spikes (after being closed for one step), we write js.

The functioning of the system, both as a number generator and as a string generator, can easily be followed on this diagram. The transition diagram is very useful as a tool involved in the formal verification of an SN P system. A way to automatically generate such a diagram is also a part of the software described in [47].

5 Open Problems and Research Topics

The following list of problems should be read with the standard precautions: it is not meant to be exhaustive, there is no ordering of the problems (according to their significance/interest), some problems are very general, others are much more particular, in many cases, the formulation is preliminary/informal and addressing the problem should start with a precise/suitable formulation, in many cases related results exist in the literature, and so on. Most problems are stated in a short way, with reference to the discussion from Sect. 3 and the definitions from Sect. 4.

A. Let us start with a general and natural idea: linking the study of SN P systems with neural computing. This can be a rich source of ideas, based on transferring from an area to the other one research topics which make sense also in the destination framework. What means, for instance, training (in general, learning, adaptation, evolving) in terms of SN P systems? More elementary: what means solving a problem by using an SN P system, implicitly, what means to solve a problem in a better way? Maybe the starting point should not be (only) neural computing, which is already an abstract, specialized, reductionistic framework, but (also) from neurology, from learning in the general psycho-pedagogical sense.

This problem is related to another general, natural, and important one: bringing more ingredients from neurology. Just a few quick ideas: considering an energy associated with firing/spiking; taking into consideration the anti-port processes which are performed in synapses; introducing Circadian periodicity in the functioning of neurons and of nets of neurons, with "tiredness", "resting periods", etc. How can a natural notion such as "“memory” captured in this framework (short-term, long-term memory, forgetting information, etc.)?

B. In particular, the recent discoveries related to the role of astrocytes in the functioning of the brain need to be examined and formalized. Astrocytes are a class of cells that form a supporting and insulating structure for the neurons, but also participate in the process of communication between neurons. They "listen" to the spikes passing along axons and accordingly regulate the release of neurotransmitters from the nerve terminals, thus relating in an intricate way the functioning of different neighboring axons. The regulation is either excitatory or inhibitory, and it is done by means of calcium waves. We refer to [46] and [50] for further details, and further references. How can astrocytes be considered in an SN P system and with what consequences?

An attempt in this respect is that from [3], where some preliminary (computability) results were obtained. Then a particular case, much simpler, was considered in [41] in the following setup. A further component of an SN P system is considered, $astro \subseteq sun^{\leq k}$; an element of this set is called *astrocyte*. The idea is that such an astrocyte controls a number t of axons (actually, synapses, because *syn* identifies synapses) less than or equal to a given constant k and, if a number of spikes are transmitted along the t axons, then only one of them is selected and let to go, all others are simply removed. Because exactly one spike is moved along the controlled axons, this can lead to deadlock situations, where several astrocytes controlling common axons cannot work together according to the previous definition. The occurrence of such a deadlock in SN P systems with astrocytes is proved in [41] to be undecidable. Another result proved in [41] concerns the possibility of passing from a system with astrocytes with an arbitrary *degree* (the constant k above) with an equivalent system having the minimal degree, two. For the case of generating numbers (in the sense of the set $N_{\mathrm{gen}}(\Pi)$ defined above), the answer is affirmative; the minimal degree can be reached.

Many issues are left open in [41]: changing the definition in order to avoid the deadlock; studying astrocytes of a more realistic type (for instance, controlling axons, not synapses); dealing also with unsynchronized systems, etc.

The neuron-astrocyte coupling is based on signaling pathways of a kind which reminds the controlling pathways which were recently modeled and simulated in terms of P systems in many papers, and this suggests the next general research challenge: applications (in neurology). This is perhaps a too ambitious goal at this stage of the development of the study of SN P systems and it is first necessary to have answers to the previous two problems, but it is important to keep in mind the possibility of applications when devising new classes of SN P systems. It is difficult to forecast which would be the most promising types of applications—looking for conceptual clarifications, for analytical results, for computer experiments and simulations, for all these intertwined? Of course, the cooperation with a biologist/neurologist would be very important in this respect.

Making a step from neurobiology to mathematics, the problem appears to consider systems using more than one type of spikes. At the first sight, this is against the spirit of spiking neural computing, and can lead to standard membrane systems. Still, the question makes sense in various setups. For instance, neurology deals both with excitatory and inhibitory impulses, both in neurons and at the level of astrocytes. How can inhibitory spikes be defined and used?

C. Then there are features of SN P systems which were not considered for general P systems. Using a regular expression for enabling a rule looks like controlling the application of rules by means of promoters, inhibitors, and activators, but a notion of delay does not exits in membrane computing. Can it be of any interest also for usual P systems? Then defining the result of a computation in a P system in terms of the time elapsed between two specified events, in particular, sending a given object outside, was briefly investigated in [6], but this issue deserves further research efforts.

Conversely, there are many ingredients of usual P systems which were not considered for SN P systems and might make sense also in this area, at least at a mathematical level. Of a particular interest can be tools to exponentially increase the working space in a polynomial (if possible, even linear) time, for instance, by operations similar to cell division and cell creation in P systems with active membranes. How new neurons can be created (added to a system) in such a way to make possible polynomial solutions to computationally hard (typically, **NP**-complete) problems? The brain is supposed to be a very efficient computing device—how can SN P systems be made efficient from this point of view?

D. This touches a more general issue, that of considering SN P systems with a dynamical structure. The dynamism can be achieved both in terms of neurons and synapses, or only for synapses. From birth to maturity, the brain essentially evolves at the level of synapses, learning means establishing new synapses, cutting them, making them more stable/fast when used frequently, and so on and so forth. How this can be incorporated in SN P systems? A related idea is to associate a duration to each synapse (which is not of interest when the duration is constant), and to vary it in time, according to the intensity of using that synapse, and this looks rather motivated from a learning point of view.

Making synapses to have a duration or a length, depending on their use, can be related to a similar idea [16] at the level of spikes: considering a duration of life also for spikes, in the form of a decaying constant associated with them (at the level of the whole system, or locally, for each neuron). If a spike is not used, a number of steps larger than the decaying threshold, then it is removed (a sort of forgetting rules are thus implicitly acting, depending on the age of each spike).

E. Moving further to theoretical issues, let us consider an idea related both to "classic" membrane computing and to the efficiency issue: using the rules in a parallel manner. This has been already considered in [27] in the particular form of using the rules in the exhaustive mode: if a neuron contains $kn + r$ spikes and has a rule $E/a^n \rightarrow a; d$ such that $a^{kn+r} \in L(E)$ and $k \geq 1, 0 \leq r < n$, then the rule is enabled and it is applied k times; kn spikes are consumed, r remain unused, and k are produced. Besides continuing the research from [27] (where it is only proved that SN P systems with an exhaustive use of rules are Turing complete both in the generative and the accepting modes), several other problems remain to be investigated. Actually, most problems usually considered for SN P systems with a sequential use of rules can be formulated also for the exhaustive mode: generating or accepting languages, translating strings of infinite sequences, looking for small universal systems, etc.

Then the problem arises to consider other forms of parallelism at the level of each neuron or at the level of the whole system. What about using several rules at the same time in the same way as the rules of a usual P system are applied in the maximally parallel manner? Variants inspired from grammar systems area can also be considered, thus obtaining a bounded parallelism: at least k, at most k, exactly k rules to be used at a time. This last idea can be transferred also at the level of neurons: in each step, only a prescribed number of neurons, non-deterministically chosen, to be active. Finally, one can borrow to this area the idea of minimal parallelism from [15]: when a neuron can use at least one rule, then at least one must be used, without any restriction about how many. Similarly, we can extend this to the whole system or to pre-defined blocks of the system: if at least one neuron from a block can fire, then at least one should do it, maybe more. A significant non-determinism is introduced in this way in the functioning of the system.

F. When the number of rules to be used in each neuron is "at least zero" (and this is equivalent with making evolve "at least zero" neurons at a time), we get the rather natural idea of a non-synchronized functioning of an SN P system. In such a case, in each time unit, any neuron is free to use a rule or not.

We have described the functioning of such a system in the end of Sect. 3. We only recall that because now “the time does not matter”, the spike train can have arbitrarily many occurrences of 0 between any two occurrences of 1, hence the result of a computation can no longer be defined in terms of the steps between two consecutive spikes, but as the total number of spikes sent into the environment by (or contained in) the output neuron. In this way, only halting computations can be considered as successful.

In [8], it is proved that SN P systems with *extended* rules are Turing equivalent even in the non-synchronized case, but the problem was left open whether this is true also for systems using standard rules. The conjecture is that this does not happen, hence that synchronization plays a crucial role in this case. If true, such a result would be of a real interest.

Similar to the exhaustive mode of using rules, also the non-synchronization can be investigated in relation with many types of problems usual in the SN P systems area: handling languages, looking for small universal systems, etc.

A related issue is to consider the class of systems for which the synchronization does not matter, i.e., they generate/accept the same set of numbers in both modes. Furthermore, time-free, clock-free, time-independent systems can be considered, in the same way as in [5, 7, 48].

G. Several times so far, the idea of efficiency was invoked, with the need to introduce new ingredients in the area of SN P systems in such a way to make possible polynomial solutions to intractable problems. Actually, such a possibility was already considered in [10]: making use of arbitrarily large pre-computed resources. The framework is the following: an arbitrarily large net of neurons is given of a regular form (as the synapse graph) and with only a few types of neurons (as contents and rules) repeated indefinitely; the problem to be solved is plug-in by introducing a polynomial number of spikes in certain neurons (of course, polynomially many), then the system is left to work autonomously; in a polynomial time, it activates an exponential number of neurons, and after a polynomial time, it outputs the solution to the problem. The problem considered in [10] was the SAT problem.

This strategy is attractive from a natural computing point of view (we may assume that the brain is arbitrarily large with respect to the small number of neurons currently used, the same with the cells in liver, etc.), but it has no counterpart in the classic complexity theory. A formal framework for defining acceptable solutions to problems by making use of pre-computed resources needs to be formulated and investigated. What kind of pre-computed workspace is acceptable, i.e., how much information may be provided for free there, what kind of net of neurons, and what kind of neurons? (We have to prevent “cheating” by already placing the answer to the problem in the given resources and then “solving” the problem just by visiting the right place where the solution waits to be read.) What means introducing a problem in the existing device? (Only spikes, also rules, or maybe also synapses?) Defining complexity classes in this case remains as an interesting research topic.

In fact, SN P systems contains an in-built ingredient which makes them intrinsically efficient: by definition, the use of a rule takes one time unit; however, using a rule $E/a^c \to a; 0$ means (i) checking whether or not the neuron is covered by the regular expression E, (i) removing c spikes, and (iii) producing one spike. Step (i) assume solving the membership problem for a regular expression in constant time, in one step, which is not as known for regular languages, whose membership problem is of a linear complexity (the parsing time is proportional with the length of the parsed string). This means that we tacitly introduced an oracle, of a rather simple form—a regular set, but still bringing a considerable speed-up. Details can be found in [31, 32], where it is also proved that in certain cases this oracle does not help, a deterministic SN P system with particular regular expressions can be simulated in polynomial time by a deterministic Turing machine.

In the above mentioned papers, one also address another interesting issue: solving decidability problems in constant time, *in a non-deterministic* way. This possibility is illustrated with solutions to SAT and Subset-Sum. Uniform solutions (still non-deterministic) to these problems are provided in [30].

Anyway, the complexity investigations in the SN P systems area need and deserve further efforts. Defining complexity classes (for deterministic or non-deterministic systems, with or without pre-computed resources), clarifying the role of "oracles" involved in applying the spiking rules (the brain seems to have such capabilities, e.g., when recognizing patterns), improving and extending the results from [30–32], ways to generate an exponential working space, other ideas inspired from neurobiology are only a few topics to explore.

H. Coming back to the initial definitions, there are several technical issues which are worth clarifying (most probably, for universality and maybe also for efficiency results, they do not matter, but it is also possible to exist other situations where these details matter). For instance, the self-synapses are not allowed in the synapse graph. However, a neuron with a rule $a \to a$ and a self-synapse can work forever, hence it can be used for rejecting a computation in the case when successful computations should halt. Similarly, (in the initial definition from [26]) the forgetting rules $a^s \to \lambda$ were supposed to have $a^s \notin L(E)$ for all spiking rules $E/a^c \to a; d$ from the same neuron, while in extended rules $E/a^c \to a^p; d$ it was assumed that $c \geq p$. Is there any situation where these restrictions make a difference? Then in [21] it was shown that some of the ingredients used in the definition of SN P systems with standard rules can be avoided. This is the case with the delay, the forgetting rules, the generality of regular expressions. Can these normal forms be combined, thus avoiding at the same time two of the mentioned features?

What then about using a kind of rules of a more general form, namely $E/a^n \to a^{f(n)}; d$, where f is a partial function from natural numbers to natural numbers (maybe with the property $f(n) \leq n$ for all n for which f is defined), and used as follows: if the neuron contains k spikes such that $a^k \in L(E)$, then c of them are consumed and $f(c)$ are created, for $c = \max\{n \in \mathbf{N} \mid n \leq k, \text{ and } f(n) \text{ is defined}\}$; if f is defined for no n smaller than or equal to k, then the rule cannot be applied. This kind of rules looks both adequate from a neurobiological point of view (the sigmoid excitation function can be captured) and powerful from a mathematical point of view (arbitrarily many spikes can be consumed at a time, and arbitrarily many produced).

J. A standard problem when dealing with accepting devices concerns the difference between deterministic and non-deterministic systems. Are they different in power, does determinism imply a decrease of the computing power? Up to now, all computability completeness proofs for the accepting version of SN P systems of various types were obtained for deterministic systems. Are there classes (maybe non-universal) for which the determinism matters?

Actually, the problem can be refined. The determinism is defined usually in terms of non-branching during computations: a computation is deterministic if for every configuration there is (at most) one next configuration. A first subtle point: is this requested for *all* possible configurations or only for all configurations which are *reachable* from the initial one?

Maybe more interesting for SN P systems is the possibility to define a *strong determinism*, in terms of rules: an SN P system is said to be strongly deterministic if $L(E) \cap L(E') = \emptyset$ for all rules $E/a^c \to a; d$ and $E'/a^{c'} \to a; d'$ from any neuron. Obviously, such a system is deterministic also when defining this notion in terms of branching (even for arbitrary configurations, not only for the reachable ones).

Is any class of SN P systems for which these types of determinism are separated?

K. Different from the case of general P systems, where finding infinite hierarchies on the number of membranes was a long awaited result, for SN P systems one can easily find such hierarchies, based on the characterization of semi-linear sets of numbers (by means of systems with a bounded number of spikes in their neurons): if for each finite automaton with n states (using only one symbol) one can find an equivalent SN P system with $g(n)$ neurons, and, conversely, for each SN P system with m neurons one can find an equivalent (i.e., generating strings over an one-letter alphabet whose lengths are numbers generated/accepted by the SN P system) with $h(m)$ states, then because there is an infinite hierarchy of regular one-letter languages in terms of states, we get an infinite hierarchy of sets of numbers with respect to the number of neurons. Still, several problems arise here. First, not always the characterization of semi-linear sets of numbers is based on proving the equivalence of bounded SN P systems with the finite automata. Then this reasoning only proves that the hierarchy is infinite, not also that it is "dense" (*connected* is the term used in classic descriptional complexity: there is n_0 such that for each $n \geq n_0$ there is a set Q_n whose neuron-complexity is exactly n). Finally, what about finding classes intermediate between semi-linear and Turing computable for which the hierarchy on the number of neurons is infinite (maybe connected)?

The previous question directly suggests two others. The first one is looking for small universal SN P systems (here "universal" is understood in the sense of "programmable"—the existence of a fixed system which can simulate any particular system after introducing a code of the particular system in it—not in the sense of "Turing complete", although there is a direct connection between these two notions). This question is considered in [37] for SN P systems with standard and with extended rules, working either in the computing mode or in the generating mode. For standard rules, 84 and 76 neurons were used, while for extended rules 49 and 50 neurons were used, respectively. Are these results optimal? A negative answer is expected (however, a significant improvement is not very probable). What about universal SN P systems of other types—in particular, with exhaustive or non-synchronized use of rules?

L. Problem **K** also suggests to look for classes of SN P systems which are not equivalent with Turing machines, but also not computing only semi-linear sets of numbers, hence equivalent in power with finite automata. This does not look as an easy question, but it is rather interesting, in view of the possibility of finding classes of systems with decidable properties, but (significantly) more powerful than bounded SN P systems. Such a class would be attractive also from the point of view of applications, because of the possibility of finding properties of the modeled processes by analytical, algorithmic means.

Again in a direct continuation with the previous issue, there appears the need to find characterizations of classes of languages, other than finite, regular, and recursively enumerable, in terms of SN P systems. The investigations from [9, 12, 14] have left open these questions, and this fits with the general situation in membrane computing (as well as in DNA computing): the Chomsky hierarchy seems not to have a counterpart in nature, families like those of linear, context-free, and context-sensitive languages do not have (easy) characterizations in bio-inspired computing models. The same challenge appears for families of languages generated by L systems (sometimes, with the exception of ET0L languages).

L systems can be related with SN P systems also at the level of infinite sequences: both by iterating morphisms (D0L systems) and by taking infinite spike trains we can get classes of infinite sequences. Directly as spike trains we have binary sequences, but for extended rules (and for SN P systems with a parallel use of rules) we can get as an output of a computation a string or an infinite sequence over an arbitrary alphabet. A preliminary examination of the binary case was done in [43], but many problems were left open, starting with the comparison of SN P systems as tools for handling infinite sequences (of bits) with other tools from language and automata theory (with ω-languages computed by finite automata, Turing machines, etc.) and with known infinite sequences, e.g., those from [52].

A particular problem from [43] is the following. SN P systems cannot compute arbitrary morphisms, but only length preserving morphisms (codes). An extension of these latter functions are the so-called *k-block morphisms*, which are functions $f : \{0, 1\}^k \longrightarrow \{0, 1\}^k$ (for a given $k \geq 1$) prolonged to $f : \{0, 1\}^\omega \longrightarrow \{0, 1\}^\omega$ by $f(x_1 x_2 \ldots) = f(x_1) f(x_2) \ldots$. In [43], it is only shown that 2-block morphisms can be computed by SN P systems, and the conjecture was formulated that this is true for any k.

In general, more should be found about the use of SN P systems as tools for transducing strings and infinite sequences.

Maybe useful in addressing the previous problem—and interesting also from other points of view (e.g., if starting investigations in terms of process algebra), is the issue of compositionality: looking for ways to pass from given systems to more complex systems, for instance, to systems generating/accepting the result of an operation between the sets of numbers or the languages generated/accepted by the initial systems. Morphisms were mentioned also above, but there are many other set-theoretic or language-theoretic operations to consider, as well as serial and parallel composition, embedding as a subsystem, etc. Of course, a central point in such operations is that of synchronization. It is expected that the case of non-synchronized systems is much easier (maybe, instead, less interesting theoretically).

M. We have mentioned at the beginning of these notes that the axon is not a simple transmitter of spikes, but a complex information processor. This suggests considering computing models based on the axon functioning (Ranvier nodes amplification of impulses, and other processes) and a preliminary investigation was carried out in [13]. Many questions remain to be clarified in this area (see also the questions formulated in [13]), but a more general and probably more interesting problem appears, namely, of combining neurons and axons (as information processing units) in a global model; maybe also astrocytes can be added, thus obtaining a more complex model, closer to reality.

N. We will conclude with two general issues, where nothing was done up to now. First, SN P systems have a direct (pictural) similarity with Petri nets, where tokens (like spikes) are moved through the net according to specific rules. Bridging the two areas looks then rather natural—with "bridging" understood as a move of notions, tools, results in both directions, from Petri nets to SN P systems and the other way round.

Then directly important for possible applications is the study of SN P systems as dynamical systems, hence not focusing on their output, but on their evolution, on the properties of the sequences of configurations reachable from each other. The whole panoply of questions from the (discrete) dynamical systems theory can be brought here, much similar to what happened in general membrane computing.

6 Final Remarks

Many other open problems and research topics can be found in the papers devoted to SN P systems—the interested reader can check the titles below in this respect (the bibliography contains most of the papers about SN P systems which we were aware of at the beginning of December 2007). On the other hand, because the research in this area is quite vivid, it is possible that some of these problems were solved at the same time or shortly after writing these notes, without being possible to mention the respective results here. That is why, the reader is advised to follow the developments in this area, for instance, through the information periodically updated at the membrane computing web page [54]. In particular, one can find there the paper [40], on which the present paper is based.

Acknowledgements Work supported by Projects BioMAT 2-CEx06-11-97/19.09.06 and CellSim PC-1284 (Gh. Păun), and Project TIN 2006-13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the Project of Excellence TIC-581 of Junta de Andalucia (M.J. Pérez-Jiménez).

References

1. Alberts B, Johnson A, Lewis J, Raff M, Roberts K, Walter P (2002) Molecular biology of the cell, 4th edn. Garland Science, New York
2. Alhazov A, Freund R, Oswald M, Slavkovik M (2006) Extended variants of spiking neural P systems generating strings and vectors of non-negative integers. In: Hoogeboom HJ, Păun Gh, Rozenberg G, Salomaa A (eds) Membrane computing, international workshop, WMC7, revised, selected, and invited papers, Leiden, The Netherlands. Lecture notes in computer science, vol 4361. Springer, Berlin, pp 123–134
3. Binder A, Freund R, Oswald M, Vock L (2007) Extended spiking neural P systems with excitatory and inhibitory astrocytes. In: Gutiérrez-Naranjo MA et al (eds) Proceedings of fifth brainstorming week on membrane computing. Fenix Editora, Sevilla, pp 63–72
4. Carnell A, Richardson D Parallel computation in spiking neural nets. Available at http://people.bath.ac.uk/masdr/
5. Cavaliere M, Deufemia V (2006) On time-free P systems. Int J Found Comput Sci 17(1):69–90
6. Cavaliere M, Freund R, Leitsch A, Păun Gh (2005) Event-related outputs of computations in P systems. In Proceedings of the third brainstorming week on membrane computing, Sevilla. RGNC report 01/2005, pp 107–122
7. Cavaliere M, Sburlan D (2005) Time-independent P systems. In: Membrane computing. International workshop WMC5, Milano, Italy, 2004. Lecture notes in computer science, vol 3365. Springer, Berlin, pp 239–258
8. Cavaliere M, Egecioglu E, Ibarra OH, Ionescu M, Păun Gh, Woodworth S (2009) Asynchronous spiking neural P systems Theor Comp Sci (in press)
9. Chen H, Freund R, Ionescu M, Păun Gh, Pérez-Jiménez MJ (2007) On string languages generated by spiking neural P systems. Fund Inform 75(1–4):141–162
10. Chen H, Ionescu M, Ishdorj T-O (2006) On the efficiency of spiking neural P systems. In: Gutiérrez-Naranjo MA et al (eds) Proceedings of fourth brainstorming week on membrane computing, vol I. Fenix Editora, Sevilla, pp 195–206; and in: Proceedings of the 8th international conference on electronics, information, and communication, Ulanbator, Mongolia, June 2006, pp 49–52
11. Chen H, Ionescu M, Ishdorj T-O, Păun A, Păun Gh, Pérez-Jiménez MJ Spiking neural P systems with extended rules: universality and languages. Nat Comput 7(2):147–166
12. Chen H, Ionescu M, Păun A, Păun Gh, Popa B (2006) On trace languages generated by spiking neural P systems. In: Gutiérrez-Naranjo MA et al (eds) Proceedings of fourth brainstorming week on membrane computing, vol I. Fenix Editora, Sevilla, pp 207–224; and in: Proceedings of the eighth international workshop on descriptional complexity of formal systems (DCFS 2006), Las Cruces, NM, USA, 21–23 June 2006, pp 94–105
13. Chen H, Ishdorj T-O, Păun Gh (2007) Computing along the axon. Prog Nat Sci 17(4):418–423
14. Chen H, Ishdorj T-O, Păun Gh, Pérez-Jiménez MJ (2006) Spiking neural P systems with extended rules. In: Gutiérrez-Naranjo MA et al (eds) Proceedings of fourth brainstorming week on membrane computing, vol I. Fenix Editora, Sevilla, pp 241–265
15. Ciobanu G, Pan L, Păun Gh, Pérez-Jiménez MJ (2007) P systems with minimal parallelism. Theor Comput Sci 378(1):117–130
16. Freund R, Ionescu M, Oswald M (2007) Extended spiking neural P systems with decaying spikes and/or total spiking. In: ACMC/FCT 2007 workshop, Budapest
17. Gerstner W, Kistler W (2002) Spiking neuron models. Single neurons, populations, plasticity. Cambridge University Press, Cambridge
18. Gutiérrez-Naranjo MA et al (eds) (2006) Proceedings of fourth brainstorming week on membrane computing. Fenix Editora, Sevilla
19. Gutiérrez-Naranjo MA et al (eds) (2007) Proceedings of fifth brainstorming week on membrane computing. Fenix Editora, Sevilla
20. Hoogeboom HJ, Păun Gh, Rozenberg G, Salomaa A (eds) (2006) Membrane computing, international workshop, WMC7, revised, selected, and invited papers, Leiden, The Netherlands, 2006. Lecture notes in computer science, vol 4361. Springer, Berlin

21. Ibarra OH, Păun A, Păun Gh, Rodríguez-Patón A, Sosik P, Woodworth S (2007) Normal forms for spiking neural P systems. Theor Comput Sci 372(2–3):196–217
22. Ibarra OH, Woodworth S (2007) Spiking neural P systems: some characterizations. In: Proceedings of the FCT 2007, Budapest. Lecture notes in computer science, vol 4639. Springer, Berlin, pp 23–37
23. Ibarra OH, Woodworth S (2007) Characterizing regular languages by spiking neural P systems. Int J Found Comput Sci 18(6):1247–1256
24. Ibarra OH, Woodworth S, Yu F, Păun A (2006) On spiking neural P systems and partially blind counter machines. In: Proceedings of the UC2006, York. Lecture notes in computer science, vol 4135. Springer, Berlin, pp 113–129
25. Ionescu M, Păun A, Păun Gh, Pérez-Jiménez MJ (2006) Computing with spiking neural P systems: traces and small universal systems. In: Mao C, Yokomori Y, Zhang B-T (eds) Proceedings of the DNA12, Seoul, June 2006, pp 32–42
26. Ionescu M, Păun Gh, Yokomori T (2006) Spiking neural P systems. Fund Inform 71(2–3):279–308
27. Ionescu M, Păun Gh, Yokomori T (2007) Spiking neural P systems with exhaustive use of rules. Int J Unconv Comput 3(2):135–154
28. Ionescu M, Sburlan D Several applications of spiking neural P systems. In: Proceedings of the WMC8, Thessaloniki, June 2007, pp 383–394
29. Korec I (1996) Small universal register machines. Theor Comput Sci 168:267–301
30. Leporati A, Mauri G, Zandron C, Păun Gh, Pérez-Jiménez MJ (2007) Uniform solutions to SAT and subset sum by spiking neural P systems (submitted)
31. Leporati A, Zandron C, Ferretti C, Mauri G (2007) Solving numerical NP-complete problems with spiking neural P systems. In: Eleftherakis G, Kefalas P, Păun Gh, Rozenberg G, Salomaa A (eds) Membrane computing, international workshop, WMC8, selected and invited papers, Thessaloniki, Greece. Lecture notes in computer science, vol 4860. Springer, Berlin
32. Leporati A, Zandron C, Ferretti C, Mauri G (2007) On the computational power of spiking neural P systems. Int J Unconv Comput 5(5)
33. Maass W (2002) Computing with spikes. Found Inform Process TELEMATIK 8(1):32–36 (special issue)
34. Maass W (2002) Paradigms for computing with spiking neurons. In: van Hemmen JL, Cowen JD, Domany E (eds) Models of neural networks. Early vision and attention. Springer, Berlin, pp 373–402
35. Maass W, Bishop C (eds) (1999) Pulsed neural networks. MIT Press, Cambridge
36. O'Dwyer C, Richardson D (eds) Spiking neural nets with symbolic internal state. Available at http://people.bath.ac.uk/masdr/
37. Păun A, Păun Gh (2007) Small universal spiking neural P systems. Biosystems 90(1):48–60
38. Păun Gh (2002) Membrane computing—an introduction. Springer, Berlin
39. Păun Gh (2006) Languages in membrane computing. Some details for spiking neural P systems. In: Proceeding of developments in language theory conference, DLT 2006, Santa Barbara, June 2006. Lecture notes in computer science, vol 4036. Springer, Berlin, pp 20–35
40. Păun Gh (2007) Twenty six research topics about spiking neural P systems. In: Gutiérrez-Naranjo MA et al (eds) Proceedings of fifth brainstorming week on membrane computing. Fenix Editora, Sevilla, pp 263–280
41. Păun Gh (2007) Spiking neural P systems with astrocyte-like control. J Univ Comput Sci 13(11):1707–1721
42. Păun Gh, Pérez-Jiménez MJ, Rozenberg G (2006) Spike trains in spiking neural P systems. Int J Found Comput Sci 17(4):975–1002
43. Păun Gh, Pérez-Jiménez MJ, Rozenberg G (2006) Infinite spike trains in spiking neural P systems (submitted)
44. Păun Gh, Pérez-Jiménez MJ, Salomaa A (2006) Bounding the indegree of spiking neural P systems. TUCS technical report 773
45. Păun Gh, Pérez-Jiménez MJ, Salomaa A (2007) Spiking neural P systems. An early survey. Int J Found Comput Sci 18:435–456

46. Perea G, Araque A (2002) Communication between astrocytes and neurons: a complex language. J Physiol (Paris) 96:199–207
47. Ramírez-Martínez D, Gutiérrez-Naranjo MA (2007) A software tool for dealing with spiking neural P systems. In: Gutiérrez-Naranjo MA et al (eds) Proceedings of fifth brainstorming week on membrane computing. Fenix Editora, Sevilla, pp 299–312
48. Sburlan D (2006) Promoting and inhibiting contexts in membrane computing. PhD thesis, University of Sevilla, Spain
49. Segev I, Schneidman E (1999) Axons as computing devices: basic insights gained from models. J Physiol (Paris) 93:263–270
50. Shen X, De Wilde P (2009) Long-term neuronal behavior caused by two synaptic modification mechanisms. Neurocomputing (to appear)
51. Shepherd GM (1994) Neurobiology. Oxford University Press, Oxford
52. Sloane NJA, Plouffe S (1995) The encyclopedia of integer sequences. Academic Press, New York
53. Wang J, Pan L (2007) Excitatory and inhibitory spiking neural P systems. Manuscript
54. The P systems web page. http://ppage.psystems.eu

Membrane Computing Schema: A New Approach to Computation Using String Insertions

Mario J. Pérez-Jiménez and Takashi Yokomori

Abstract In this paper, we introduce the notion of a membrane computing schema for string objects. We propose a computing schema for a membrane network (i.e., tissue-like membrane system) where each membrane performs unique type of operations at a time and sends the result to others connected through the channel. The distinguished features of the computing models obtained from the schema are:

1. only *context-free insertion operations* are used for string generation,
2. some membranes assume filtering functions for *structured objects* (*molecules*),
3. generating model and accepting model are obtained in the *same schema*, and both are computationally universal,
4. several known rewriting systems with universal computability can be reformulated by the membrane computing schema in a uniform manner.

The first feature provides the model with a simple uniform structure which facilitates a biological implementation of the model, while the second feature suggests further feasibility of the model in terms of DNA complementarity.

Through the third and fourth features, one may have a unified view of a variety of existing rewriting systems with Turing computability in the framework of membrane computing paradigm.

1 Introduction

In the theory of bio-inspired computing models, membrane systems (or P systems) have been widely studied from various aspects of the computability such as the optimal system designs, the functional relations among many ingredients in different levels of computing components, the computational complexity and so forth. Up to the present, major concerns are focused on the computational capability of multisets of certain objects in a membrane structure represented by a rooted tree, and there are a relatively limited amount of works in the membrane structure of other types (like a network or graph) on string objects and their languages: those are,

M.J. Pérez-Jiménez (✉)
Research Group on Natural Computing, Department of Computer Science and Artificial Intelligence, University of Sevilla, Avda Reina Mercedes s/n, 41012 Sevilla, Spain
e-mail: marper@us.es

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_16, © Springer-Verlag Berlin Heidelberg 2009

for example, in the context of P system on graph structure [15], of the tissue P systems [11], and of spiking neural P systems [3, 7].

On the other hand in DNA computing theory, a string generating device called insertion-deletion system has been proposed and investigated from the unique viewpoint of non-rewriting nature in generating string objects [10, 14]. Among others, string insertion operation with no context is of our particular interests, because of the relevance to biological feasibility in terms of DNA sequences.

In this paper, we are concerned with tissue-like membrane systems with string insertion operations and investigate the computational capability of those systems. By using the framework of tissue-like membrane systems, however, our major focus is on studying the new aspects of the computational mechanisms used in a variety of existing models based on string rewriting.

To this aim, we propose the notion of a *membrane computing schema* which provides a unified view and framework to investigate new aspects of the variety of computational mechanisms. More specifically, let M be a given computing device (grammar or machine) based on string manipulation. Then by a membrane computing schema Π, we represent the *core* structure of M in question. At the same time, we also consider an interpretation I to Π which specifies the *details* of M. In this manner, we are able to have M that is embodied as a tissue-like membrane system $I(\Pi)$ with string insertion operation.

The advantages of this schematic approach to computing are the following:

(1) High transparency of the computing mechanism is obtained by separating the skeletal (core) part from other detailed specificity of the computation.

(2) Structural modularity of the computing model facilitates our better understanding of the computing mechanism.

With this framework, we will present not only new results of the computing models with universal computability but also a unified view of those models from the framework of tissue-like membrane system with string insertion operations.

2 Preliminaries

We assume the reader to be familiar with all formal language notions and notations in standard use. For unexplained details, consult, e.g., [14, 16].

For a string x over an alphabet V (i.e., x in V^*), $lg(x)$ denotes the length of x. For the empty string, we denote it by λ. For an alphabet V, $\overline{V} = \{\overline{a} \mid a \in V\}$. A binary relation ρ over V is called an *involution* if ρ is injective and ρ^2 is an identity (i.e., for any $a \in V$, if we write $\rho(a) = \overline{a}$, then it holds that $\rho(\overline{a}) = a$). A *Dyck* language D over V is a language generated by a context-free grammar $G = (\{S\}, V, P, S)$, where $P = \{S \rightarrow SS, S \rightarrow \lambda\} \cup \{S \rightarrow aS\overline{a} \mid a \in V\}$ and k is the cardinality of V.

An *insertion system* [10] is a triple $\gamma = (V, P, A)$, where V is an alphabet, A is a finite set of strings over V called axioms, and P is a finite set of insertion rules. An *insertion rule* over V is of the form (u, x, v), where $u, x, v \in V^*$. We define the relation $\mapsto$ on V^* by $w \mapsto z$ iff $w = w_1uvw_2$ and $z = w_1uxvw_2$ for some insertion rule $(u, x, v) \in P$, $w_1, w_2 \in V^*$. As usual $\mapsto^*$ denotes the reflexive and transitive closure of $\mapsto$. An *insertion language* generated by γ is defined as follows: $L(\gamma) = \{w \in V^* \mid s \mapsto^* w, s \in A\}$. An insertion rule of the form (λ, x, λ) is said to be *context-free*, and we denote it by $\lambda \to x$.

We denote by *RE*, *CF*, *LIN*, and *RG* the families of recursively enumerable languages, of context-free languages, of linear languages, and of regular languages, respectively.

A *matrix grammar with appearance checking* is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets, $S \in N$, M is a finite set of sequences of the form $(A_1 \to x_1, \dots, A_n \to x_n)$, $n \geq 1$, of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases), and F is a set of occurrences of rules in M (we say that N is the non-terminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called matrices).

For $w, z \in (N \cup T)^*$, we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \dots, A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*$, $1 \leq i \leq n+1$, such that $w = w_1$, $z = w_{n+1}$, and for all $1 \leq i \leq n$, either $w_i = w_i' A_i w_i''$, $w_{i+1} = w_i' x_i w_i''$, for some $w_i', w_i'' \in (N \cup T)^*$, or $w_i = w_{i+1}$, A_i does not appear in w_i, and the rule $A_i \to x_i$ appears in F. (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied; we say that these rules are applied in the *appearance checking* mode.) If $F = \emptyset$, then the grammar is said to be without appearance checking (and F is no longer mentioned).

We denote by $\Longrightarrow^*$ the reflexive and transitive closure of the relation $\Longrightarrow$.

The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac}. When we use only grammars without appearance checking, then the obtained family is denoted by *MAT*. It is known that $MAT \subset MAT_{ac} = RE$.

A matrix grammar $G = (N, T, S, M, F)$ is said to be in the *binary normal form* if $N = N_1 \cup N_2 \cup \{S, \#\}$, with these three sets mutually disjoint, and the matrices in M are of one of the following forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$,
2. $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in N_2 \cup N_2^2 \cup T \cup \{\lambda\}$,
3. $(X \to Y, A \to \#)$, with $X, Y \in N_1, A \in N_2$,
4. $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2$, and $x \in T \cup \{\lambda\}$.

Moreover, there is only one matrix of type 1 and F consists exactly of all rules $A \to \#$ appearing in matrices of type 3; # is a trap-symbol, once introduced, it is never removed. A matrix of type 4 is used only once, at the last step of a derivation. A matrix of type 3 is called *appearance checking matrix rule*.

For each matrix grammar (with appearance checking) there effectively exists an equivalent matrix grammar (with appearance checking) in the binary normal form. (Note that the definition of the binary normal form presented here is a variant of the one in [5].)

A *random context grammar* is a construct $G = (N, T, S, P)$, where N, T are disjoint alphabets, $S \in N$, P is a finite set of rules of the form $(A \to x, Q, R)$, where $A \to x$ is a context-free rule $(A \in N, x \in (N \cup T)^*)$, Q and R are subsets of N.

For $\alpha, \beta \in (N \cup T)^*$, we write $\alpha \Longrightarrow \beta$ iff $\alpha = uAv$, $\beta = uxv$ for some $u, v \in (N \cup T)^*$, $(A \rightarrow x, Q, R) \in P$ and all symbols of Q appear in uv, and no symbol of R appears in uv. We denote by $\Longrightarrow^*$ the reflexive and transitive closure of the relation $\Longrightarrow$.

The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form is denoted by *RC*. It is known that *RC* = *RE* [5].

Note that in what concerns the definitions above for matrix and random context grammars, we only deal with the type 2 (context-free) grammar as the core grammar.

3 Membrane Computing Schema, Interpretation and Languages

We now introduce the notion of a membrane computing schema in a general form, then we will present a restricted version, from which a variety of specific computing models based on insertion operations and filtering can be obtained in the framework of a tissue-like membrane computing. That is, a membrane computing schema is given as a skeletal construct consisting of a number of *membranes* connected with synapses (or channels) whose structure may be taken as a kind of *tissue P systems* (e.g., [11]).

3.1 Membrane Computing Schema

A *membrane computing schema* of degree $(\mathtt{p}, \mathtt{k}, \mathtt{t})$ is a construct

$$\Pi = (V, T, Com, Ope, Fil, Syn, i_s, Out),$$

where

- V is a finite alphabet with an involution relation ρ called the *working alphabet*.
- T is a subset of V called the *terminal alphabet*.
- $Com = \{Com_1, \ldots, Com_{\mathtt{p}}\}$ is a finite set with $\mathtt{p}$ elements, called *communication cells*.
- $Ope = \{Ope_1, \ldots, Ope_{\mathtt{k}}\}$ is a finite set with $\mathtt{k}$ elements, called *operation cells*.
- $Fil = \{SF, FF\} \cup \{FF_1, \ldots, FF_{\mathtt{t}}\}$ is a finite set with $(\mathtt{t} + 2)$ elements, called *filtering cells*. (More specifically, *SF* and *FF* are called *structured filter* and *final filter*, respectively, and $FF_1, \ldots, FF_{\mathtt{t}}$ are called *sub-filter cells*).
- *Syn* is a subset of $(Com \times Ope) \cup (Ope \times Com) \cup (Com \times (SubFil \cup \{SF\})) \cup (\{FF\} \times Com) \cup (SubFil \times Ope) \cup \{(SF, FF), (FF, Out)\}$, where $SubFil = \{FF_1, \ldots, FF_{\mathtt{t}}\}$, which determines the tissue membrane structure of Π. (See Fig. 1.)
- i_s $(= Com_1)$ is the distinguished cell (to designate some specific role).
- *Out* is the *output cell* (for obtaining the outputs).

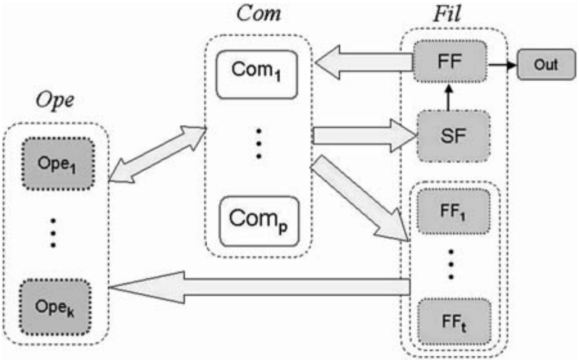


Fig. 1 Modular structure of membrane network in Π

Remark

(1) For $i = 1, \ldots, \mathtt{p}$, each cell Com_i serves as a communication channel, that is, any string x in the cell Com_i is sent out to all the cells indicated by *Syn*.

(2) For $j = 1, \ldots, \mathtt{k}$, each cell Ope_j consists of a finite number of rules $\{\sigma_{j1}, \ldots, \sigma_{js}\}$, where each σ_{ji} is a string insertion operation of the form: $\lambda \to u$, where $u \in V^*$.

(3) *SF* and *FF* are associated with two languages L_{SF} and L_{FF} over V, respectively. Further, for $\ell = 1, \ldots, \mathtt{t}$, each cell FF_ℓ is also associated with a language L_{FF_ℓ}.

3.2 Interpretation of Π

In order to embody a membrane computing schema Π, we need to give further information for Π specifying the initial configuration, each operation in *Ope*, and materializing the filtering cells *SF*, *FF*, and FF_ℓ $(1 \leq \ell \leq \mathtt{t})$. Let us call such a notion an interpretation I to the schema Π which enables us to have an embodied computing model $I(\Pi)$ that is feasible in a usual sense. In what follows, we use the following notation:

Notation For any x in $Com \cup SubFil \cup \{FF\}$, let Syn-from$(x) = \{y \mid (x, y) \in Syn\}$, and for any y in $Ope \cup SupFil \cup \{SF\}$, let Syn-to$(y) = \{x \mid (x, y) \in Syn\}$.

Formally, an *interpretation* I to Π of degree $(\mathtt{p}, \mathtt{k}, \mathtt{t})$ is a construct

$$I = \big(w_0, \{R_1, \ldots, R_{\mathrm{k}}\}, L_{SF}, L_{FF}, \{L_{FF_\ell} \mid 1 \leq \ell \leq \mathtt{t}\}\big),$$

where

- w_0 is a string in V^* called *the axiom*, where V is the working alphabet of Π.
- R_i specifies a set of insertion operations used in Ope_j (for $j = 1, \ldots, \mathtt{k}$)
- L_{SF} (L_{FF}) materializes a concrete specification about the function of *SF* (*FF*, respectively). In practical operational phases (described below), we assume the following:

1. In the cell *SF*, each string is assumed to form a certain *structure* (e.g., structured molecule based on hybridization in terms of H-bonds via minimal energy principle). *SF* takes as input a string u over V and allows it to filter through if it is in L_{SF} (otherwise, a string u is lost). Then after building up a structured form s_u of u, *SF* removes all parts of structures from s_u and produces as output the concatenation of all remaining strings. The output v is sent out to the cell *FF*.
2. *FF* receives as input a string v over V (from *SF*). A string v filters through if it is in L_{FF} and is sent out to *Out*. Otherwise, it is sent out to all cells in Syn-from(*FF*).
3. Each FF_ℓ receives as input a string u over V. Then a string v filters through if it is in L_{FF_ℓ} and is sent out to all cells in Syn-from(FF_ℓ). Otherwise, it is lost.
4. Filtering applies simultaneously to all strings in the filtering cell.

Note that in the case *SubFil* is empty in a given Π, an interpretation to Π is simply written as $I = (w_0, \{R_1, \dots, R_{\mathrm{k}}\}, L_{SF}, L_{FF})$.

3.3 Transitions and Languages

Given a schema Π of degree $(\mathtt{p}, \mathtt{k}, \mathtt{t})$ and an interpretation I to Π, we now have a membrane system $I(\Pi)$ based on string insertions. In what follows, we define a transition sequence of $I(\Pi)$ and the language associated with $I(\Pi)$.

The $(\mathtt{p}+1)$-tuple of languages over V represented by $(L_1, \dots, L_{\mathtt{p}}, L_{\mathrm{out}})$ constitutes a *configuration* of the system, where each L_i represents the set of all strings in the cell Com_i (for all $i = 1, \dots, \mathtt{p}$), and L_{out} is the set of strings presented in the output cell (of the system at some time instance).

Let $C_1 = (L_1, \dots, L_{\mathtt{p}}, L_{\mathrm{out}})$ and $C_2 = (L'_1, \dots, L'_{\mathtt{p}}, L'_{\mathrm{out}})$ be two configurations of the system.

We define one transition from C_1 to C_2 in the following steps:

(0) *Pre-checking Step*: For each $\ell = 1, \dots, \mathtt{t}$, consider $L[\ell] = \bigcup_{i=1}^{q} L_{\ell_i}$, where Syn-to($FF_\ell$) $= \{Com_{\ell_1}, \dots, Com_{\ell_q}\}$. Then each cell FF_ℓ filters out all strings of $L[\ell]$ that are not in L_{FF_ℓ}, and all strings that have passed through are sent to all the cells in Syn-from(FF_ℓ). (In the case when $\mathtt{t} = 0$, i.e., $SubFil = \emptyset$, this step is skipped.)

(1) *Evolution Step*: For each $j = 1, \dots, \mathtt{k}$, let $\sigma_{j1}, \dots, \sigma_{js}$ be all the operations given in Ope_j.

Suppose that we apply operations $\sigma_{ji} : \lambda \to u_{ji}$ $(1 \le i \le s)$ to a string v which means that each σ_{ji} is applied to v *simultaneously*. Further, when we apply σ_{ji} to v, the location in v to insert u_{ji} is non-deterministically chosen and the result $\sigma_{ji}(v)$ is considered as the set of *all possible strings* obtained from v by σ_{ji}. (Note that if two or more rules share the same location to insert, then all possible permutations of those rules are considered to apply to the location.) The result of such an application of all operations in Ope_j to v is denoted by $Ope_j(v)$.

Let $L(j) = \bigcup_{m=1}^{d} L_{j_m}$, where Syn-to($Ope_j$) $\cap$ $Com = \{Com_{j_1}, \dots, Com_{j_d}\}$. Then the total result performed by Ope_j to $L(j)$ is defined as

$$Ope_j\big(L(j)\big) = \bigcup_{v \in L(j)} Ope_j(v).$$

This result is then sent out to all cells in Syn-from(Ope_j) simultaneously.

(2) *Filtering Step*: For each $i = 1, \dots, \mathtt{p}$, let $\tilde{L}_i = \bigcup_{n=1}^{r} Ope_{j_n}(L(j_n))$, where Syn-from($Com_i$) $= \{Ope_{j_1}, \dots, Ope_{j_r}\}$. Further, let $L_e = \bigcup_{m=1}^{g} \tilde{L}_{i_m}$, where Syn-to($SF$) $= \{Com_{i_1}, \dots, Com_{i_g}\}$.

SF takes as input the set L_e and produces as output a set of strings L_f. (Recall that in the cell *SF*, each string u is assumed to form a certain structure, and the output of *SF* is the reduced string by removing structural parts from u in L_{SF}.) Then *SF* sends out L_f to *FF*. (Any element of L_e that was filtered off by *SF* is assumed to be lost.)

Finally, the cell *FF* filters out strings of L_f depending upon whether they are in L_{FF} or not. All strings in L_f that passed through *FF* are sent out to *Out*, while others are simultaneously sent to all Com_i in Syn-from(FF) or they are all lost if Syn-from(FF) $= \emptyset$.

Let L_{ff} be the set of all strings that were filtered off by *FF*. Then we define $C_2 = (L'_1, \dots, L'_{\mathtt{p}}, L'_{\text{out}})$ by setting for each $i = 1, \dots, \mathtt{p}$

$$L'_i = \begin{cases} L_{ff} & \text{if } Com_i \in \text{Syn-from}(FF), \\ \tilde{L}_i & \text{otherwise.} \end{cases}$$

Further, let $L'_{\text{out}} = L_{\text{out}} \cup L_f(Out)$, where $L_f(Out) = L_f - L_{ff}$ (the set of all strings that have passed through *FF* and been sent to *Out*).

Remark

(1) Each cell in *Com* not only provides a buffer for storing intermediate results in the computation process but also *transmit* them to the cells specified by *Syn*.
(2) Each cell in *Ope* takes as input a set of strings L and applies insertion operations to L, and sends out the result to the cells specified by *Syn*.
(3) Each filtering cell in *Fil* also takes as input a set of strings L and performs filtering of L in a way previously described, and sends out the result to the cells specified by *Syn*.
(4) The system has a global clock and counts time in such a way that every cell (including communication cells) takes one unit time to perform its task (described in (1), (2), and (3) for *Com*, *Ope* and *Fil*, respectively), irrespective of the existence of strings in it.

Let $I = (w_0, \{R_1, \dots, R_{\mathtt{k}}\}, L_{SF}, L_{FF}, \{L_{FF_\ell} \mid 1 \le \ell \le \mathtt{t}\})$ be an interpretation to Π. We write $C_1 \Longrightarrow C_2$ if there is a transition from C_1 to C_2 of $I(\Pi)$.

A configuration $C_0 = (\{w\}, \overbrace{\emptyset, \ldots, \emptyset}^{\mathtt{p}})$ with w in V^* is called the *initial configuration*. (We assume that filtering cells are all *empty* in the initial configuration.)

For any $n \geq 0$, let $C_0 \Longrightarrow^n C_n = (L_{1,n}, \ldots, L_{\mathtt{p},n}, L_{\text{out}}^{(n)})$ be a sequence of n-transitions which we call a *computation with n transitions* from C_0. Then a language $L_{\text{out}}^{(n)}$ is called the *nth output* from C_0.

Now, we consider two types of computing models induced from $I(\Pi)$; one is the generating model and the other the accepting model.

[*Generating Model*] In the case of a generating model, we concern all computations whose results are present in the output cell.

We define the language generated by $I(\Pi)$ as follows:

$$L_g\big(I(\Pi)\big) = \bigcup_{n \geq 0} L_{\text{out}}^{(n)},$$

where $L_{\text{out}}^{(n)}$ is the nth output from $(w_0, \emptyset, \ldots, \emptyset)$ and w_0 is the axiom of I.

[*Accepting Model*] In the case of an accepting model, we make a slight modification on an interpretation I and consider the following $I = (\lambda, \{R_1, \ldots, R_{\mathtt{k}}\}, L_{SF}, L_{FF}, \{L_{FF_\ell} \mid 1 \leq \ell \leq \mathtt{t}\})$ with L_{FF} which functions in such a way that if a string v filters through L_{FF}, then (not v but) a special symbol "*Yes*" is sent to *Out*.

Let w be an input string to be recognized. Then w is accepted iff the result *Yes* is present in the output cell after a certain number of transitions from $C_0 = (\{w\}, \emptyset, \ldots, \emptyset)$. Thus, we define the language accepted by $I(\Pi)$ as follows:

$$L_a\big(I(\Pi)\big) = \big\{w \in T^* \mid \mathit{Yes} \in L_{\text{out}}^{(n)}\text{: the } n\text{th output from } (w, \emptyset, \ldots, \emptyset) \\ \text{for some } n \geq 0\big\}.$$

Finally, for a class of schemes $\mathcal{S} = \{\Pi \text{ of degree } (\mathtt{p}, \mathtt{k}, \mathtt{t}) \mid \mathtt{p}, \mathtt{k}, \mathtt{t} \geq 0\}$ and for a class of interpretations $\mathcal{I}$, we denote by $\mathcal{LMS}_x(\mathcal{S}, \mathcal{I})$ the family of all languages $L_x(I(\Pi))$ specified by those systems as above, where Π is in $\mathcal{S}$ and I is in $\mathcal{I}$. That is,

$$\mathcal{LMS}_x(\mathcal{S}, \mathcal{I}) = \big\{L_x(I(\Pi)) \mid I \in \mathcal{I} \text{ is an interpretation to } \Pi \in \mathcal{S}\big\}$$

where x is in $\{a, g\}$.

4 Characterizations by Membrane Schema Π_0

The structure of the membrane computing schema Π introduced in the previous section seems to be general enough to induce a computing device of the universal computability by finding an appropriate interpretation. In what follows, we will show that such a universal computability can be realized by much simpler schemes together with appropriate interpretations of moderately simple filtering cells.

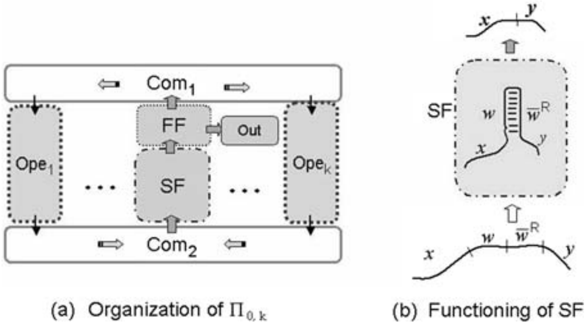


Fig. 2 Membrane computing schema: $\Pi_{0,k}$

First, we consider the following simple membrane computing schema of degree $(2, \mathtt{k}, 0)$:

$$\Pi_{0,\mathtt{k}} = (V, T, Com, Ope, Fil, Syn, i_s, Out),$$

where:

(1) V, T, Ope, i_s and Out are the same as in Π
(2) $Com = \{Com_1, Com_2\}$
(3) $Fil = \{SF, FF\}$ (i.e., *SubFil* is empty)
(4) $Syn = \{Com_1, Ope_i),\ (Ope_i, Com_2) \mid 1 \le i \le \mathtt{k}\} \cup \{(FF, Com_1), (Com_2, SF), (SF, FF), (FF, Out)\}$ (See (a) of Fig. 2).

Let $\Pi_0 = \bigcup_{\mathtt{k}\ge 0} \Pi_{0,\mathtt{k}}$.

We are now in a position to present our first result.

Theorem 4.1 *There exists a class of interpretations $\mathcal{I}_G$ such that $RE = \mathcal{LMS}_g(\Pi_0, \mathcal{I}_G)$.*

Proof We prove only the inclusion $\subseteq$. (The opposite inclusion is due to a consequence of the Turing–Church thesis.)

Let L be any language in RE that is generated by a Chomsky type-0 grammar $G = (N, T, S, P)$. Then we consider the following interpretation $I_G = (S, R_G, L_{SF}, L_{FF})$ to $\Pi_{0,\mathtt{k}}$, where

(i) For each $r : u \to v$ in P, construct $R_r = \{\lambda \to vr,\ \lambda \to \overline{u}^R\overline{r}\}$, and let $R_G = \{R_r \mid r \in P\}$. (Note that the cardinality of R_G gives $\mathtt{k}$ of $\Pi_{0,\mathtt{k}}$.)
(ii) • L_{SF} is given as the following language: $L_{\text{mir}} = \{xw\overline{w}^R y \mid x, y, w \in V^*\}$. That is, a string filters through SF iff it is an element in L_{mir}, where $V = N \cup T \cup \{r \mid r \in P\}$. (Recall that, by definition of SF, any string in SF is assumed to form a structure, and we assume the hybridization by involution relation ρ over V. Specifically, SF performs two functions: it only accepts all structures of molecules containing a single hairpin formed by $w\overline{w}$, and then it removes the portion of a hairpin from the structure and sends out the rest part of the string to FF (see (b) of Fig. 2). The structures rejected by SF are *all lost*.)

- L_{FF} is simply given as T^*, so that only strings in T^* can pass through *FF* and are sent to the output cell *Out*. Other strings are all sent to Com_1.

For any $n \geq 1$, let $C_0 = (\{S\}, \emptyset, \emptyset) \Longrightarrow^n C_n = (L_{1,n}, L_{2,n}, L_{\text{out}}^{(n)})$ be a computation with n transitions in $I_G(\Pi_{0,\text{k}})$. Suppose that $S \Longrightarrow^{n-1} \alpha \Rightarrow_r \beta$ in G, where $\alpha = xuy$, $\beta = xvy$ and $r : u \to v$ is used. Then we can show that

(a) α is in $L_{1,(n-1)}$,
(b) $\beta' = xvru\overline{u}^R\overline{r}y$ is in *SF*, and
(c) after filtering by *SF*, the reduced string of β', i.e., a string $xvy(= \beta)$ is sent to *FF*.

In *FF*, if β is not in T^*, then it is sent to Com_1 and, therefore, in $L_{1,n}$, where $C_n = (L_{1,n}, L_{2,n}, L_{\text{out}}^{(n)})$. Otherwise, β is sent to $L_{\text{out}}^{(n)}$. That is, if β is in $L(G)$, then we have β is in $L_g(I_G(\Pi_{0,\text{k}}))$.

Conversely, suppose that for any $n \geq 1$, α is in $L_{1,n}$. Then there exists $\alpha' = \alpha_1 w\overline{w}^R\alpha_2$ in Com_2 such that $\alpha = \alpha_1\alpha_2$. From the way of constructing R_G, there exists a unique rule $r : u \to v$ in P such that $R_r = \{\lambda \to vr, \lambda \to \overline{u}^R\overline{r}\}$ and $w = ru$. (No other $R_{r'}$ $(r' \neq r)$ can make a substring $w\overline{w}^R$ by insertion operations because of the uniqueness of r.)

Therefore, we can write $\alpha' = \alpha_1 ru\overline{u}^R\overline{r}\alpha_2$ for some α_1, α_2. Then there must exist α_1' such that $\alpha_1 = \alpha_1' v$ because of the rule $\lambda \to vr$. Hence, we have $\alpha' = \alpha_1' vru\overline{u}^R\overline{r}\alpha_2$ from which we can derive that $\alpha_1' u\alpha_2$ is in $L_{1,(n-1)}$. Thus, there exists a derivation: $\alpha_1' u\alpha_2 \Longrightarrow_r \alpha_1' v\alpha_2 = \alpha$ in G. By iteratively applying the above argument, we eventually conclude that there exists a derivation: $S \Longrightarrow^n \alpha$ in G. (For more details, consult discussion in Sect. 3 of [13].)

Taking $L_{1,0} = \{S\}$ into consideration, it holds that for any $n \geq 0$, $L_{1,n} = \{\alpha \mid S \Longrightarrow^n \alpha \text{ in } G\}$. If α is in $L_g(I_G(\Pi_{0,\text{k}}))$, then it is also in $L(G)$. Thus, we have $L(G) = L_g(I_G(\Pi_{0,\text{k}}))$. Clearly, considering for $\mathcal{I}_G$ the class of interpretations I_G for all type-0 grammars G, we complete the proof. □

Note The language L_{mir} used for L_{SF} can be replaced with simpler (regular) language $L_G = \bigcup_{r \in P} V^*\{ru\overline{u}^R\overline{r}\}V^*$, where $r : u \to v \in P$ and P is the set of productions of G. However, we choose L_{mir} here because of its independence of G.

Theorem 4.2 *There exists a class of interpretations $\mathcal{I}_M$ such that* $RE = \mathcal{LMS}_a(\Pi_0, \mathcal{I}_M)$.

Proof We use the same strategy as in Theorem 4.1, but start with a (nondeterministic) Turing machine M. That is, let $L(M)$ be any language in *RE* accepted by $M = (Q, T, U, \delta, p_0, F)$, where $\delta \subseteq Q \times U \times Q \times U \times \{L, R\}$. For $(p, a, q, c, i) \in \delta$, we write $(p, a) \to (q, c, i) \in \delta$. Without loss of generality, we may assume that M immediately stops as soon as it enters into a final state of F. An instantaneous description (ID) of M is represented by a string $\#xpay\#'$ in $\{\#\}U^*QU^*\{\#'\}$, where xay is the tape content $(x, y \in U^*, a \in U)$, M is in the state $p(\in Q)$ and the tape head is on a, and $\#(\#')$ is the left-boundary (right-boundary).

Given an input $w(\in T^*)$, M starts computing w from the state p_0, represented by an ID: $\#p_0w\#'$. (We may assume that the tape content is one-way extendible to the right.) In general, suppose that a transition rule $(p,a) \rightarrow (q,c,i) \in \delta$ is applied to an ID: $\#xbpay\#'$. Then we have a transition between IDs of M:

- $\#xbpay\#' \Longrightarrow \#xbcqy\#'$ (if $i = R$),
- $\#xbpay\#' \Longrightarrow \#xqbcy\#'$ (if $i = L$),
- $\#xbp\#' \Longrightarrow \#xbcq\#'$ (if $i = R$, and $y = \lambda$),
- $\#xbp\#' \Longrightarrow \#xqbc\#'$ (if $i = L$, and $y = \lambda$).

In each case, one can consider a rewriting rule, for example, a rule $pa \rightarrow cq$ for the first case, or a rule $p\#' \rightarrow cq\#'$ for the third case. Let P_M be the set of all rewriting rules obtained from δ in the manner mentioned above. Further, we define $L(M)$ as $\{\#p_0w\#' \mid \#p_0w\#' \Longrightarrow^* \#xqy\#' \text{ for some } q \in F, x, y \in U^*\}$.

We now consider the following interpretation $I_M = (R_M, L_{SF}, L_{FF})$:

(i) For each $r : u \rightarrow v$ in P_M, construct $R_r = \{\lambda \rightarrow vr,\ \lambda \rightarrow \overline{u}^R\overline{r}\}$, and let $R_M = \{R_r \mid r \in P_M\}$, where the cardinality of R_M gives $\mathtt{k}$ in the degree of $\Pi_{0,\mathtt{k}}$.
(ii) L_{SF} is given in the same way as in I_G in the proof for Theorem 4.1.
(iii) L_{FF} is given as V^*FV^*, where $V = U \cup \{\#, \#'\} \cup \{r \mid r \in P_M\}$.

Let w be any string in T^* and $n \geq 0$. Then from the way of constructing I_M together with discussion above, it is easily seen that $\#p_0w\#' \Longrightarrow^n \#xqy\#'$ for some $q \in F$, $x, y \in U^*$ iff there exists $Yes \in L_{\text{out}}^{(n)}$ such that $C_0 = (\{\#p_0w\#'\}, \emptyset, \emptyset) \Longrightarrow^n C_n = (L_{1,n}, L_{2,n}, L_{\text{out}}^{(n)})$. Thus, we have $L(M) = L_a(I_M(\Pi_{0,\mathtt{k}}))$. Let $\mathcal{I}_M$ be the class of interpretations I_M for all Turing machines M, which completes the proof. □

5 Further Results on Some Variants of Membrane Schema

We now introduce two classes of membrane computing schemes $\Pi_{1.5}$ and Π_1 which are variants of Π_0. With an appropriate class of interpretations $\mathcal{I}$, both are able to induce a family of computing devices that can again characterize *RE*.

A membrane computing schema $\Pi_{1,\mathtt{k},\mathtt{t}}$ is given as follows:

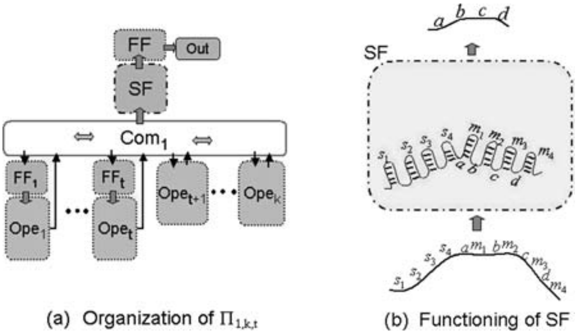
$$\Pi_{1,\mathtt{k},\mathtt{t}} = (V, T, Com, Ope, Fil, Syn, i_s, Out),$$

where:

(1) V, T, Ope, i_s and Out are the same as in $\Pi_{0,\mathtt{k}}$.
(2) $Com = \{Com_1\}$.
(3) $Fil = \{SF, FF\} \cup SubFil$, where $SubFil = \{FF_i \mid 1 \leq i \leq \mathtt{t}\}$ or $= \emptyset$ ($\mathtt{t} = 0$).
(4) $Syn = \{(Com_1, FF_i), (FF_i, Ope_i) \mid 1 \leq i \leq \mathtt{t}\} \cup \{(Com_1, Ope_j) \mid \mathtt{t} + 1 \leq j \leq \mathtt{k}\} \cup (Ope_i, Com_1) \mid 1 \leq i \leq \mathtt{k}\} \cup \{(Com_1, SF), (SF, FF), (FF, Out)\}$. (See (a) of Fig. 3.)

(Note: In (3) and (4) above, $\mathtt{t}$ can take any integer in $\{0, 1, \ldots, \mathtt{k}\}$, and when $\mathtt{t} = 0$, it means the corresponding set is empty. $\Pi_{1,\mathtt{k},\mathtt{t}}$ is a membrane computing schema of degree $(1, \mathtt{k}, \mathtt{t})$ with $\mathtt{t} \leq \mathtt{k}$.)

Fig. 3 Membrane computing schema: $\Pi_{1,\mathtt{k},\mathtt{t}}$


FF
Out
SF
Com$_1$
FF$_1$
Ope$_1$
FF$_t$
Ope$_t$
Ope$_{t+1}$
Ope$_k$
a b c d
SF
s_1 s_2 s_3 s_4 m_1 m_2 m_3 m_4
a b c d
s_1 s_2 s_3 s_4 a m_1 b m_2 c m_3 d m_4

(a) Organization of $\Pi_{1,k,t}$

(b) Functioning of SF

Notation We now consider the following two classes of schemes:

$$\Pi_{1.5} = \bigcup_{\mathtt{k}>\mathtt{t}\geq 0} \Pi_{1,\mathtt{k},\mathtt{t}},$$

$$\Pi_1 = \bigcup_{\mathtt{k}\geq 0} \Pi_{1,\mathtt{k},\mathtt{k}}.$$

Theorem 5.1 *There exists a class of interpretations $\mathcal{I}_{G_m}$ such that $RE = \mathcal{LMS}_g$ $(\Pi_{1.5}, \mathcal{I}_{G_m})$.*

Proof sketch We use an argument similar to the one in Theorem 4.1 and start with a matrix grammar. That is, let L be any language in RE generated by a matrix grammar $G_m = (N, T, S, M, F)$ with appearance checking, where $N = N_1 \cup N_2 \cup \{S, \#\}$, and we may assume that G_m is in the binary normal form (see Sect. 2).

We consider the following interpretation $I_{G_m} = (XA, R_{G_m}, L_{SF}, L_{FF}, \{L_{FF_i} \mid 1 \leq i \leq \mathtt{t}\})$, where

(i) XA is the string in $(S \to XA)$ of G_m.

(ii)-1: $\mathtt{k}$ is the cardinality of M and $\mathtt{t}$ is the number of appearance checking matrix rules in M. For each appearance checking rule $m_i : (X \to Y, A \to \#)$ $(1 \leq i \leq \mathtt{t})$, construct $R_{m_i} = \{\lambda \to Ys_{m_i},\ \lambda \to \overline{X}\overline{s}_{m_i}\}$.

(ii)-2: For other rules $m_j : (X \to Y, A \to x)$ in M (where $Y \in N_1 \cup \{\lambda\}, x \in T \cup N_2 \cup N_2^2 \cup \{\lambda\}$; $\mathtt{t} + 1 \leq j \leq \mathtt{k}$), construct $R_{m_j} = \{\lambda \to Ys_{m_j},\ \lambda \to \overline{X}\overline{s}_{m_j}, \lambda \to xr_{m_j}, \lambda \to \overline{A}\overline{r}_{m_j}\}$. Then let $R_{G_m} = \{R_{m_1}, \ldots, R_{m_\mathtt{k}}\}$.

(iii) • L_{SF} is given as the following regular language: $L_{\mathrm{mat}} = L_s L_r$, where

$$L_s = \{s_{m_i} X\overline{X}\overline{s}_{m_i} \mid m_i : (X \to Y, A \to y) \in M,\ 1 \leq i \leq \mathtt{k}\}^*, \quad \text{and}$$

$$L_r = \big(T \cup N_2 \cup \{r_{m_j} A\overline{A}\overline{r}_{m_j} \mid m_j : (X \to Y, A \to x) \in M,$$
$$\mathtt{t} + 1 \leq j \leq \mathtt{k}\}\big)^*.$$

- L_{FF_i} is given as follows: For each appearance checking rule $m_i : (X \to Y, A \to \#)$ $(1 \le i \le \mathtt{t})$, consider

  $$L_{m_i} = (V \cup \overline{V})^* - (V \cup \overline{V})^* \{A\} (V \cup \overline{V})^*,$$

  where $V = T \cup N \cup \{s_m, r_m \mid m \in M\}$.

  Then L_{FF_i} is given as L_{m_i}. (Thus, FF_i performs in such a way that it allows only strings in L_{m_i} to pass through and sends them to the cell Ope_i. Other strings are all lost.)
- Finally, L_{FF} is given as T^*, so that only strings in T^* can pass through FF and are sent to *Out*.

Let $C_0 = (\{XA\}, \emptyset)$ and consider a transition sequence: $C_0 \Longrightarrow^n C_n = (L_{1,n}, L_{\text{out}}^{(n)})$. Then for any $\alpha \in (N \cup T)^*$ and $n \ge 0$, it holds that $\alpha \in L_{1,n}$ iff $XA \Longrightarrow^n \alpha$ in G_m. Thus, we have $L(G_m) = L_g(I_{G_m}(\Pi_{1,\mathtt{k},\mathtt{t}}))$. Let $\mathcal{I}_{G_m}$ be the class of interpretations I_{G_m} for all matrix grammars G_m, which completes the proof. □

Theorem 5.2 *There exists a class of interpretations $\mathcal{I}_{G_r}$ such that $RE = \mathcal{LMS}_g(\Pi_1, \mathcal{I}_{G_r})$.*

Proof sketch We use the same argument as the one in Theorem 5.1, but start with a random context grammar $G_r = (N, T, S, P)$ generating arbitrary recursively enumerable language L (see Sect. 2). Then consider the following interpretation $I_{G_r} = (S, R_{G_r}, L_{SF}, L_{FF}, \{L_{FF_i} \mid 1 \le i \le \mathtt{k}\})$, where

(i) $\mathtt{k}$ is the cardinality of P. For each rule $r_i : (A \to x, Q, R)$ $(1 \le i \le \mathtt{k})$, construct $R_{r_i} = \{\lambda \to x r_i,\ \lambda \to \overline{A}\overline{r_i}\}$. Then let $R_{G_{r_i}} = \{R_{r_i} \mid r_i \in P\}$.

(ii)
- L_{SF} is defined by the regular language:

  $$L_m = \left(T \cup \{r A \overline{A} \overline{r} \mid r : (A \to x, Q, R) \in P\}\right)^*.$$

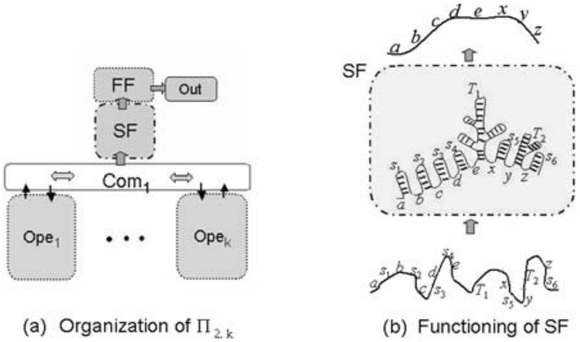
- L_{FF_i} is given as follows: For each rule $r_i : (A \to x, Q, R)$, let

  $$\begin{aligned} L_{r_i} = {} & (V \cup \overline{V})^* \{A\} (V \cup \overline{V})^* \\ & \cap \bigcap_{X \in Q} (V \cup \overline{V})^* \{X\} (V \cup \overline{V})^* \\ & \cap \bigcap_{X \in R} \left((V \cup \overline{V})^* - (V \cup \overline{V})^* \{X\} (V \cup \overline{V})^*\right), \end{aligned}$$

  where $V = N \cup T \cup \{r \mid r \in P\}$. Then L_{FF_i} is defined by L_{r_i}. (That is, each FF_i performs in such a way that it allows only strings in L_{r_i} to pass through and sends them to the cell Ope_i. Other strings are all lost.)
- Finally, L_{FF} is given as T^*, so that only strings in T^* can pass through FF and are sent out to *Out*.

Let $C_0 = (\{S\}, \emptyset)$ (note that S is the starting symbol of G_r) and consider a transition sequence: $C_0 \Longrightarrow^n C_n = (L_{1,n}, L_{\text{out}}^{(n)})$. Then for any $\alpha \in (N \cup T)^*$ and $n \ge 0$,

Fig. 4 Membrane computing schema: $\Pi_{2,\mathrm{k}}$



(a) Organization of Π2,k

(b) Functioning of SF

it holds that $\alpha \in L_{1,n}$ iff $S \Longrightarrow^n \alpha$ in G_r. Thus, we have $L(G_r) = L_g(I_{G_r}(\Pi_{1,\mathrm{k},\mathrm{k}}))$. Letting $\mathcal{I}_{G_r}$ be the class of interpretations I_{G_r} for all random context grammars G_r, we complete the proof. □

We present yet another class of membrane computing schemes Π_2 which is simpler than Π_0 but still able to provide the universal computability of the models induced from the schema with appropriate interpretations, but at the sacrifice of increase of the structural complexity in the filtering function *SF*.

A membrane computing schema $\Pi_{2,\mathrm{k}}$ is given as follows:

$$\Pi_{2,\mathrm{k}} = (V, T, Com, Ope, Fil, Syn, i_s, Out),$$

where

(1) V, T, Ope, Fil, i_s and Out are the same as in $\Pi_{0,\mathrm{k}}$.
(2) $Com = \{Com_1\}$.
(3) $Syn = \{(Com_1, Ope_i),\ (Ope_i, Com_1) \mid 1 \leq i \leq \mathrm{k}\} \cup \{(Com_1, SF), (SF, FF), (FF, Out)\}$ (see (a) of Fig. 4).

Let $\Pi_2 = \bigcup_{\mathrm{k}\geq 0} \Pi_{2,\mathrm{k}}$.

From this simpler class of schemes Π_2, we can induce a family of computing devices $\mathcal{I}(\Pi_2)$ (with an appropriate class of interpretations $\mathcal{I}$) that can characterize *RE*.

Theorem 5.3 *There exists a class of interpretations $\mathcal{I}'_G$ such that $RE = \mathcal{LMS}_g(\Pi_2, \mathcal{I}'_G)$.*

Proof sketch Let L be any language in *RE* that is generated by a Chomsky type-0 grammar $G = (V, T, S, P)$. Then consider the following interpretation $I'_G = (S, \{R_G\}, L_{SF}, L_{FF})$, where

(i) For each $r : u \to v$ in P, construct $R_r = \{\lambda \to vr,\ \lambda \to \overline{u}^R\overline{r}\}$, and let $R_G = \bigcup_{r\in P} R_r$. The cardinality of R_G gives k in the degree of $\Pi_{2,\mathrm{k}}$.
(ii) L_{SF} adopts the Dyck language D over the alphabet $N \cup T \cup \{r \mid r \in P\}$.
(iii) L_{FF} is given as T^*, so that only strings in T^* can pass through FF and are sent out to *Out*.

Consider a transition sequence: $C_0 = (\{S\}, \emptyset) \Longrightarrow^n C_n = (L_{1,n}, L_{\text{out}}^{(n)})$. Then for any $\alpha \in (N \cup T)^*$ and $n \geq 0$, it holds that $\alpha \in L_{1,n}$ iff $S \Longrightarrow^n \alpha$ in G. Thus, it holds that $L(G) = L_g(I'_G(\Pi_{2,\text{k}}))$. In order to complete the proof, we have only to consider for $\mathcal{I}'_G$ the class of interpretations I'_G for all type-0 grammars G. (The proof is based on the following result that each recursively enumerable language L can be represented in the form $L = h(L' \cap D))$, where L' is an insertion language, h is a projection and D is a Dyck language. (Theorem 3.1 in [13]). In order to understand the idea of the proof, it would be helpful to note that R_G in Com_1 generates the insertion language L', while a pair of *SF* and *FF* plays the same role as a pair of D and h, respectively.) □

6 Concluding Remarks

In this paper, we have introduced the notion of a membrane computing schema and showed that several known computing models with the universal computability can be reformulated in a uniform manner in terms of the framework of the schema together with its interpretation. A similar idea in the context of grammar schema has been proposed and discussed in [2, 6] to prove the computational completeness of new type of P systems based on the framework of the random context grammars for both string and multi-set languages. (Note that the definition of random context in those papers is not on a string to be rewritten but on the applicability of rules to re-write, different from the standard notion.) As for the communication by sending objects and the use of filtering function, there are several papers that have been devoted to studying the computational powers of communicating distributed H systems (e.g., [4, 12]), and of the hybrid networks of evolutionary processors (e.g., [8, 9]). Among others, the notion of observers in G/O systems proposed in [1] may be of special interests in that it seems to have a close relation to the filtering mechanism (for structured or string objects) of the membrane computing schema introduced in this paper.

Table 1 summarizes the results we have obtained. From the table, one can have a unified view of the existing various models of computation based on *string rewrit-*

Table 1

Computing model	Schema	*SF*	*FF*	*SubFil*
Chomsky type-0 grammar	Π_0	$L_G(\in RG)$ or $L_{\text{mir}}(\in LIN)$	T^*	(N.A.)
Turing machine	Π_0	$L_G(\in RG)$ or $L_{\text{mir}}(\in LIN)$	V^*FV^*	(N.A.)
Random context grammar	Π_1	$L_m(\in RG)$	T^*	$L_{r_i}(\in RG)$
Matrix$_{ac}$ grammar	$\Pi_{1.5}$	$L_{\text{mat}} = L_s L_r(\in RG)$	T^*	$L_{m_i}(\in RG)$
Chomsky type-0 grammar	Π_2	$D(\in CF)$	T^*	(N.A.)

ing. For example, it is seen that there exists a trade-off between the complexity of network structure in the schema and the complexity of the filtering *SF*.

More specifically, for new terminologies, L is a *star language* iff $L = F^*$ for some finite set F. Further, L is an *occurrence checking language* iff $L = V^*FV^*$ for some finite set F. Then it should be noted that in Table 1:

(i) L_G is a finite union of occurrence checking languages,
(ii) L_s, L_r and L_m are star languages,
(iii) L_{r_i} is a finite intersection of occurrence checking languages and their complements,
(iv) L_{m_i} is the complement of an occurrence checking language.

Since Π_0 (or Π_1) is more complex than Π_2, one may see a trade-off between the complexity of the schema and that of *SF*, telling that L_G for single (or L_{mat} for multiple) hairpin checking is simpler than a Dyck language D for nested hairpin checking. This kind of trade-off can also be seen in complexity between a series of schemes $(\Pi_1, \Pi_{1.5}, \Pi_2)$ and the corresponding SFs$(L_m, L_{\mathrm{mat}}, D)$.

In this paper, we have just made the first step in the new direction toward understanding and characterizing the nature of the Turing computability from the novel viewpoint of *modularity* in the membrane computing schema. There seems to remain many left for the future works:

- it would be the most interesting to study the relation between the complexity of the language classes and that of SF within a given schema. For instance, we can show that within the schema Π_2, *CF* can be characterized by star (regular) languages for *SF*.
- Instead of insertion operations we adopted in this paper, what kind of operations can be considered for the unique operation in the cells *Ope*? What kind of different landscape of the computing mechanism can be seen from the new schema?

Acknowledgements The first author wishes acknowledge the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, co-financed by FEDER funds and the support of the project of excellence TIC-581 of the Junta de Andalucía. The second author gratefully acknowledges the support of the Grant of Faculty Development Award, Waseda University and Grant-in-Aid for Scientific Research on Priority Area No. 14085202, Ministry of Education, Culture, Sports, Science, and Technology, Japan.

References

1. Cavaliere M, Leupold P (2004) Evolution and observation—a non-standard way to generate formal languages. Theor Comput Sci 321:233–248
2. Cavaliere M, Freund R, Oswald M, Sburlan D (2007) Multiset random context grammars, checkers, and transducers. Theor Comput Sci 372:136–151
3. Chen H, Freund R, Ionescu M, Păun Gh, Pérez-Jiménez MJ (2006) On string languages generated by spiking neural P systems. In: Gutierrez-Naranjo MA, Păun Gh, Riscos-Nunez A, Romero-Campero FJ (eds) Reports on fourth brainstorming week on membrane computing, vol 1, pp 169–193

4. Csuhaj-Varju E, Kari L, Păun Gh (1996) Test tube distributed systems based on splicing. Comput Artif Intell 15(2–3):211–232
5. Dassow J, Păun Gh (1989) Regulated rewriting in formal language theory. Springer, Berlin
6. Freund R, Oswald M (2004) Modeling grammar systems by tissue P systems working in the sequential mode. In: Proceedings of grammar systems workshop, Budapest
7. Ionescu M, Păun Gh, Yokomori T (2006) Spiking neural P systems. Fund Inform 71(2–3):279–308
8. Margenstern M, Mitrana V, Pérez-Jiménez M (2005) Accepting hybrid networks of evolutionary processors. In: Lecture notes in computer science, vol 3384. Springer, Berlin, pp 235–246
9. Martin-Vide C, Mitrana V, Pérez-Jiménez M, Sancho-Caparrini F (2003) Hybrid networks of evolutionary processors. In: Proceedings of GECCO. Lecture notes in computer science, vol 2723. Springer, Berlin, pp 401–412
10. Martin-Vide C, Păun Gh, Salomaa A (1998) Characterizations of recursively enumerable languages by means of insertion grammars. Theor Comput Sci 205:195–205
11. Martin-Vide C, Păun Gh, Pazos J, Rodriguez-Paton A (2003) Tissue P systems. Theor Comput Sci 296:295–326
12. Păun Gh (1998) Distributed architectures in DNA computing based on splicing: limiting the size of components. In: Calude CS, Casti J, Dinneen MJ (eds) Unconventional models of computation. Springer, Berlin, pp 323–335
13. Păun Gh, Pérez-Jiménez MJ, Yokomori T (2007) Representations and characterizations of languages in Chomsky hierarchy by means of insertion-deletion systems. In: Proceedings of DCFS2007, High Tatras, Slovakia, pp. 129–140. Also, to appear in Int J Found Comput Sci, 2008
14. Păun Gh, Rozenberg G, Salomaa A (1998) DNA computing. Springer, Berlin
15. Păun Gh, Sakakibara Y, Yokomori T (2002) P-systems on graph of restricted forms. Publ Math Debrecen 60:635–660
16. Rozenberg G, Salomaa A (1997) Handbook of formal languages. Springer, Berlin

Part VI Formal Models and Analysis

Finite Splicing: Generative Capacity, New Models and Complexity Aspects

Paola Bonizzoni and Remco Loos

Abstract Splicing systems have been introduced twenty years ago as a basic abstract model of the DNA recombination mechanism. In fact, it was the first of a long series of computational models based on a molecular process. Much research has been done on the generative capacity of these systems, mostly considering enhanced variants of the original definition. However, some important questions about the original finite systems are still unsolved. For example, we do not have any systematic way to go about constructing a splicing system for a given language, and we still lack significant algorithmic results for this model.

In this work, we survey new research directions on finite splicing that could suggest a new approach to the solution of these basic problems and could shed a new light on the splicing formalism. These include an alternative definition of the splicing language, splicing systems as accepting devices, and complexity issues for splicing systems.

1 Introduction

This work aims to present recent developments in a research topic that originated from the fundamental theoretical studies on biological computations based on operations modifying DNA molecules. This topic concerns the *splicing operation* as a basic operation of computational models called *Splicing Systems*, also known as *H-systems*. Splicing is a basic biological mechanism that consists in the cutting and recombination of DNA molecules under the influence of restriction enzymes. It can be formalized as on operation on a pair of strings by means of the well-known notion of *a splicing rule* consisting of a pair of the form $(u_1, u_2)\$(u_3, u_4)$, where u_1u_2, u_3u_4 are specific strings over an alphabet representing the substrings or location where the rule applies, called *splice sites*. The cutting occurs in the location between u_1 and u_2 and u_3 and u_4. Then whenever the rule applies to strings wu_1u_2v and yu_3u_4z, after cutting the recombination generates new strings wu_1u_4z or yu_3u_2v specified by the *paste sites* u_1u_4 and u_3u_2.

Work supported by Research Grants BES-2004-6316 and ES-2006-0146 of the Spanish Ministry of Education and Science.

P. Bonizzoni (✉)
Dipartimento di Informatica Sistemistica e Comunicazione, Università degli Studi di Milano—Bicocca, Viale Sarca 336, 20126 Milano, Italy
e-mail: bonizzoni@disco.unimib.it

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_17, © Springer-Verlag Berlin Heidelberg 2009

This simple formulation of the splicing operation has attracted the interest of several researchers toward the investigation of computational models inspired by molecular operations. After the first relevant results in this field showing the capacity of this basic formalism to generate regular languages, the general research has moved to several variants, with additional computational mechanisms, showing even the universality of such systems (see [27] for an overview). Then the interest moved toward the formalization of models using other molecular operations and models [11].

However, in spite of the huge literature on splicing systems, it turned out that the simpler notion of splicing system invented by T. Head still lacked a deep investigation explaining the real language generative capacity of such systems.

This open question has been the subject of recent work in [3, 6, 7, 14, 15] aiming to characterize languages generated by the splicing operation by means of classical notions of formal language theory. At the same time, this research direction raised the question of investigating the computational power of the splicing operation when acting on circular string, showing interesting connections with classical concepts on circular regular languages [2, 4, 5].

In this paper, we will survey recent improvements in this research direction, while pointing out the open questions that still are unsolved in this field, basically decision problems on the generative capacity of these systems.

Moreover, while the basic splicing operation still needs to be clearly understood from a formal language and algebraic view, other aspects of splicing systems have recently been addressed which show that the basic finite model of splicing still offers room for new research.

On the one hand, a new definition of a splicing language has been proposed, based on the properties of the biological recombination operation [20, 23]. Such a new definition has a great impact on the computational power of finite splicing systems. On the other hand, complexity aspects of splicing systems have been systematically addressed [21, 24, 25]. This allows to explore dimensions of the splicing formalism left aside by most of the research done so far, which had been mainly focused on computational power. Here, we will mostly address descriptional complexity, which also has some relation to many of the basic questions mentioned above.

Finally, new work shows that splicing systems can also be regarded as accepting devices [21, 22]. It turns out that results and methods for generating systems do not carry over straightforwardly to the accepting systems, which makes that these give rise to a whole new set of questions.

The paper is organized as follows. In Sect. 2, we recall the basic notions and terminology on splicing systems and splicing languages, and survey the most recent research developments on the characterization of splicing languages in a formal language framework. We explore their relation with the notion of a Schützenberger constant, and look into decision algorithms for splicing systems [30].

Section 3 is devoted to recent results showing splicing systems from a different perspective. The notion of a language generated by a non-preserving splicing system is introduced in Sect. 3.1. Definitions and results on descriptional complexity for splicing systems are treated in Sect. 3.2. Finally, Sect. 3.3 addresses accepting splicing systems.

We conclude the paper with a section discussing open questions in this formalism and pointing to directions for further research.

2 Formal Languages and Finite Splicing Systems

Since the early work by Head in 1987, linear splicing systems have been conceived as generative devices of formal languages.

The recombinant behavior of DNA molecules under the action of restriction and ligase enzymes is modeled by means of a basic operation on strings. The splicing operation acts on a pair of strings, or strands of DNA, by cutting then at specified substrings, described by splice sites and modeling restriction enzymes that recognize a pattern inside the molecules. Then fragmented strings are pasted according substrings of the splice sites modeling the action of ligase enzymes. In particular, Head was concerned with the structure of the languages of those DNA molecules (strings) which could be produced through the splicing operation, performed by a splicing system consisting of a finite set of initial DNA molecules (initial language) and a finite set of enzymes (set of rules).

Under this assumptions, a splicing system, called H system, consists of a triple $H=(V,A,R)$ where V is a finite alphabet, A is a finite set of words over V, or initial language and R is a finite set of splicing rules. Let us recall that given a rule $r=(u_1,u_2)\$(u_3,u_4)$ that applies to $w'=wu_1u_2v$ and $y'=yu_3u_4z$, then $x=wu_1u_4z$ is a new string generated by splicing; then the derivation of x from w' and y' by rule r is denoted as $(w',y')\vdash_r x$.

Then the *splicing language* generated by the system is defined using the notion of *closure* of a language L under the set R of rules. Formally, given

$cl(L,R)=\{w:(x,x')\vdash_r w, r\in R, x,x'\in L\}$, then the language generated by the system H is $\sigma_R^*(A)$, where

$$\begin{aligned}\sigma_R^0(L)&=L,\\ \sigma_R^{i+1}(L)&=cl\bigl(\sigma_R^i(L),R\bigr)\cup\sigma_R^i(L),\qquad i\geq 0,\\ \sigma_R^*(L)&=\bigcup_{i\geq 0}\sigma_R^i(L).\end{aligned}$$

We say that a splicing language L *is closed under a set R of rules* iff $cl(L,R)\subseteq L$.

Also, an *extended splicing system* can be defined by adding a terminal alphabet $T\subseteq V$ to the definition. Such an extended system $H=(V,T,A,R)$ generates the language $L(H)=\sigma_R^*(A)\cap T^*$.

The language generative capacity of H-systems has been extensively investigated under which level in Chomsky hierarchy A, R belong to, showing that these systems can be as powerful as the Turing machines [18, 27]. At the lowest level of the hierarchy, the regularity of splicing languages generated by regular initial languages and a finite set of rules is proved in [29]. On the other side, H-systems with finite initial set of rules and a finite initial language generate a proper subclass of regular languages [8]. The strict inclusion firstly proved in [12], has been proved several times (see [19, 29]) and [33] as examples).

A constructive proof has been given, for example, in [19], where an algorithm to construct a finite state automaton that recognizes the language generated by a not necessarily finite splicing system (V, L, R), where L is a regular language and R is a finite set, is described.

The investigation of this class has opened new questions that show the strict connection between formal language frameworks and the splicing operation on strings. Indeed, the complete characterization of regular splicing languages is an intriguing open problem. Since the pioneering work of Head, it has been evident that well-known concepts in formal language theory, such as the one of a *constant* for a regular language L, given by Schützenberger in [30] play an important role in the description of the structure of the above-mentioned class of regular languages.

Indeed, Head showed that under some conditions, the generated language is in the class of strictly locally testable languages [16].

Later on, even the notion of *syntactic congruence* for a regular language L [28] has revealed to be crucial to relate the splicing operation to the class of regular languages [3, 15].

In the following, we will reconstruct the basic steps that have been done to achieve basic results in this research field: the characterization of the original class of Head regular splicing languages [7] and decisions procedures for this class of splicing languages [10, 15].

It must be pointed out that in the literature at least two other notions of splicing rules and splicing operations have been proposed. These are known as Head and Pixton splicing operations, respectively. In [9], it has been shown that splicing systems based on Pixton splicing operation are more powerful than the ones based on the standard (Păun) splicing, and these systems are more powerful than Head splicing systems.

A classification of these different notions of splicing may be given by using the standard (Păun) splicing operation adopted also in this paper, simply by requiring that the set R of rules defines a specific (symmetric, reflexive, or transitive) binary relation over the pair of splice sites, as pointed out partially in [6].

Let us recall that a set R of rules is called *reflexive* whenever given a rule $r \in R$ relating the splice site u_1u_2 to u_3u_4 then the rule r' with splice sites both identical u_1u_2 is still in R. Similarly, R is *symmetric* whenever given $r \in R$, then the rule relating u_3u_4 to u_1u_2 is given in R.

Observe that reflexivity and symmetry can be stated as properties of a splicing language. More precisely, a language is defined to be reflexive or symmetric iff it is closed under a set of reflexive or symmetric rules, respectively, generating the language.

The relationship between symmetric and non-symmetric splicing languages has been investigated in [32]. These languages are called 1-*splicing languages* in [27] and they are properly included in the class of symmetric splicing languages as proved in [32], also called 2-*splicing languages*: indeed the languages of Lemma 1 show the strict inclusion.

Now, reflexivity and symmetry are the most relevant properties that relate the different Păun and Head notions of splicing operation. These two properties were implicitly introduced by Head's definition of splicing languages as necessary conditions for an accurate biological representation of DNA splicing systems. Indeed, Head splicing languages are reflexive and symmetric splicing languages, while according to Păun's definition there exist non-reflexive and non-symmetric splicing languages.

Actually, the reflexive property of rules is equivalent to the property of splice sites of the rules being constants for the regular splicing language. This is the first intermediate significative result relating splicing languages to constants and has been proved for symmetric and reflexive languages in [7] and [15]). Formally, the result provides the following characterization of reflexive (symmetric or non-symmetric) languages: a regular language is a (symmetric) reflexive splicing language L iff it is generated by a splicing system having (symmetric) rules with splicing sites that are constants for the language L.

A basic consequence of this result is that classes of regular splicing languages can be characterized under this well-known notion. Indeed, classes of reflexive splicing languages having splice sites of rules that are constants of a specific form can be related to known classes of regular languages characterized under the notion of constant.

2.1 Constant Languages and Classes of Regular Splicing Languages

The seminal work on splicing operation [16] is a clear example of how the form of constants in a regular splicing language i.e. a characterization of the regular language in terms of constants can be translated into rules of a specific form as detailed below.

Languages generated by a system $S = (V, A, R)$ where each rule $r \in R$ is of the form $(x, 1)\$(x, 1)$ or $(1, x)\$(1, x)$, for $x \in V^+$, called *null context splicing languages* (NCS, in short) are *strictly locally testable languages*.

Strictly locally testable languages (SLT) are characterized in [13] as those languages for which there exists a positive integer k for which every string in V^* of length k is a constant.

Languages generated by systems with *one-sided* rules of the form $(1, v)\$(1, u)$ or $(v, 1)\$(u, 1)$, for $u, v \in V^*$ are *finitely constant generated splicing languages*, or simply FCS languages [17].

These languages are characterized by a crucial notion in finite splicing theory that has been firstly introduced in [17]: that of *constant language*.

Let L be a regular language and m be a word in V^* that is a constant for L. Let us recall that a word $m \in V^+$ is a constant for language L if $V^*mV^* \cap L \neq \emptyset$ and for all words $x_1, x_2, v_1, v_2 \in V^*$, $x_1 m x_2 \in L$, $v_1 m v_2 \in L$ implies that $x_1 m v_2, v_1 m x_2 \in L$. A *constant language in L for m* is the language $L(m, L) \subseteq L$ such that $L(m, L) = V^* m V^* \cap L$.

In the paper, for simplicity, we use the notation $L(m)$ for denoting a constant language $L(m, L)$ in L.

A language L is a FCS language if it is a finite union of a finite set with a finite set of constant languages in L for a set $\mathcal{M}$ of constants of L.

The language b^+ab^+ is an example of FCS language that is not a NCS language as every string consisting only of $b's$ is not a constant. Vice versa, note that a NCS language is not necessarily a constant language, as it holds in the case of language $L = a^* \cup b^*$, as L is the union of two constant languages over two distinct symbols of the alphabet.

The characterization of the whole class of (symmetric) reflexive splicing languages has required the introduction of the notion of *split-language*.

Indeed, given L a reflexive symmetric splicing language, then L is characterized in terms of a finite set $\mathcal{M}$ of constants for language L. More precisely, L is defined in finite terms as a finite union of split languages.

Given L a regular language, a splicing operation is defined for a pair of constant languages $L(m_1)$, $L(m_2)$ in L by a splicing rule r if each of the constants m_1 and m_2 is a distinct splice site of rule r.

Formally, given $r = (u_1, u_2)\$(u_3, u_4)$, such that $u_1u_2 = m_1$, $u_3u_4 = m_2$, and $L(m_1) = L_{i1}u_1u_2L_{i2}$, $L(m_2) = L_{j1}u_3u_4L_{j2}$, then the result of a splicing operation of $L(m_1)$, $L(m_2)$ by r is the language $L_{i1}u_1u_4L_{j2}$ denoted as $\mathit{SPLIT}(L(m_1), L(m_2), r)$ and called *split language*.

Given $\mathcal{M}$ a finite set of constants for language L, we define the set $F(\mathcal{M})$ of 2-factors of words in $\mathcal{M}$:

$$F(\mathcal{M}) = \{(m_{i1}, m_{i2}) : m_{i1}m_{i2} \in \mathcal{M}\}.$$

A binary relation over $F(\mathcal{M})$ induces a set $R_{\mathcal{M}}$ of rules, precisely, $R_{\mathcal{M}} \subseteq \{(s_1\$s_2) : s_1, s_2 \in F(\mathcal{M})\}$: let us call $R_{\mathcal{M}}$ set of *constant based rules over* $\mathcal{M}$.

The characterization theorem for reflexive symmetric splicing languages in [7], is then stated below.

Theorem 1 *Let L be a regular language. The following are equivalent*:

1. *L is a* (*symmetric*) *reflexive splicing language.*
2. *There exists a finite set* $X \subset A^*$, *a finite set of constants* $\mathcal{M}$ *for L*, *a set* $R_{\mathcal{M}}$ *of* (*symmetric*) *constant based rules over* $\mathcal{M}$ *and*

$$L = \bigcup_{m_i \in \mathcal{M}} L(m_i) \bigcup_{r_{ij} \in R_{\mathcal{M}}} \mathit{SPLIT}\big(L(m_i), L(m_j), r_{ij}\big) \cup X. \tag{1}$$

Then the set $\mathcal{M}$ is called *generating set of constants* for language L.

Example 1 The regular language $L = a^+ba^+ba^+ \cup a^+ca^+ba^+$ is a reflexive symmetric splicing language. Indeed, given the set $\mathcal{M} = \{c, bab\}$ of constants for L and the constant languages $L_1 = a^+m_1a^+$ and $L_2 = a^+m_2a^+ba^+$, where $m_1 = bab$, $m_2 = c$, then $L = L_1 \cup L_2 \cup Split(L_1, L_2, r)$, where $r = (b, ab)\$(ac, 1) \in R_{\mathcal{M}}$.

Observe that language L is not a FCS language, as indeed language $a^+ba^+ba^+$ cannot be obtained as a finite union of constant languages and of a finite set.

The following remark holds as a consequence of the above result.

Remark 1 Given L a regular language, a constant language $L(m)$ is a special case of split language, as indeed $L(m) = SPLIT(L(m), L(m), r)$, where $r = ((m, 1)\$(m, 1))$ is a constant based rule. Moreover, observe that given L a splicing language characterized by (1) of Theorem 1, then for every pair m_i, m_j it holds that $cl(L(m_i) \cup L(m_j), r_{ij}) = SPLIT(L(m_i), L(m_j), r_{ij})$.

By using the above remark, we can obtain as a corollary of Theorem 1 the following characterization of reflexive splicing languages which is proved in [15].

Corollary 1 *Let L be a regular language. Then the following are equivalent*:

1. *L is a reflexive* (*symmetric*) *splicing language.*
2. *There exists a set R of reflexive* (*symmetric*) *rules such that L is closed under R and* $L - cl(L, R) = X$, *for* $X \subset A^*$ *a finite set.*

There exist splicing languages that are reflexive and not symmetric as stated in Lemma 1. Indeed, by applying a Theorem stated in [32], it is proved that $L_1 = a^* \cup a^*ba^*$ and $L_2 = a^* \cup da^* \cup a^*c$ are not symmetric languages, while we can show that these languages are reflexive.

Lemma 1 *Languages* $L_1 = a^* \cup a^*ba^*$ *and* $L_2 = a^* \cup da^* \cup a^*c$ *are splicing languages that are reflexive and not symmetric.*

Proof The language L_1 can be expressed as $L(b) \cup cl(L(b), R)$, where $R = \{r\}$, $r = (1, b)\$(b, 1)$, $L(b) = a^*ba^*$ is a constant language. Similarly, the language $L_2 = L(d) \cup L(c) \cup cl(L(c) \cup L(d), r)$, where $L(d) = da^*$ and $L(c) = a^*c$ and $r = (1, c), (d, 1)$. Then by Proposition 1, L_1 is a reflexive splicing language. □

In [15], it is proved that the regular language $a^*b^*a^*b^*a^* \cup a^*b^*a^*$ is a symmetric, non-reflexive splicing languages, while language $b^*a^*b^*a^* \cup a^*b^*a^* \cup a^*$ is a splicing language that is neither symmetric nor reflexive.

The following Lemma 2 shows also the existence of splicing languages that are neither reflexive nor symmetric.

Lemma 2 [15] *The language* $L = a^+d^+b^+a^+d^+b^+a^+ \cup a^+d^+b^+a^+ \cup a^+$ *is a splicing language that is neither symmetric nor reflexive.*

2.2 Decision Procedures

The characterization of reflexive splicing languages stated in Theorem 1 and Corollary 1 leads to decision procedures for reflexive languages.

A crucial step in these decision procedures is to have an algorithm to decide whether a language is closed under a set of rules. This procedure has been detailed in [6].

In [10], a decision procedure for reflexive splicing languages based on the characterization stated in Theorem 1 is provided. It mainly consists of the following steps:

(1) First, a characterization of the classes of the syntactic monoid that allow to compute a set of generating constants for a regular splicing language is provided using a notion of equivalence relation among words. Then it has been proved that this relation leads to a refinement of constant classes of the syntactic monoid. Smallest representatives of these classes directly provide the finite largest set of candidate generating constants for the splicing language.

(2) For each subset $\mathcal{M}$ of candidate generating constants for the language, then the decision algorithm tests whether the union of constant languages is closed under a given subset of constants based rules over $\mathcal{M}$. Clearly, this step requires exponential time in the size of the reduced automaton for the regular language.

Note that bounding the length of generating constants for the language is a crucial step in the decision procedure.

A different algorithm is proposed in [15], based on an alternative characterization of reflexive splicing languages using interesting properties concerning the syntactic monoid of a splicing language. Again, in the decision algorithm, the construction of a bound for the length of rules under which the regular language is closed is a crucial step. Actually, finding a tight upper bound for the length of splice sites of rules is an interesting open question.

3 New Directions

Since the introduction of the splicing, and the early results showing regularity of the basic formalism, research into splicing systems has continued, but has moved away from these basic systems. As mentioned before, this research has been mainly focused on universal models of splicing involving additional control mechanism or special features. Recently, however, new research directions for finite splicing systems have been explored. These shed a new light on the splicing formalism and show that the basic formalism still has room for further study. We here introduce some of these new directions, present their results and point to open problems and opportunities for further research.

3.1 Non-preserving Splicing

The general picture that had emerged from years of splicing research was the following. Basic finite splicing systems are computationally weak, but when adding some kind of additional feature they are computationally complete. These features can be a regular set of rules, control mechanisms taken from regulated rewriting, or distributed architectures (see [27] for an overview of models and results).

In [20] and [23], another approach was taken. Instead of adding features, the basic definition of a splicing system was reexamined. If we recall the definition of a splicing systems from Sect. 2, we see that its language is defined by iteratively adding strings to an initial set. However, this is not necessarily what happens in the biochemical recombination operation. For instance, suppose we use restriction enzymes *TaqI* and *SciNI*, with the following patterns given in Fig. 1.

Since the sticky ends are the same, new molecules can be produced by recombination:

$$\begin{array}{cc} \ldots TCGC \ldots & \ldots GCGA \ldots \\ \ldots AGCG \ldots & \ldots CGCT \ldots \end{array}$$

These new molecules cannot be cut by either of the two enzymes, which makes that all old molecules will be replaced by new molecules. In splicing terms, this means that a language cannot only evolve by adding strings, but also by removing strings and replacing them with new ones. *Non-preserving splicing* reflects this behavior. In these models, at each step, whenever a rule is applied to a string, this string is not present in the next step, unless it is created as the result of a rule application in the same step. Strings which are not involved in any rule remain unchanged in the language.

Formally, we define a new splicing step for a set of splicing rules R over V and for a language $L \subseteq V^*$:

$$\tau_R(L) = \sigma_R(L) \cup \big\{ w \in L \mid \big(\neg\exists x \in L, r \in R, y, z \in V^*\big)\big[(w,x) \vdash_r (y,z)\big]\big\}.$$

Then we can define a *non-preserving splicing system* as a usual splicing system, but where the language produced is $\tau_R^*(A)$, which is defined as follows:

$$\begin{aligned} \tau_R^0(L) &= L, \\ \tau_R^{i+1}(L) &= \tau_R\big(\tau_R^i(L)\big), \quad i \geq 0, \\ \tau_R^*(L) &= \bigcup_{i \geq 0} \tau_R^i(L). \end{aligned}$$

Surprisingly, with this language definition, finite spicing systems are very powerful.

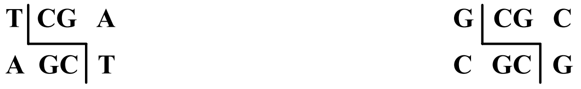


Fig. 1 Restriction sites of enzymes *TaqI* and *SciNI*

Theorem 2 [20, 23] *Finite extended non-preserving splicing systems are computationally complete.*

From this result, we automatically obtain the following decision properties.

Corollary 2 *For finite non-preserving splicing systems, membership, emptiness, finiteness, equivalence, and generally all non-trivial properties are undecidable.*

3.2 Complexity Aspects

Splicing systems have been mainly studied from the point of view of computational power. Other aspects, such as complexity issues, have not been systematically addressed. For example, descriptional complexity has been previously studied, but this work focused on limiting a specific resource or structural parameter. While measuring specific resources can provide insights into the role that these parameters play, it does not necessarily say that much about the overall description necessary to specify a particular system. In [21], a first real attempt is made to address descriptional complexity for splicing systems in terms of the total size of the system. Extended finite splicing systems are compared with non-deterministic finite automata (NFA). Both systems generate (respectively accept) exactly all regular languages [26].

3.2.1 Complexity Measures

The first step in studying descriptional complexity is defining fair and meaningful measures. For finite automata, the primary complexity measure considered in the literature is the number of states. This is normally a good indicator of total size, since the number of transitions of an n-state NFA over Σ is bounded by $n^2 \cdot Card(\Sigma)$ and it gives fair comparisons when comparing with other formalisms having states or a comparable resource (e.g., non-terminals in context-free grammars). However, since for splicing systems there does not seem to be such a resource and since we are interested in the total size of the systems, we also consider the number of transitions of the NFA. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA. Then we denote

$$\mathcal{Q}(M) = Card(Q),$$
$$\mathcal{T}(M) = Card(\delta).$$

For splicing systems, there are several measures we could consider. For instance, one could consider the number of rules of the system. But because of its structure consisting of an initial language and a set of rules, even a system with an empty set of rules can be arbitrarily complex in terms of the size of the equivalent NFA (as complex as the finite initial language). Even if we also consider the size of A, the length of the words in A can make the equivalent NFA arbitrarily complex. Also, as we will see, the length of the rules is relevant. This is why we will measure the size of a splicing system by considering both the number and the length of the rules and the initial language. Given an extended finite H system $\Gamma = (V, T, A, R)$, we define the following measures:

$$\mathcal{R}(\Gamma) = \sum_{r \in R} |r|,$$

$$\mathcal{A}(\Gamma) = \sum_{w \in A} |w|.$$

The total size of Γ, denoted $Size(\Gamma)$, is defined as $Size\mathcal{R}(\Gamma) + Size\mathcal{A}(\Gamma)$.

3.2.2 Descriptional Complexity of Finite Splicing Systems

We first show upper and lower bounds for the increase in descriptional complexity when changing from an finite extended splicing system to an equivalent NFA.

Theorem 3 [21] *For the conversion of a finite extended splicing system Γ to an equivalent NFA M, the following bounds hold*:

1. *Upper bound*: $\mathcal{Q}(M) \leq \mathcal{A}(\Gamma) + \mathcal{R}(\Gamma) + 1$.
2. *Lower bound*: $\mathcal{Q}(M) \geq \mathcal{A}(\Gamma) + \mathcal{R}(\Gamma) - 8$.

Proof (sketch) For (1), this follows from a streamlined version of Pixton's construction [29]. We can improve this construction by realizing that the first state of each path describing each u_1u_4 is only reached by λ-transitions and that all outgoing transitions of the last state of the path are also λ-transitions. This means that removing λ-transitions in the usual way and removing all superfluous states, we can eliminate those states.

For part (2), we define an infinite sequence of languages $L_n, n \geq 1$, such that each L_n is generated by an finite extended splicing system Γ_n, where $\Gamma_n = (V, A, R_n)$, with

- $V = \{a, b, c, d\}$,
- $A = \{cabc, dd\}$,
- $R_n = \{(b, c)\$(c, a), ((ab)^n, c)\$(\lambda, d)\}$.

For each n, Γ_n generates $L_n = \{c(ab)^i c \mid i \geq 0\} \cup \{c(ab)^j d, c(ab)^j dd, c(ab)^j dc \mid j \geq n\} \cup \{c, dc, dd\}$. The smallest NFA M_n accepting this language has $2n + 4$ states, as can be shown using the extended fooling set method of [1], yielding the given bound. □

Turning to the other direction, converting an NFA into an equivalent finite extended splicing system, we can again show an upper and a lower bound.

Theorem 4 [21] *For the conversion of an NFA M to an equivalent finite extended splicing system Γ the following bounds hold, for some constant c*:

1. *Upper bound*: $\mathcal{A}(\Gamma) + \mathcal{R}(\Gamma) \leq c \cdot \mathcal{T}(M)$.
2. *Lower bound*: $\mathcal{A}(\Gamma) + \mathcal{R}(\Gamma) \geq \mathcal{T}(M)$.

Proof (sketch) For (1), let L be a regular language and $M = (Q, \Sigma, \delta, q_0, F)$ be a minimal NFA accepting L, where Q is the set of states, Σ the input alphabet, q_0 the initial state, F the set of final states and δ the set of transitions. We construct the extended finite splicing system $\Gamma = (V, \Sigma, A, R)$, where

- $V = \Sigma \cup Q \cup \{Z\}$, where Z is a new symbol not in $\Sigma \cup Q$,
- $A = \{ZZ\} \cup \{Zq_i a q_j Z \mid q_i, q_j \in Q, a \in \Sigma \cup \{\lambda\}, (q_i, a, q_j) \in \delta\}$,
- and R consists of the following rules:
  - $(\lambda, qZ)\$(Zq, \lambda)$ for all $q \in Q$,
  - $(Zq_0, \lambda)\$(\lambda, ZZ)$,
  - $(\lambda, q_f Z)\$(ZZ, \lambda)$ for all $q_f \in F$.

The initial language A contains all the words of the form $Zq_i a q_j Z$ such that M can pass from q_i to q_j on reading a. Thus, A is the set of all valid paths of length 1. The rules of the form $(\lambda, qZ)\$(Zq, \lambda)$ connect two paths such that the last state of the first path coincides with the first state of the second. Thus, we build all words in $Zq_i w q_j Z$ such that $\delta(q_i, w) = q_j$. The last two rules eliminate the initial state appearing at the beginning and the final state appearing at the end, obtaining words in L. It is easily verified that the size of Γ is linear in the size of M.

For part (2), we define an infinite sequence of languages $L_n, n \geq 1$, such that $L_n = \{a^n\}$. A minimal NFA M_n accepting L_n has $n + 1$ states and n transitions. Obviously, there exists an finite extended splicing system Γ with an empty set of rules and $\mathcal{A}(\Gamma) = n$ generating L_n. Assume that a smaller finite extended splicing system Γ_n exists. This means that $\mathcal{A}(\Gamma_n) + \mathcal{R}(\Gamma_n) \leq n - 1$. Now, by part (1) of Theorem 3, we can construct an equivalent NFA with $\mathcal{A}(\Gamma) + \mathcal{R}(\Gamma) + 1 \leq n$ states. Since we chose L_n to need $n + 1$ states, this is a contradiction. □

3.2.3 Decidability Questions

It is known that some decidability questions for NFA such as, e.g., membership or emptiness are solvable in polynomial time whereas the problems of equivalence or inclusion are known to be hard, namely PSPACE-complete [31]. It is an easy observation that any finite splicing systems can be converted to an equivalent NFA and vice versa in polynomial time. Thus, every decidability question, which is solvable for NFA in polynomial time, is solvable in polynomial time for finite splicing systems as well. On the other hand, problems being hard for NFA are hard for finite splicing systems as well. Altogether, we obtain the following theorem.

Theorem 5 [21] *The following problems are solvable in polynomial time for a given finite splicing system*:

1. *membership*
2. *emptiness*

3. *finiteness*

The following problems are not solvable in polynomial time for finite splicing systems unless P = PSPACE:

1. *equivalence*
2. *inclusion*
3. *universality*

Finally, we mention here that also computational complexity for splicing systems has been studied [24, 25]. Time complexity is defined in terms of the number of 'rounds' of rule application needed to generate a word, where space complexity is defined as the size of a production tree of the word. These notions are mostly relevant for universal models of splicing, but the previous section shows that this need not exclude finite systems from consideration.

3.3 Accepting Splicing Systems

When we think of a splicing system as a possible device of molecular computation, it becomes rather natural to consider an accepting variant. Indeed, such computations typically involve finding a solution to a specific given problem.

An accepting splicing system (introduced in [22], further studied in [21]) is a quadruple $\Gamma = (V, A, R, \underline{YES}, \langle\, , \rangle)$ where $\underline{YES}, \langle\, , \rangle \in V$ and $H_\Gamma = (V, A, R)$ is a splicing system. Let $\Gamma = (V, A, R, \underline{YES}, \langle\, , \rangle)$ be an accepting splicing system. We say that Γ *accepts* a word $w \in V^*$ if and only if the following condition holds:

$$\underline{YES} \in \sigma_R^k\big(A \cup \{\langle w \rangle\}\big) \quad \text{for some integer } k.$$

Thus, the language accepted by Γ is defined as

$$L(\Gamma) = \big\{w \in V^* \mid \Gamma \text{ accepts } w\big\}.$$

An extended accepting splicing system $\Gamma = (V, T, A, R, \underline{YES}, \langle\, , \rangle)$ is defined similarly as in the generating case.

One of the most interesting aspects of accepting splicing systems is that there does not seem to be any straightforward way to translate results from the generating case to the accepting variant. So, while it is fairly easy to show that all regular languages can be recognized by accepting splicing systems (see [21]), we do not yet have a Pixton-like construction that shows their inclusion in the regular languages.

Other indications of this difference, and of the interest of accepting systems come from the results on their descriptional complexity [21].

Theorem 6 *There is no polynomial function f such that for any finite extended accepting splicing system Γ there exists an equivalent finite extended generating splicing system Γ' such that $f(\mathsf{Size}(\Gamma)) \geq \mathsf{Size}(\Gamma')$.*

This theorem follows from the results presented earlier in this section and the following theorem.

Theorem 7 *There is no polynomial function f such that for any finite extended accepting splicing system Γ there exists an equivalent NFA M such that $f(\mathsf{Size}(\Gamma)) \geq \mathsf{Size}(M)$.*

Proof (sketch) We can show that the intersection of two regular languages can be accepted by an extended finite accepting splicing system of size $O(m+n)$, where m and n are the sizes of the systems accepting each of the languages. On the other hand, it is known that for two minimal NFAs M_n and M_m accepting the languages $(a^n)^*$ and $(a^m)^*$, where n and m are co-primes, any NFA accepting the intersection of these languages has at least $\mathcal{Q}(M_n) \cdot \mathcal{Q}(M_m)$ states. We define an infinite series of languages as follows. For each $k \in \mathbb{N}$, let L^k denote the language $(a^k)^*$. Moreover, let p_i denote the ith prime number. We now define an infinite sequence of languages $L_n, n \geq 1$, where

$$L_n = \bigcap_{i=1}^{n} L^{p_i}.$$

Any NFA accepting L^k needs at least k states. By the intersection result stated above, for any NFA M_n accepting L_n

$$\mathcal{Q}(M_n) \geq \prod_{i=1}^{n} p_i.$$

Now, if a polynomial function f exists such that $f(\mathsf{Size}(\Gamma)) \geq \mathsf{Size}(M)$, using the intersection result for accepting splicing systems, it is possible to construct an automaton M_n of size polynomial in n. But since $\mathcal{Q}(M_n) \geq \prod_{i=1}^{n} p_i \geq 2^n$, this is a contradiction. So there exists no such f. □

4 Open Questions and Further Research

As we have seen in Sect. 2, some of the main questions in the theory of splicing have been open since the introduction of splicing systems. Specifically, there are two main questions.

1. A characterization of the family of languages generated by basic finite splicing systems.
2. An algorithm to decide whether a given language is a splicing language.

Many specific subproblems of these basic questions or related questions can be formulated.

Closely related to the above problems is the issue of finding a procedure to construct a splicing system for a language. It is striking that after many years of research, we do not have any systematic way to go about constructing a splicing system for a given language.

One relevant question is understanding the relation of regular splicing languages with constants. More precisely, it has been conjectured [10] that a splicing language must contain a constant. If this is the case, as pointed out in [15], the structure of a regular splicing language should be strictly related to constants of the language.

The characterization of reflexive splicing languages given in [7] shows that a splicing language can be expressed as a finite union of languages obtained by splicing a finite set of constant languages by means of constant splicing rules. We wonder whether a similar characterization could hold in the general case, that is, can we find a finite set of languages and rules such that any splicing language is obtained by finite splicing of languages?

Also, the new directions discussed in Sect. 3 offer a variety of research questions.

Descriptional complexity aspects are closely related to the issues mentioned above, and could be a way to approach these open problems. For instance, finding theoretical lower and upper bounds on the size increase when converting an FA or equivalent representation of a splicing language to a splicing system, could be helpful in finding a procedure to construct a splicing system for this language.

Also, the notion of minimality of a splicing system would be interesting to explore. We believe that the notions introduced in Sect. 3.2 would be well suited for such investigations.

In this respect, questions to address would include finding algorithms for

- deciding whether a given splicing system is minimal,
- finding the minimal size of a splicing system for a given language,
- minimizing a splicing system.

For accepting splicing systems, the same series of basic questions can be addressed as for the generating variant. As we mentioned in Sect. 3.3, so far no straightforward way of relating the two versions is known. Thus, it would be very interesting to also find a characterization for the accepting variant, as well as algorithms for recognizing accepting splicing languages and for constructing an accepting splicing system for such a language.

Also, issues of minimality would be worth exploring here. The results of Sect. 3.3 suggest that the results would be significantly different from those for generating systems. Moreover, comparing the minimal generating splicing system and the minimal equivalent accepting system might provide important insights into the working of splicing.

Finally, the non-preserving definition for splicing systems opens the way to several directions of further work. First of all, the universality of finite systems under the non-preserving definition allows to explore the algorithmic possibilities of the basic splicing operation without extra features. In this way, we could conceive problem solving algorithms based on these finite systems. In addition, computational complexity issues can be studied for these systems. We already have well-defined complexity measures for splicing systems, touched upon in Sect. 3.2, which could also be applied in this case. These original measures are based on extended splicing systems with regular rules, and with them, splicing complexity classed were characterized in terms of well-known Turing machine classes. It would be very interesting to extend this line of work to non-preserving systems, given that these systems only rely on finite splicing rules and thus offer a fairer reflection of the power of splicing.

In all, we believe that finite splicing systems still leave ample room for further investigation.

References

1. Birget J-C (1992) Intersection and union of regular languages and state complexity. Inf Process Lett 43:185–190
2. Bonizzoni P, De Felice C, Mauri G, Zizza R (2004) Circular splicing and regularity. Theor Inform Appl 38:189–228
3. Bonizzoni P, De Felice C, Mauri G, Zizza R (2006) Linear splicing and syntactic monoid. Discrete Appl Math 154(3):452–470
4. Bonizzoni P, De Felice C, Mauri G, Zizza R (2005) On the power of circular splicing. Discrete Appl Math 150(1–3):51–66
5. Bonizzoni P, De Felice C, Mauri G, Zizza R (2003) Decision problems on linear and circular splicing. In: Ito M, Toyama M (eds) Proceedings of the DLT 2002. Lecture notes in computer science, vol 2450. Springer, Berlin, pp 78–92
6. Bonizzoni P, De Felice C, Mauri G, Zizza R (2003) Regular languages generated by reflexive finite linear splicing systems. In: Proceedings of the DLT 2003. Lecture notes in computer science, vol 2710. Springer, Berlin, pp 134–145
7. Bonizzoni P, De Felice C, Zizza R (2005) The structure of reflexive regular splicing languages via Schützenberger constants. Theor Comput Sci 334(1–3):71–98
8. Bonizzoni P, Ferretti C, Mauri G (1998) Splicing systems with marked rules. Rom J Inf Sci Technol 1(4):295–306
9. Bonizzoni P, Ferretti C, Mauri G, Zizza R (2001) Separating some splicing models. Inf Process Lett 79(6):255–259
10. Bonizzoni P, Mauri G (2006) A decision procedure for reflexive regular splicing languages. Dev Lang Theory 315–326
11. Calude CS, Păun Gh (2001) Computing with cells and atoms: an introduction to quantum, DNA and membrane computing. Taylor & Francis, London
12. Culik K, Harju T (1991) Splicing semigroups of dominoes and DNA. Discrete Appl Math 31:261–277
13. De Luca A, Restivo A (1980) A characterization of strictly locally testable languages and its application to semigroups of free semigroup. Inf Control 44:300–319
14. Goode E (1999) Constants and splicing systems. PhD thesis, Binghamton University
15. Goode E, Pixton D (2007) Recognizing splicing languages: syntactic monoids and simultaneous pumping. Discrete Appl Math 155:988–1006
16. Head T (1987) Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviours. Bull Math Biol 49:737–759
17. Head T (1998) Splicing languages generated with one sided context. In: Păun Gh (ed) Computing with bio-molecules. Theory and experiments. Springer, Singapore
18. Head T, Păun Gh, Pixton D (1996) Language theory and molecular genetics: generative mechanisms suggested by DNA recombination. In: Rozenberg G, Salomaa A (eds) Handbook of formal languages, vol 2. Springer, Berlin, pp 295–360
19. Kim SM (1997) Computational modeling for genetic splicing systems. SIAM J Comput 26:1284–1309

20. Loos R (2006) An alternative definition of splicing. Theor Comput Sci 358:75–87
21. Loos R, Malcher A, Wotschke D (2008) Descriptional complexity of splicing systems. Int J Found Comput Sci 19(4):813–826
22. Loos R, Martín-Vide C, Mitrana V (2006) Solving SAT and HPP with accepting splicing systems. In: PPSN IX. Lecture notes in computer science, vol 4193. Springer, Berlin, pp 771–777
23. Loos R, Mitrana V (2007) Non-preserving splicing with delay. Int J Comput Math 84(4):427–436
24. Loos R, Ogihara M (2007) Complexity theory for splicing systems. Theor Comput Sci 386:132–150
25. Loos R, Ogihara M (2007) Time and space complexity for splicing systems. Theory Comput Syst (in press)
26. Păun Gh, Rozenberg G, Salomaa A (1996) Computing by splicing. Theor Comput Sci 168(2):321–336
27. Păun G, Rozenberg G, Salomaa A (1998) DNA computing, new computing paradigms. Springer, Berlin
28. Perrin D (1990) Finite automata. In: Van Leeuwen J (ed) Handbook of theoretical computer science, vol B. Elsevier, Amsterdam, pp 1–57
29. Pixton D (1996) Regularity of splicing languages. Discrete Appl Math 69:101–124
30. Schützenberger MP (1975) Sur certaines opérations de fermeture dans le langages rationnels. Symp Math 15:245–253
31. Stockmeyer L, Meyer AR (1973) Word problems requiring exponential time: preliminary report. In: Fifth annual ACM symposium on theory of computing, pp 1–9
32. Verlan S, Zizza R (2003) 1-splicing vs. 2-splicing: separating results. In: Proceedings of Words03, Turku, Finland, pp 320–331
33. Verlan S (2004) Head systems and applications to bio-informatics. PhD thesis, University of Metz

Formal Models of the *Calyx of Held*

**Andrea Bracciali, Marcello Brunelli,
Enrico Cataldo, and Pierpaolo Degano**

Abstract We survey some recent work on the behavior of the *calyx of Held* synapse. The analysis considered are based on formal and quantitative models aimed at capturing emerging properties about signal transmission and plasticity phenomena. While surveying work about a specific and real-scale biological system, we distinguish between *deterministic* and *stochastic* approaches. We elaborate on the fact that in some cases, as in the *calyx*, the latter ones seem to be more adequate. The stochastic models, which we have developed, are based on a computational interpretation of biological systems. We illustrate the advantages of this approach in terms of expressiveness.

1 Introduction

A *systemic* approach to the study of living systems and the properties emerging from their behavior is becoming recognized as a needed, although articulate, methodology for understanding many biological phenomena. This has recently been embodied in the *Systems Biology* research field [11, 35]. Within it, computer science can provide models and analysis tools, far beyond its traditional contribution to categorizing acquired knowledge. This is justified by a reading of biological dynamics in terms of computational processes—"Cells as computation" [52].

This paper is a survey of some studies on a specific biological domain: *the calyx of Held* synapse. Several are deterministic, while others combine computational models and stochastic semantics. Overall, we illustrate how these systemic and formal methods considerably contribute to address some of the many not understood biological mechanisms. More specifically, we discuss some desirable properties enjoyed by the computational and stochastic approach, such as adequacy, expressiveness, and compositionality.

Neuroscience is experiencing an explosive growth in the amount of detailed and high quality experimental data on neural processes underlying behavior. Neurons represent the elementary components of the nervous systems, able to communicate with each other at highly specialized contact sites called *synapses*. In general, each neuron consists of a somatic cellular body, on which a variable number of thin elongated structures called *dendrites* converge and from which a long single structure called *axon* emerges, branching in several synaptic terminals. The synapses are

A. Bracciali (✉)
Dipartimento di Informatica, Università di Pisa, Pisa, Italy
e-mail: braccia@di.unipi.it

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_18, © Springer-Verlag Berlin Heidelberg 2009

the places of functional contacts between neurons, where the information is stored and transmitted from one (*pre-synaptic terminal*) to another (*post-synaptic terminal*) neuron, by releasing chemical molecules (*neurotransmitters*) [47]. Furthermore, the releasing process can be altered by repeated activity, making the signal transmission a *plastic* phenomenon, i.e. the synaptic terminal is then a kind of computational unit, which changes its output based on its previous activity and ongoing modulation [46, 76]. Plasticity is hence a basic mechanism for the processes of memory and learning.

Mathematical and computational models are necessary to describe and understand available data, with different methodologies and approaches according to the level of abstraction chosen for describing the systems [13, 14, 19, 37, 43, 60]. Traditionally, the chemical kinetics of neurons (and other bio-systems) has been described deterministically in terms of differential equations.

Several models have been proposed for phenomena like the intracellular signaling pathways, which underly most of the phenomena cited above, and at this scale, often the most appropriate ones are stochastic [15, 30, 63, 69, 72]. The major drawback of a deterministic description is due to the fact that cellular processes occur in a small volume frequently involving a very small number of interacting molecules. Significant stochastic fluctuations can hence occur, which can significantly affect the dynamics and can be seen as random variations about a mean number of molecules [63]. For instance, models for the circadian rhythm or genetic networks can oscillates or rests, depending on the description used. In these cases, the stochastic approach has to be preferred.

A significant part of the stochastic and systemic approach consists in the development of formal models which, beyond tackling the inherent complexity, allow for a faithful representation of phenomena at the right level of abstraction. Also, they have to be suitable for predictive simulations and automated processing. Then investigated phenomena can be precisely modeled and virtual experiments performed in silico. Such experiments may be easy and fast, and often result in satisfying approximations of their in vitro/vivo counterparts.

At this level of abstraction, the cited interpretation of life systems in terms of computational systems “cells as computation” seems definitely profitable, as properly recognized in [52]. This analogy is even stronger in the light of the relevance that *interaction* has progressively acquired in the theory of computation, starting from the early development of concurrent systems and with the recent advent of new computational paradigms, which are component-based, interactive, and distributed over open environments, such as service oriented architectures. Furthermore, stochastic semantics has been easily embedded in these computational frameworks, e.g. [29, 49].

According to this metaphor, cells, molecules and biological “active” components are assimilated to computer processes, the computational units of a concurrent software system. Biological interaction corresponds to process communication. By communicating, processes may exchange information or synchronize themselves. A biological experiment has then a direct correspondence with a computation. This justifies the attempt to exploit all those modeling and analysis techniques, extensively developed for understanding computational systems, within systems biology.

That is, biological processes cannot only be simulated by *in silico* experiments, but it is also possible to formally reason about their computational models and infer properties of interest.

Technically, these models can be based on *process calculi*, developed for formally describing the behavior of systems composed by processes. Process behavior is defined in terms of basic interaction steps, e.g. communication or synchronization. Compositional operators, like sequential, parallel, or non-deterministic compositions, allow more structured behavior to be expressed. Process calculi are generally equipped with an operational semantics consisting of a transition system defined by rules that, given the process representing the current state of the system, determine the next states it can evolve to. Often, these models and their analysis enjoy nice compositional properties, being system behavior defined in terms of the behavior of its components. The level of abstraction in representing system dynamics can be chosen by adopting suitable interaction and composition primitives. Several process calculi have been proposed for modeling the dynamics of living systems, e.g. [51], while others have been extended or defined with operators oriented to describing different aspects of biological interaction, such as specific abstractions for molecular interaction [44], compartments [50], and active membranes [9].

In the following, we recollect a number of formal models for the *calyx of Held* synapse recently appeared in literature so as to illustrate different systemic approaches to an open case of study.

We first discuss several deterministic models proposed for describing issues about the calcium triggered release of neurotransmitters in the synapse. Then we illustrate a model of the same and other aspects of the synapse behavior we have developed (see [5–7] for technical details). To our knowledge, ours is the first stochastic model of a neural terminal based on a process calculus. We present the models of the pre- and post-synaptic terminals and experiments regarding, for instance, plasticity and the dynamics of vesicle release and recruitment. The model of the whole synapse can be built from the pre- and post- synaptic models in a pure compositional manner, which is not always the case in the settings of stochastic models. The entire model allows plastic events to be observed throughout the whole synapse. This modeling effort seems to confirm the viability of the approach by the construction of a quite detailed model and, moreover, it allows for a pragmatic evaluation of its expressiveness. Neurons, the *calyx of Held* and some technical background are recapitulated in short.

2 Background

Some background from both biology and computer science is briefly recapitulated in this section and references to more technical treatments are provided.

2.1 Neurons

Information in neurons is in the form of space and time variation of the electrical potential across the cytoplasmatic membrane. While in a resting state, the neuron maintains an electrical polarization of about -70 mV between its interior and the cellular context. The signal (action potential) through the cell body and along the axon is transmitted in an all-or-none fashion as a sequence of depolarizations, up to $+40$ mV, and repolarizations, ending up at the synaptic terminal. These induce the opening of the calcium ion (Ca^{2+}) channels with a transient elevation of their concentration in the pre-synaptic terminal. These ions control the transmitter release process (calcium-triggered-release hypothesis) [75], consisting in the exocitosis of synaptic *vesicles* (small elements containing the neurotransmitters) located at the pre-synaptic so-called *active zone* [65]. It is worth noting that some chemical messengers and modulators regulate, intracellularly and extracellularly, respectively, the relationship between action potential and release in a synaptic terminal. This relationship is altered by repeated activity giving rise to plasticity [76]. The pre-synaptic terminal is then a kind of computational unit, which changes its output based on its previous activity and ongoing modulation [46, 76].

The arrival and binding of neurotransmitters on post-synaptic receptors trigger post-synaptic potentials, which can be excitatory or inhibitory. These consist of graded potentials that induce an action potential in the post-synaptic neuron when reaching a threshold value. Synapses involving neurotransmitters are chemical and the connection between the pre- and the post-synaptic neuron occurs in a variety of possible combinations, such as axodendritic, axosomatic, axoaxonic, or dendrodendritic. The last case implies that the flow of neural information is bidirectional. In addition to chemical synapses, there are electrical synapses which allow current to flow directly from one neuron to another. There are also non-synaptic interneuronal signals, such as volume transmission, involving glia, and ephaptic transmission, in which the extracellular electric field of one neuron influences the activity of another neuron. Beyond the above general properties, neurons can be distinguished according to their size, shape, membrane electrical and neurochemical properties, and connectivity. The traditional vision of the neuron as input-integration-output device must be replaced by that of a complex computational machines, like a spatio-temporal filter, a coincidence detector, a unit of internally distributed devices of memory stored locally or metabolic assemblies [2].

The ever increasing collection of data regarding anatomy and physiology of parts of different nervous systems still has no correspondence with a unifying theory of the brain. The difficulty resides in the multi-scale complexity of the these systems, in which information processing occurs simultaneously at several levels: molecular, channels and synapses, cellular, local networks, projections areas and receptive fields, brain systems, and brain-behavior [15].

Some neural systems, such as the mammalian neocortex, are characterized by connectivity reaching thousands of synaptic inputs for each neuron. To study these kind of biological neural networks, it is worthwhile to build network models of neuron-like units with the action potential replaced by the firing rate. A rate represents the probability per unit of time to have an action potential in a small time interval. The synaptic connections are represented by a set of parameters varying according to some bio-inspired rules. These seemingly oversimplified models can give many insights for the understanding of the collective behavior of large neural networks [28].

Models more adherent to biology need to also take into account the dynamic behavior of the membrane electrical potentials. At the most abstract side of this approach, there is the description of the action potentials as a string of numbers, representing the times at which they occur. This approach is rooted in the information theory and its principal goal is to uncover the neural code, that is the information content in a single action potential and in their relative time intervals. When the shape of the action potential and/or the neural spatial structure play a role, then the last description is inadequate and one must invoke models that describe the observed electrical behavior of the membrane, by representing it as resistive-capacitive circuits [1, 32, 36, 40, 53].

The intracellular processes underlying the phenomena cited above are described within deterministic or stochastic frameworks [15, 30, 63, 69, 72].

2.2 The Calyx of Held

The *calyx of Held* is a giant glutamatergic synapse of the mammalian auditory tract. Its post-synaptic target is the neuron of the medial nucleus of the trapezoid body. They belong to the relay pathway of the sound source localization in the auditory brainstem. Its size allows the manipulation of its intracellular biochemical composition, and this makes the *calyx of Held* an ideal model system for studying presynaptic mechanisms of transmission in central nervous system, specifically the intracellular pathway underlying exocitosys and its regulation, and the short-term plasticity. Spatially, the *calyx of Held* is structured as a parallel arrangement of a large array of active zones, ranging from 300 to almost 700 in number [56]. Active zones, each containing up to 10 vesicles, are clustered in groups of about 10 of them, in a volume having a diameter of almost 1 μm. Each action potential activates all the active zones. Each active zone is morphological similar to the active zones of the small nerve terminals.

This model system is one among few experimental preparations in which it has been possible to measure, simultaneously, neurophysiological parameters in both the pre- and post-synaptic terminals. Major attention has been directed to the study of the Ca^{2+} ionic channels for the role that Ca^{2+} plays in the exocitotic process. In addition, this is an excellent model system for addressing the mechanisms of short-term plasticity [76]. It has been shown that the release of vesicles is highly sensitive to Ca^{2+} concentration and a model has been proposed for describing this behavior [57]. The *calyx of Held* shows short-term facilitation [20], which seems dependent on the build up of a small but effective amount of residual Ca^{2+}. During prolonged stimulation, this synapse depresses caused by a depletion of the readily releasable vesicle pool according to [45, 70, 76]. Several models have been proposed to uncover some of the mechanisms underlying these plastic processes, all characterized by the assumption of continuity and determinism, which in these cases might not be appropriate.

2.3 Deterministic Models of Biochemical Pathways

Biochemical reaction networks are modeled with several approaches. At sufficiently low concentrations of the reactants (which does not mean small numbers of molecules), these processes are modeled with the Generalized Mass Action Rate Law (GMA) [61]. We describe this law (known as Law of Mass Action for simple processes) and the underlying hypotheses for the simple particular case of bimolecular reaction, in which a molecule A reversibily interacts with the molecule B to form a molecule C. The variables $A(t)$, $B(t)$ and $C(t)$ represent molecular concentrations (numbers of molecules per unit volume at time t). The assumptions underlying the GMA are: (1) the rate of formation of C is proportional to the joint concentrations of A and B; (2) each C molecule acts independently of other C molecules and has a given probability to decay in its constituents. The last assumption means that the C decay rate is proportional to the number of molecules C present. Conventionally:

$$A + B \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} C$$

where k_1 and k_{-1} are the rate constants. The following system of Ordinary Differential Equations, in short ODEs corresponds to the above kinetic schema:

$$\frac{dC}{dt} = k_1 AB - k_{-1}C, \qquad \frac{dA}{dt} = -k_1 AB + k_{-1}C, \qquad \frac{dB}{dt} = -k_1 AB + k_{-1}C.$$

Given initial conditions, the solution of the above system provides $A(t)$, $B(t)$ and $C(t)$. An extensive treatment of biochemical ODEs modeling is in [67], a critical analysis on the adequacy of ODEs deterministic description of biochemical processes and of the, sometimes subtle, relationship between the continouos deterministic approach and the stochastic discrete one is in [71].

2.4 The Stochastic Simulation Algorithm

The description of biochemical networks is carried out within a stochastic kinetics framework [66] when considering, for example, a small volume where only few molecules interact [21, 23, 24, 33, 69]. A system of N chemical species $S_1 \ldots S_N$

interacting through M elemental (instantaneous) chemical reactions (uni- or bimolecular) $R_1 \ldots R_M$ could be fully described, in principle, by resolving the trajectories of all the individual molecules. This approach is computationally untractable.

The study simplifies notably when the system is well-stirred: molecules are randomly distributed in a given volume V and with assigned temperature T. In this case, the system is completely described by knowing the time evolution of the vector $\mathbf{X}(t) = (X_1(t), \ldots, X_N(t))$, from a given starting value $\mathbf{X}(t_0) = \mathbf{X}_0$, with $X_i(t)$ representing the number of molecules of the species i at time t. The R_i are characterized by two quantities: the propensity function $a_i(\mathbf{x})$ (where, for the sack of simplicity, we write $\mathbf{x}$ for $\mathbf{X}(t)$) and the state-change vector $\boldsymbol{\nu}_i$.

The definition of the propensity function $a_i(\mathbf{x})$ is such that $a_i(\mathbf{x})\,dt$ represents the probability that the reaction R_i occurs in the volume V in the time interval $[t, t+dt]$. More specifically, one wants to find a probability rate constant c_i, such that $c_i\,dt$ represents the probability that any two, or one, molecules, randomly chosen amongst those reacting according to R_i, will react in the time interval dt. By knowing c_i and summing over all the possible distinct combination of R_i reacting molecules (addition law of probability), it is immediate to show that for the R_i unimolecular reaction $S_j \rightarrow$ *products*, then $a_i(\mathbf{x}) = c_i x_j$, while for a bimolecular reaction $S_j + S_p \rightarrow$ *products* $a_i(\mathbf{x}) = c_i x_j x_p$. When the reaction R_i occurs, the state vector $\mathbf{x}$ becomes $\mathbf{x} + \boldsymbol{\nu}_i$.

Given that the description is probabilistic, one wants to know what is the probability $P(\mathbf{x}, t|\mathbf{x}_0, t_0)$ to find the system in the state $\mathbf{X}(t) = \mathbf{x}$, given that it was in the state $\mathbf{X}(t_0) = \mathbf{x}_0$ at time t_0.

The equation for the time evolution of the probability can be written as the sum of the probabilities of the ways, mutually exclusive, in which the system can evolve from a state $\mathbf{X}_0$ at time t_0 to a state $\mathbf{x}$ at time $t + dt$:

$$P(\mathbf{x}, t+dt|\mathbf{x}_0, t_0) = P(\mathbf{x}, t|\mathbf{x}_0, t_0)\left[1 - \sum_{j=1}^{M} a_j(\mathbf{x})\,dt\right] + \sum_{j=1}^{M} P(\mathbf{x} - \boldsymbol{\nu}_j|\mathbf{x}_0, t_0) a_j(\mathbf{x} - \boldsymbol{\nu}_j)\,dt.$$

By rearranging the previous equation, with dt however small, one obtains the Chemical Master Equation (CME):

$$\frac{\partial P(\mathbf{x}, t|\mathbf{x}_0, t_0)}{\partial t} = \sum_{j=1}^{M} [a_j(\mathbf{x} - \boldsymbol{\nu}_j) P(\mathbf{x} - \boldsymbol{\nu}_j|\mathbf{x}_0, t_0) - a_j(\mathbf{x}) P(\mathbf{x}, t|\mathbf{x}_0, t_0)].$$

Note that the CME represents a set of coupled differential equations, as many as all the possible molecular composition of the system, a number which is prohibitively large. Anyway, even for very simple systems, the numerical solution of the CME is out of question because the transition probability matrix becomes rapidly untractable [18, 62].

Daniel Gillespie proposed a way to overcome these difficulties, the Stochastic Simulation Algorithm (SSA), by using a so-called inversion method of Monte Carlo theory [23, 24]. The approach consists in finding numerical realization of $\mathbf{X}(t)$ as function of t.

Just to gain some intuition on the SSA, we recall that the key for generating simulated trajectories of $\mathbf{X}(t)$ is the function $P(\tau, i)$, with $P(\tau, i)\,d\tau$ representing the probability that the next reaction will be of type R_i and will occur in the interval $(t + \tau, t + \tau + d\tau)$, in a given volume and with the system in a state $\mathbf{X}(t)$ (written as usual as $\mathbf{x}$) at time t. Now, $P(\tau, i)$ can be expressed as $P_1(\tau) \times P_2(i, \tau)$, where $P_1(\tau)\,d\tau$ is the probability that the next reaction will occur in the interval $(t + \tau, t + \tau + d\tau)$ and $P_2(i, \tau)$ is the probability that the next reaction will be of the type R_i.

Gillespie showed that $P(\tau, i) = a_i(\mathbf{x}) \exp(-a_0(\mathbf{x})\tau)$, where $a_0(\mathbf{x}) = \sum_{k=1}^{M} a_k(\mathbf{x})$, and suggested to calculate the time and the type of the next reaction by drawing two random numbers r_1 and r_2. It can be shown that the time of the next reaction is $\tau = (1/a_0(\mathbf{x})) \times \ln(1/r_1)$. The next reaction R_i to occur will be the one for which the number i is the smallest integer satisfying the relation $\sum_{k=1}^{i} a_k(\mathbf{x}) > r_2 a_0(\mathbf{x})$. When the time and the kind of reaction have been found, the number of the molecules taking part to that reaction are updated. The process can run indefinitely, as far as reactions may occur (typically a maximal duration of the simulation interval is fixed).

Summing up the SSA is: (1) Initialize the time and the system: $t = t_0$, $\mathbf{x} = \mathbf{x}_0$; (2) Evaluate all the $a_k(\mathbf{x})$ and $a_0(\mathbf{x})$; (3) Generate τ and i as indicated above; (4) Replace the time t with $t + \tau$ and the number of molecules $\mathbf{x}$ with $\mathbf{x} + \boldsymbol{\nu}_i$; (5) Return to step 2.

The SSA can be applied to homogeneous volumes, it can handle reactions but not diffusion. The Chemical Master Equation is a special form of the Master Equation for jump processes governed by chemical reactions [66]. Note that the Gillespie algorithm can be applied to all continuous time Markov jump processes [18, 62].

2.5 Process Calculi and the Stochastic Pi-calculus

Process calculi have been defined within concurrency theory to formally represent the observable behavior of a system made of interacting components, also called processes. The observable behavior is an abstraction of the actual behavior of the components at the desired level of abstraction. Process calculi are based on an algebraic notation whose main ingredients are

(i) the basic *actions* that components can perform together with a set of operators for defining *processes* by composing basic actions and simpler processes;
(ii) a *semantics* describing the possible dynamics of a process according to the actions it can perform. Often it is operational and consists of a *labeled transition system* (*LTS*). LTS have *states*, roughly processes themselves, and *transitions*, i.e. a relation among a state and the next one the process can evolve to. Transition may have *labels* recording relevant information about the step, such as

the observable behavior. LTS are usually defined compositionally by inductive rules over the structure of the components of a process;

(iii) *properties* and *methodologies* to analyse process behavior. Examples of properties can be equivalence relations, such as bi-simulations, or reachability properties, such as checking whether a process can reach a given state of interest. Examples of analysis techniques are model checking, i.e. verifying whether an LTS can fulfill a formula of a suitable logic expressing the property of interest, and static analysis, which reasons about an abstraction of an LTS to check properties. These methods can often be automated.

The Pi-calculus [42] models systems of concurrent communicating processes whose communication network may also evolve in time. It is based on the notion of *name*. Basic actions are communications through shared channels. Channels are identified by a name. Informally, $a!n$ represents sending message n trough channel a, while $a?m$ associates the message received through channel a to (variable) m. One of the operators of the calculus is sequential composition, written _ ._. Given a process P', the process $P = a!n.P'$ can perform the output action and then evolve to P'. Such a general behavior is formalized by a semantic rule as

$$a!m.P' \xrightarrow{am} P'$$

with the label am recording the channel and the message. Analogously,

$$a?m.P' \xrightarrow{an} P'[n/m]$$

where n can be any received message and $P'[n/m]$ stands for the fact that, as effect of the communication, the value of m within P' is n. Another operator is parallel composition _ |_. In $P|Q$, the two processes can either evolve independently, e.g. $P|Q \xrightarrow{an} P'|Q$ if $P \xrightarrow{an} P'$, or communicate through a shared communication channel, if any, i.e. they can execute complementary input/output actions on a channel identified by the same name. Communication is formalized by an inductive rule as

$$\frac{P \xrightarrow{a(n)} P' \quad Q \xrightarrow{an} Q'}{P|Q \xrightarrow{\tau} P'|Q'}.$$

This rule can be applied to generate a transition for a process C whenever it has the structure $P|Q$, as required by the conclusion of the rule (lower row). For instance, if $C = a!n.A'|a?(m).B'$ the rule can be applied ($P = a!n.A'$ and $Q = a?(m).A'$) and, since its premises are satisfied (upper row, $P = a!n.A' \xrightarrow{an} A' = P'$ and $Q = a?(m).B' \xrightarrow{an} B'[n/m] = Q'$ are justified by the previous rules), the transition in the conclusion exists:

$$a!n.A'|a?(m).B' \xrightarrow{\tau} A'|B'[n/m].$$

Note that labels in the premises require complementary actions, while the label τ denotes an action internal to $A|B$, e.g. it does not expose any channel for possible communications. This rule is *compositional*, as the behavior of $P|Q$ is defined in terms of the behavior of its components P and Q and it is *abstract* as only the observable behavior of P and Q is taken into consideration and not, for instance, their internal structure. Importantly, channel names themselves can be exchanged through communications, hence modeling the dynamical reconfiguration of the system communication network. Non-determinism is represented by rules like

$$\frac{P \xrightarrow{an} P'}{P + Q \xrightarrow{an} P'}$$

and symmetrically for Q. One of the two alternative processes is non-deterministically selected to let the system evolve, while the other is discharged.

A stochastic version of Pi-calculus has been defined in [29, 49]. Communications are annotated with a rate r representing the probability distribution for the action happening within a given time, e.g. $a!n@r$. This extension has been motivated by the study of system performances depending on the possible occurrence of events, such as the distribution of incoming requests or task completion times. From the rates and the current state of a process, accounting for the number of possible actions ready to be performed, the next transition can be stochastically determined. Clearly, this paves the way for an application of the SSA algorithm, when, in biological settings, processes are read as reactants, communications as reactions and the propensity functions are determined from rates, which now represent “the speed” of a reaction (see [10] for details). Several simulation tools have been developed, e.g. the Stochastic Pi-calculus Machine (SPiM) [48], to cite one. Once that processes/active biological components have been defined, together with their rates and the initial conditions of the experiment, a stochastic trajectory (i.e. a computation) is computed, allowing the behavior of the system to be analysed.

3 Deterministic Models

In this section, we describe a selection of experimental and theoretical work which have addressed the issue of synaptic plasticity in the *calyx of Held*. All these models are based on continuity assumptions and they all use Ordinary Differential Equations, in short ODEs. These models fit with the experimental data, in the range of variable values investigated, presenting limitations when considering events at smaller scales, in which the discrete aspects of the behavior emerge.

3.1 The High Ca^{2+} *Sensitivity of Vesicle Release*

In the process of synaptic transmission, a fundamental step is the vesicle release triggered by a transient elevation of the intracellular Ca^{2+}. It has been experimentally shown by [57] that the Ca^{2+} peak concentrations needed for triggering release are of the order of 10–20 μM, much smaller of the commonly accepted values of

100–300 μM. The experimental technique was the so-called Ca^{2+} un-caging, during which the intra-celluar cytosolic Ca^{2+} is elevated in a spatially homogeneous way. Under this condition, the measured signal due to the fluorescent Ca^{2+} indicators has been directly linked to the Ca^{2+} sensed by the vesicles. The amount of released vesicles has been obtained by deconvolution of the recorded excitatory post-synaptic currents. This first requires to chemically suppress the post-synaptic fast glutamatergic receptor desensitization (desensitization is a decreased responsiveness of a receptor for a stimulus). It was observed that for a Ca^{2+} concentration greater than 12 μM, most of the release occurred within 3 ms. By inducing several different cytosolic Ca^{2+} concentrations and calculating the correspondent cumulative and rate of release, Schneggenburger and Neher [57] have built a minimal kinetic model of the Ca^{2+} dependent release, consisting of five Ca^{2+} binding steps. This sequence ends with vesicle fusion coming from the fully Ca^{2+} bound state. The kinetic model consisted of a deterministic rate equations, whose parameters were determined by a fitting procedure:[1]

$$\mathrm{Ca}_i^{2+} + V \underset{k_{\mathrm{off}}b^0}{\overset{5k_{\mathrm{on}}}{\rightleftharpoons}} V_{\mathrm{Ca}_i^{2+}} + \mathrm{Ca}_i^{2+} \underset{2k_{\mathrm{off}}b^1}{\overset{4k_{\mathrm{on}}}{\rightleftharpoons}} V_{2\mathrm{Ca}_i^{2+}} + \mathrm{Ca}_i^{2+} \underset{3k_{\mathrm{off}}b^2}{\overset{3k_{\mathrm{on}}}{\rightleftharpoons}} V_{3\mathrm{Ca}_i^{2+}} + \mathrm{Ca}_i^{2+},$$

$$V_{3\mathrm{Ca}_i^{2+}} + \mathrm{Ca}_i^{2+} \underset{4k_{\mathrm{off}}b^3}{\overset{2k_{\mathrm{on}}}{\rightleftharpoons}} V_{4\mathrm{Ca}_i^{2+}} + \mathrm{Ca}_i^{2+} \underset{5k_{\mathrm{off}}b^4}{\overset{k_{\mathrm{on}}}{\rightleftharpoons}} V_{5\mathrm{Ca}_i^{2+}} \xrightarrow{\gamma} T$$

where Ca_i^{2+} represents the molar concentration of intracellular calcium ions, V the number of vesicles, and T the released vesicles. The kinetic rate constant relative to vesicle release is $\gamma = 6000\ \mathrm{s}^{-1}$; the other values are $k_{\mathrm{on}} = 9 \times 10^7\ \mathrm{M}^{-1}\,\mathrm{s}^{-1}$, $k_{\mathrm{off}} = 9500\ \mathrm{s}^{-1}$ and $b = 0.25$.

These equations were driven by transient Ca^{2+} curves with decays and amplitudes compatible with experimental findings, obtained by solving a system of differential equations.

[1] Slightly imprecisely, hereafter we shall use the notation

$$A + B \underset{\text{back}}{\overset{\text{to}}{\rightleftharpoons}} C + D \underset{\dots}{\overset{\dots}{\rightleftharpoons}} \cdots$$

to represent the reversible reaction from A and B to C and back. Then the produced C is involved, together with D, in the next (reversible) reaction producing.... That is, the formula does not intend to represent neither $A + B \leftarrow C + D$ nor $A + B \rightarrow C + D$.

3.2 Pre-synaptic Facilitation

Facilitation is a form of activity-dependent synaptic enhancement: the synaptic strength increases during a train of action potentials [20, 75]. It is generally accepted that the intracellular *residual* Ca^{2+} remaining from previous activity causes facilitation [76]. However, this is controversial, since the residual Ca^{2+} is very small and seems not sufficient to induce facilitation. Other mechanisms have been suggested, among which the permanence of Ca^{2+} bound to the high affinity binding sites of the secretory machinery or Ca^{2+} buffer saturation [20], just to cite a few.

By using the Ca^{2+} uncaging method described above and starting from [57], the authors of [20] have shown that facilitation can be ascribed mainly to the small residual Ca^{2+}. Some of their experiments consisted in the use a double Ca^{2+} pulse protocol, with a varying inter-stimulus interval, to asses the amplitude and the decay kinetics of facilitation. Felmy et al. [20] found a consistent facilitation at inter-stimulus interval of few ms, but this was neither due to an increase of the Ca^{2+} sensitivity of transmitter release nor to an increase in the size of the readily releasable vesicle pool. In [20], the uncaging technique was used to induce a controlled and small (less than 1 μM) Ca^{2+} concentration increase to reveal facilitation. The model equations were driven by Ca^{2+} functions of time obtained as solutions of systems of differential equations. The residual Ca^{2+} was modeled by adding a Ca^{2+} concentration of about 2 μM. The simulations showed an increased vesicle release, but only a fraction of that observed experimentally. Hence, other mechanisms for facilitation cannot be excluded.

3.3 Pre-synaptic Potentiation

Synaptic potentiation is a form of synaptic plasticity in which the synaptic strength increases, e.g. because the number of the releasable pool vesicles increases. Experiments on the *calyx of Held* have shown that synaptic potentiation of spontaneous and evoked vesicle release can be induced by phorbol esters, which increases the apparent Ca^{2+} sensitivity of vesicle fusion [39]. Lou et al. [39] have used Ca^{2+} uncaging and have developed a deterministic allosteric model of Ca^{2+} activation of vesicle fusion, extending the model in [57]:

$$\begin{tikzcd}
\mathrm{Ca}_i^{2+} + V \arrow[r, shift left, "5k_{\mathrm{on}}"] \arrow[d, "l_+ f^0"] & V_{\mathrm{Ca}_i^{2+}} + \mathrm{Ca}_i^{2+} \arrow[l, shift left, "k_{\mathrm{off}} b^0"] \arrow[r, shift left, "4k_{\mathrm{on}}"] \arrow[d, "l_+ f^1"] & \cdots V_{4\mathrm{Ca}_i^{2+}} + \mathrm{Ca}_i^{2+} \arrow[l, shift left, "2k_{\mathrm{off}} b^1"] \arrow[r, shift left, "k_{\mathrm{on}}"] \arrow[d, "l_+ f^4"] & V_{5\mathrm{Ca}_i^{2+}} \arrow[l, shift left, "5k_{\mathrm{off}} b^4"] \arrow[d, "\gamma"] \\
T & T + \mathrm{Ca}_i^2 \cdots & \cdots T + 4\mathrm{Ca}_i^2 & T + 5\mathrm{Ca}_i^2
\end{tikzcd}$$

where $k_{\mathrm{on}} = 10^8\ \mathrm{M}^{-1}\,\mathrm{s}^{-1}$; $k_{\mathrm{off}} = 4000$; $l_+ = 0.0002$; $f = 31.3$; $l_+ f^5 = 6000 = \gamma$; $b = 0.5$. The value of the fusion rate is $l_+ = 0.0002$. Potentiation in presence of phorbol ester was simulated by increasing fivefold the fusion rate l_+. The new model explains the spontaneous release and the smaller Ca^{2+} cooperativity in the release process. The potentiation is more evident at lower Ca^{2+} concentrations. The model differs from that in [57] mainly in the range of small Ca^{2+} concentration, in which the continuity assumption might break down.

3.4 Pre-synaptic Depression

Experimental data attest that the vesicles of the active zone can be split in two virtually separated and equally populated pools. One pool consists of the so-called readily releasable or *fast* vesicles, and the other one of so-called reluctantly releasable or *slow* vesicles [45, 54, 68, 76]. The vesicles released during a single action potential discharge belong to the fast pool and the amount of release corresponds to about 4% of it [20].

When the synapse is depolarized to elicit maximal Ca^{2+} influx, all the fast and slow vesicles of the active zones are released and the synapse active zone is completely depleted. This represents one of the proposed mechanisms for *synaptic depression*, which is a reduction of the synaptic strength that builds up during sustained neural activity. In the release process, most of the fast and slow vesicles are released with the characteristic times of about 3 and 30 ms, respectively [45, 59]. It is also known that the slow vesicles are replaced much more rapidly than the fast ones. The precise mechanisms through which this form of release happens are, however, still debated. For some cells, it is likely that the slow vesicles differ in their intrinsic kinetics from the fast ones [64]. For the *calyx of Held* synapse, some deterministic mechanisms of depression have been suggested. One of these assumes that the slow vesicles are precursors of the fast ones and that they become releasable by moving toward the fast pool [45].

Another model built upon [39] suggests that the time course of release observed during the experiments of Ca^{2+} un-caging might be due to kinetic mechanisms intrinsic to the vesicle fusion machinery of fast and slow vesicles [70]. These mechanisms differ in the value of the coefficient l_+, which is 2×10^{-4} s^{-1} for the fast pool and 7×10^{-6} s^{-1} for the slow pool. The slow vesicles are re-integrated much faster (order of hundreds milliseconds) than the fast vesicles (order of seconds). This model describes fairly well most of the features of vesicles release induced by Ca^{2+} un-caging, but it did not account for the so-called sub-maximal release, that is the increased number of fast vesicles released as the un-caging step of Ca^{2+} increases. Such findings suggest other mechanisms and/or the need of different approaches in the description of these events. In addition, sustained neural activity accelerates the recruitment of the fast vesicles, via intra-cellular Ca^{2+} reaching a speed up to ten times the normal one. A deterministic model for describing this behavior is in [31], under the hypothesis that a global Ca^{2+} concentration drives the recruitment of vesicles to the fast pool.

3.5 Post-synaptic Receptor Desensitization

Synaptic depression can be due to the occurrence of vesicle depletion and/or to post-synaptic receptor desensitization [25–27, 76]. Computational models help to distinguish the relative contributions of these two mechanisms [73]. A model of post-synaptic fast glutamatergic receptors is in [25], and it obeys the following state equations:

$$C_0 + \bar{T} \underset{r_u^1}{\overset{r_b}{\rightleftharpoons}} C_1 + \bar{T} \underset{r_u^2}{\overset{rb}{\rightleftharpoons}} C_2, \qquad C_2 \underset{r_c^2}{\overset{r_o^2}{\rightleftharpoons}} O_2,$$

$$C_2 \underset{r_c^1}{\overset{r_o^1}{\rightleftharpoons}} O_1, \qquad C_2 \underset{r_r}{\overset{r_d}{\rightleftharpoons}} D$$

where the symbol $\bar{T}$ represents the concentration of the neurotransmitter molecules, which bind to the receptor-gated channels. $\bar{T}$ is modeled as a pulse of 1 μM neurotransmitters of duration 1 ms. Receptor-gated channels C_0 are activated through a process of two reversible $\bar{T}$ binding steps: from C_0 to C_1, intermediate channel state, and then to C_2, the activated channels. There are then two possibilities. In the first, the activated channels evolve to the states O_1 and O_2, in which the channel is open to the ion flux that transmits the signal. In the other case, the activated channels evolve to the desensitized states D, and are not permeable to the ion flux; also the fraction of channels that open during a synaptic response decreases. All transitions are reversible. The model developed in [25] contains also a description of the $\bar{T}$ release mechanisms, uses a concentration of 100 μM for Ca^{2+}, with some hypotheses on releasable vesicle pool dynamics. The simulation results showed that receptor desensitization contributed to synaptic depression. This model reproduces some features of synaptic depression dynamics, e.g. vesicle depletion. Nevertheless, the scale of description is indeed too high and considers the synapse as a whole, so it not permits a study of the vesicle pool dynamics at level of single vesicles.

4 A Process-Calculus Stochastic Model

We describe a model which is stochastic and based on the Pi-calculus, following the above mentioned interpretation of biological systems as computational ones. Several process calculi have been equipped with a stochastic semantics, e.g. [8, 38, 49, 50], often based on the stochastic simulation algorithm and its variants, originally proposed to determine one of the possible evolutions of a biochemical system according to a given probability distribution. The dialect of the Pi-calculus adopted here also has a stochastic semantics and suitable simulation and analysis tools, viz. SPiM [48].

Our approach benefits from conjugating an abstract and compositional algebraic model with a formal stochastic semantics, accounting for the non-continuous, nor discrete, nature of many described phenomena, as discussed in the previous sections.

Next, we report on stochastic models of the pre- and post-synaptic terminals, obtained by building upon deterministic models, and we illustrate how the model specification has been turned into an executable representation. First, we introduce a core model of the pre-synaptic activity: from Ca^{2+} ions to released neurotransmitters. The implementation of the model is also discussed. Next, we present an experiment about pre-synaptic short-term plastic phenomena. This is based on incremental enhancements of the core model, which modularly increase its descriptive power till covering a quite articulate set of processes. Further on, we describe our model of membrane activity in the post-synaptic terminal. In the present case, a model of the complete synapse signal traversal can be compositionally obtained by the separate models of the two terminals. On top of this, another experiment about plastic events throughout the whole synapse has been carried out.

These experiments represent an initial validation of the adequacy of the approach for the long term study of more detailed plastic mechanisms, which are at the basis of memory and learning and are involved in several neural diseases.

4.1 Step- and Wave-Like Ca^{2+} *Uncaging Pre-synaptic Model*

Our work starts from the deterministic model of vesicle release in the pre-synaptic terminal presented in [57]. Moving to a stochastic and discrete model, we calculated stochastic rates according to the known relationship between stochastic and deterministic rate constants [34]. This requires to estimating the volume within which the reactions of interest occur. The particular morpho-functional organization of this synapse has allowed us to model a subunit of the pre-synaptic element, consisting of a cluster of about 10 active zones, each containing about 10 vesicles, in a volume of 0.5×10^{-15} liter (see Sect. 2.2). With this volume estimate, both the stochastic rates and the initial quantities of Ca^{2+} ions can be defined (e.g. 6000 Ca^{2+} ions correspond to a molar concentrations $[Ca^{2+}]$ of 20 μM).

Our simulations have confirmed the results obtained with the deterministic model and they agree with experimental findings, thus supporting the adequacy of our stochastic model. In particular, we verified high sensitivity of vesicles to calcium concentrations. In several other synapses, vesicle release requires a calcium concentration in the range of 100–300 μM [65], while in the *calyx of Held* [57], the concentration can be much lower than 100 μM, as confirmed by our results, according to which relatively low concentrations up to 20 μM are able to deplete the releasable pool in a few milliseconds.

Figure 1 shows the simulation results for the step-like calcium un-caging. The left side shows mainly the step-like time course of Ca^{2+}; the middle part displays the same picture in logarithmic scale so as to appreciate the intermediate states of calcium binding and vesicle activation (`Vstar`) and release (`T`); the right part focuses on activated vesicle (`Vstar`) and the total number of the transmitter released (`T`). It

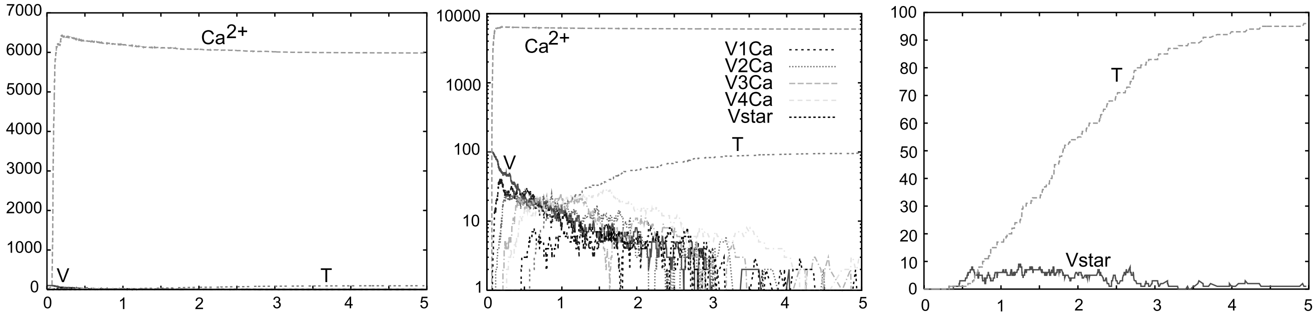


Fig. 1 Step-like calcium un-caging ($V = 100$; $\mathrm{Ca} = 6000$; $C_{\mathrm{on}} = 0.3$; $C_{\mathrm{off}} = 9500$; $\gamma = 6000$; $b = 0.25$)

can be observed that the pool of vesicles is 80% depleted within 3 ms, in accordance with the experimental findings [57].

High sensitivity of vesicle release has been observed in response to a uniform elevation of [Ca^{2+}] in the range 10 μM. Recently, the experimental work [4] has addressed the issue whether very short [Ca^{2+}] elevations are sufficient to induce a release similar to that due to an action potential. A spatially uniform and very rapidly decaying [Ca^{2+}] transient was induced by Ca^{2+} un-caging in the presence of added Ca^{2+} buffers. This short-lived elevation (wave-like) of calcium concentration has been revealed to be able to trigger vesicle release. It has been straightforward to introduce in our model a simple mechanism of calcium extrusion, adapted from [17] by tuning the rate constants to fulfill our working hypotheses:

$$\mathrm{Ca}_i^{2+} + P \underset{c_2}{\overset{c_1}{\rightleftharpoons}} \mathrm{Ca}P \xrightarrow{c_3} \mathrm{Ca}_o^{2+}$$

where Ca_o^{2+} is the extruded calcium and P is an abstraction of a pumping mechanism. The left side of Fig. 2 displays the simulated calcium wave lasting about 1 ms, with a half width 0.5 ms and a peak value of about 6,000 ions (corresponding to a calcium concentration of 20 μM), conforming to the experimental requirements in [4]. The right side of the same figure shows the release of one vesicle. Considering that a whole pre-synaptic element can be made of about 70 of simulated clusters, it results that a single action potential, and accordingly a single calcium wave, is able to release a significant amount of vesicles. This also fits with the experimental findings [4, 57, 58]. Notably, the Ca^{2+} time courses in both step and wave like cases were obtained in a very simple way (Sect. 4.2) when compared to the methods involving the solution of complex systems of differential equations, as done in [57, 58].

4.2 Implementation of the Model

Excerpts from the model, viz. its SPiM implementation, are here discussed in order to give a flavor of the process-based model adopted. As explained (bio-chemical) interaction is modeled as process communication, here pairs of complementary input/output actions over the same communication channel (`?c`/`!c`). In the present model, communication reverts to synchronization, as no data is exchanged. Communication channels can be (dynamically) created by the `new` command and have associated a stochastic rate, e.g. several rates are defined and channel `vca` is created with rate `con5 = 1.5`:

```
val con5 = 1.5
val b = 0.25
val coff5 = 47500.0 * b * b * b * b

new vca@con5:chan
```

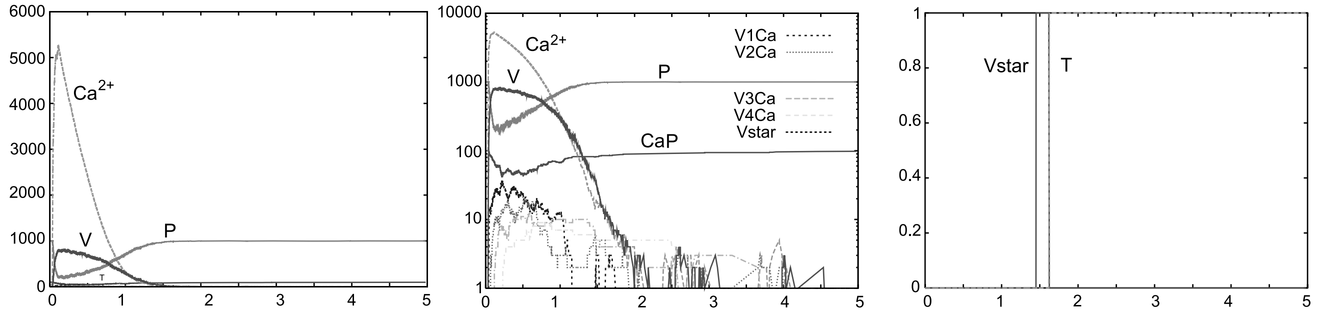


Fig. 2 Wave-like calcium un-caging ($V = 100$; $\mathrm{Ca} = 6000$; $C_{\mathrm{on}} = 0.3$; $C_{\mathrm{off}} = 9500$; $\gamma = 6000$; $b = 0.25$)

Then the specification of some interactions follows. A calcium ion (ca()) can interact with a vesicle (v()) over channel vca (beyond being able to do other things). After this communication, ca() disappears and v() becomes v_ca(), representing the binding of the two. This realizes a second order reaction.

First order reactions are modeled as interactions with a *single* dummy molecule, i.e. a fictitious molecule introduced to allow synchronization. Being in a single copy, it does not alter the stochastic dynamics since such a quantity is irrelevant. For instance, any v_ca() can then either accept other calcium bindings or degrade back to an unbound vesicle by communicating through bvca with the single copy of Dv_ca(). After such a reaction, v(), ca() and Dv_ca() itself are restored.

```
ca() =  do ?vca;()   v() =   !vca;  v_ca()
        or ?v2ca;()  v_ca() = do !bvca; v()
        ...                   or !v2ca; v_2ca()
        or ?cp;()    Dv_ca() = ?bvca; ( ca() | Dv_ca() )
```

Some of the operators used are sequence (;), multiple alternative choice (do ?c or ?d or ...), i.e. only one of the possible actions can be executed and the others discharged, and process definition (p()= ...) and invocation (...; p()), a mechanism that allows a behavior to be specified (definition) and dynamically enabled by other processes (invocation). Processes can run in parallel (p()|q()).

The dynamics of a simulation consists of a trajectory, i.e. a set of transitions from one state to the next one, determined from the actions enabled in a state and the number of processes ready to perform them, according to SSA.

The duration of the simulated dynamics of the system (here 0.005 s) is set by an initial command like:

```
    directive sample 0.005
```

So far, the system has been described by specifying simple atomic behavior, basically corresponding to chemical reactions, and then by composing them together. The parametric process w(cnt:int) allows us to suitably modulate the calcium wave. After a stochastic delay, if its integer parameter cnt is positive it replicates 80 copies of itself, with the parameter decreased, in parallel with 80 ca(). Otherwise, if cnt is not positive, it dies. This realizes a recursively defined exponential growth, which can be controlled by the delay rate and the parameter in its rapidity and quantity. The growth is turned into a wave by specifying the pumping mechanism as described in the model. Accordingly, the interaction capability, i.e. action ?cp, has to be added to ca(). This is a modular composition that only requires further to tune pump parameters to obtain the desired wave.

Finally, the initial state of the experiment can be populated specifying how many molecules of each specie are present (here 1 wave, 1,000 pumping molecules, 100 vesicles and 1 copy of the needed dummy molecules).

```
p() =    !cp; ca_p()

w(cnt:int) = do delay@40000.0;
                if 0 <= cnt then ( 80 of ca() |  80
                   of w(cnt - 1)) else ()
             or !void; ()

run   1 of w(1)   1000 of p()   100 of v()
      1 of (Dv_ca() | Dv_2ca()| ... )
```

4.3 Short Term Plasticity: Facilitation, Potentiation, and Depression in the Pre-synaptic Terminal

Here, we recast some experiments regarding plasticity phenomena in the presynaptic terminal, previously introduced in Sects. 3.2, 3.3, and 3.4 in the deterministic context. Beyond relying on a more adequate model, as discussed above, these examples also illustrate the expressiveness of the linguistic abstraction adopted.

Paired Pulse Facilitation

We consider the hypothesis on residual Ca^{2+} which actually is less than 1 μM, i.e. less than 300 Ca^{2+} ions for the volume we consider. For most synapses, the increase of local [Ca^{2+}] needed for release is between 100 and 300 μM. The residual Ca^{2+} is therefore not typically sufficient to induce facilitation.

Figure 3 shows two Ca^{2+} waves with varying time delays between them (12 and 2 ms). For the smaller delay, the amount of release caused by the second wave increases notably. From the central column, a possible explanation emerges: just before the second calcium wave develops, the amount of residual Ca^{2+} grows when the delay decreases. Also, other parts of the release machinery seems to be influenced by the residual Ca^{2+}, even at low concentrations. This is the case, e.g. for the occupancy of the pump P (central column) or for the intermediate steps of vesicle binding.

Summing up, our simulations support the hypothesis that, in the *calyx of Held*, paired pulse facilitation is likely due to the residual Ca^{2+} and to occupancy of Ca^{2+} buffers [20, 75].

Implementation

Once a wave generator has been defined, multiple instances of it can be activated with different stochastic delays, e.g.

```
snd_w() = delay@0.125; w(1)
```

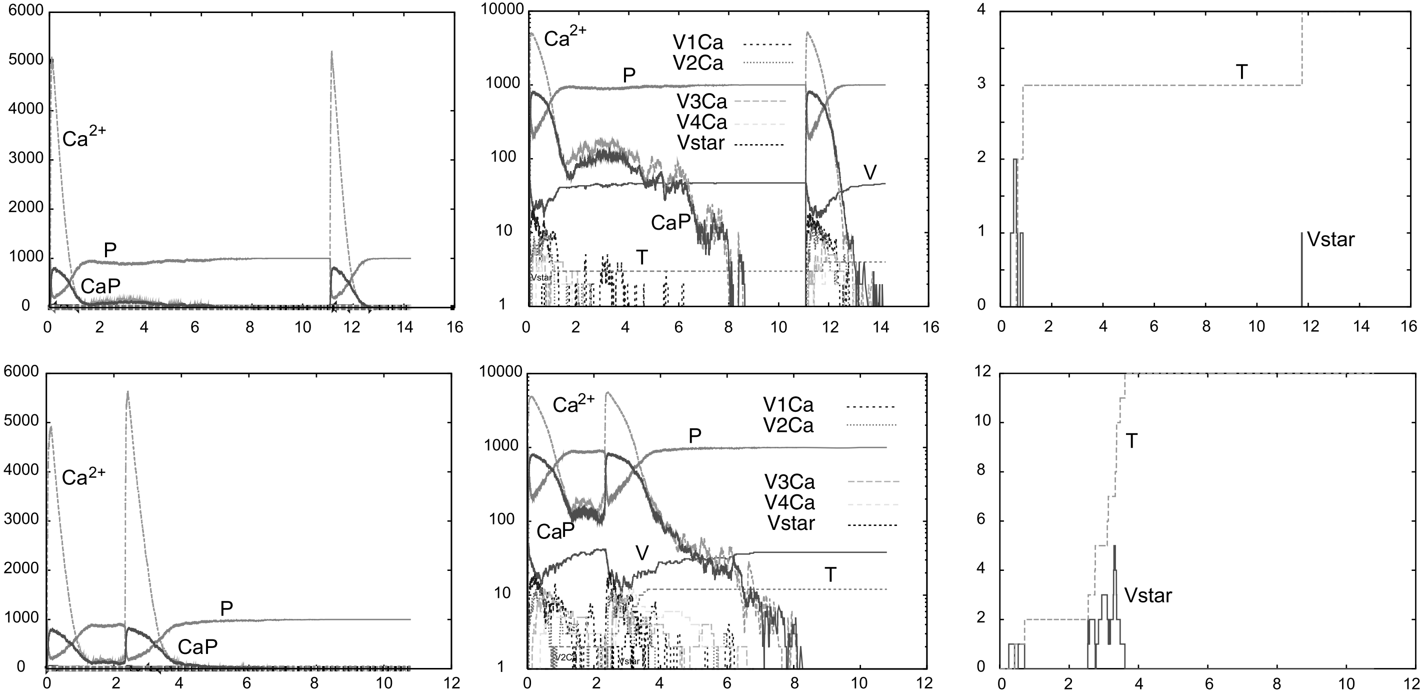


Fig. 3 Two wave-like calcium pulses at different intervals ($V = 50$; $C_{\mathrm{on}} = 0.4$)

It is worth noting that the actual delay after which the wave is generated cannot be exactly determined, because it is stochastically based on a probability distribution. Determining events at a specific time point is currently not supported by the simulation tool. As a matter of fact, this would be beyond the dynamics ruled by SSA. Also, the events we are considering rarely occur in nature at a time point that can be exactly determined.

Synaptic Potentiation

We derived a stochastic model from the model proposed in [39] that describes the synaptic potentiation of the spontaneous and evoked vesicle release induced by the phorbol esters. In this case, the effect of the esters, which increase the apparent Ca^{2+} sensitivity of vesicle fusion, can be straightforwardly studied by varying the fusion rate, which originally was $l_+ = 0.0002$ and has been increased fivefold.

Results for a step-like Ca^{2+} un-caging of 1,200 ions are in Fig. 4. In accordance with experimental findings [39], potentiation effects are more visible at smaller Ca^{2+} values. Values for the fusion willingness parameter l_+ are 0.0002 (left column) and 0.001 (middle column). A logarithmic scale has been here used to plot the calcium un-caging. The right column shows a 20 trials for the different values of l_+. In the case of increased l_+, the number of released vesicles increases, while at the same time, the number of the activated vesicles decreases.

Implementation

In this case, only small changes to parameters have been needed, so as to reflect the new hypotheses over vesicle behavior.

Synaptic Depression

In order to test competing hypotheses on the mechanisms of short-term synaptic depression, we have defined in [7] a stochastic model addressing the spatial and functional distribution of vesicles within the pre-synaptic terminal, their different kinetics, their different recruitment, and the speed-up role of intracellular Ca^{2+} for recruitment in case of intense neural activity (up to ten times faster than normal [31]). Importantly, the model exhibits dynamical equilibrium: after depletion, the initial number of vesicle is replaced. Moreover, this makes it interesting to simulate longer time scales, not limited to the “release interval” only. The implementation of these features has been facilitated by the possibility of mixing different levels of abstraction in a quite compositional construction of the whole model.

As far as release dynamics is concerned, we considered a fast and a slow pool of vesicles, as suggested in [70]. Each of the model components is inspired by the model in [39] (Sect. 3.3), with suitable values of l_+ distinguishing the two.

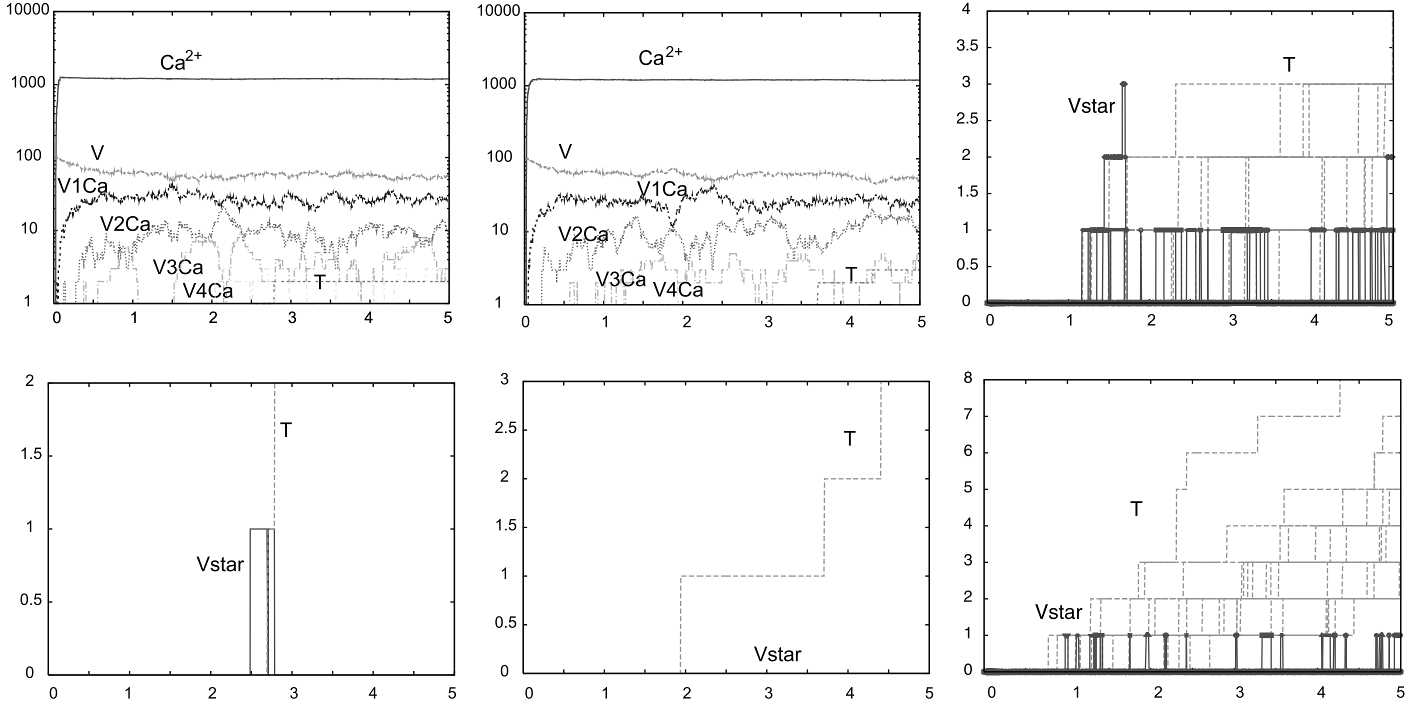


Fig. 4 Synaptic potentiation. Step-like Ca^{2+} un-caging and multi-run wave-like Ca^{2+} un-caging for $l_+ = 0.0002$ and $l_+ = 0.001$ (willingness rate). $Ca^{2+} = 1200$

For the *calyx of Held*, it has been shown that the maximum number of releasable vesicles does not vary in time. Hence, we have fixed a maximum (and starting) number of vesicles and other species that can be present at the same time in the system. This is supported by a spatial interpretation that we adopted as a working hypothesis: hosting capability is constrained by the saturation of the space available to the vesicle pools. Hence, we label specie names with numbers. Indeed, two possible ways to obtain dynamic stability for vesicle trafficking are either limiting the maximum number of vesicles in each pool, or imposing some temporal dependence on the stochastic constants.

This model is graphically summarized in Fig. 5 and formally specified as (together with the chosen vesicle dynamics):

$$V_{RP}^{\infty} \underset{k_2}{\overset{k_1}{\rightleftharpoons}} V_{DSP}^{\mathbf{100}} \underset{k4}{\overset{k_3}{\rightleftharpoons}} V_S^{\mathbf{50}},$$

$$\mathrm{Ca}_i^{2+} + C \underset{k_7}{\overset{k_5}{\rightleftharpoons}} \mathrm{Ca}_i^{2+}C + V_{RP}^{\infty} \xrightarrow{k_6} V_F^{\mathbf{50}} + \mathrm{Ca}_i^{2+} + C.$$

As a matter of fact, recruitment is a vesicle maturation process. The recruitment is faster for the slow vesicles, while for the fast ones it may well go beyond the time interval considered. The slow vesicles go back and forth between the releasable ($V_S^{\mathbf{50}}$) and *docked* ($V_{DSP}^{\mathbf{100}}$) pool states. Docked vesicles are not yet releasable. They are replenished from a reservoir pool of vesicles (V_{RP}^{∞}), which is refilled with mechanisms on longer time scales, and hence not considered here. Here, the label ∞ represents a significantly large pool of elements. The recruitment of fast vesicles may be accelerated by the action of Ca^{2+} and the molecule *calmoduline* (C) [31, 59], which form a complex ($\mathrm{Ca}^{2+}\ C$) inducing vesicle maturation.

Figure 6 shows linear scale, log scale, and a selection of relevant simulation values, as usual. The first row illustrates the effects of a continuous depolarization, mimicked by a step-like Ca^{2+} (6000) un-caging experiment. The release T_F due to

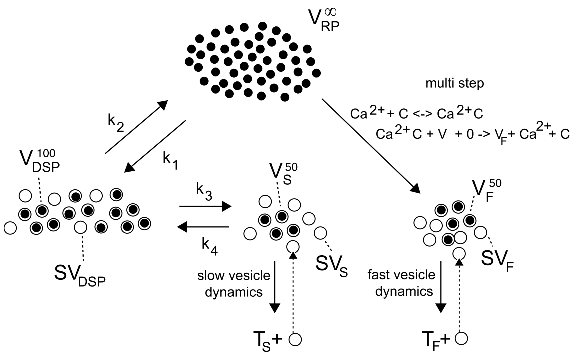


Fig. 5 Schema of the multi-pool vesicle refilling and depletion

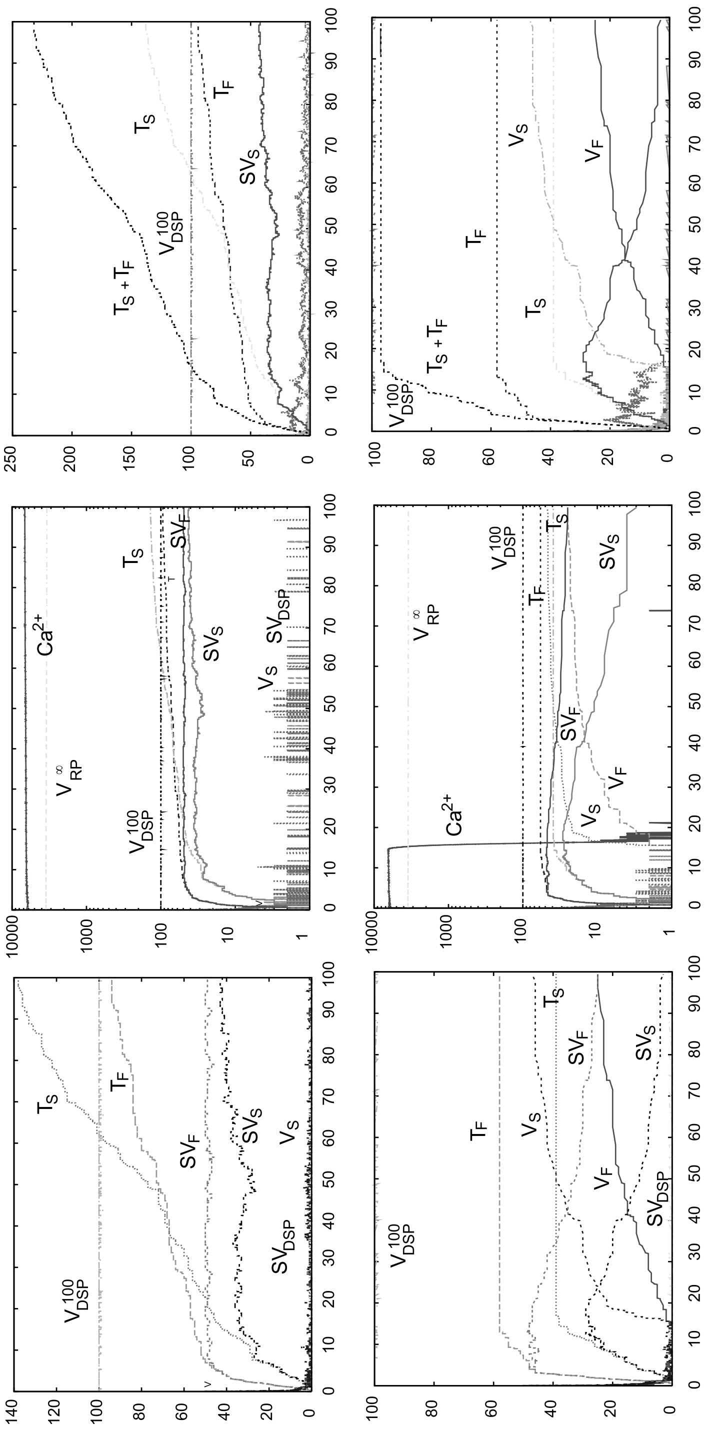


Fig. 6 Multi-pool vesicle populations time course during step and plateau like Ca^{2+} un-caging

the fast vesicles V_F (synchronous) happens within the first 3 ms, that of T_S due to the slow V_S (asynchronous) has a longer duration and the double slope of their cumulative release ($T_F + T_S$) is evident. Moreover, the variation in the space availability for the different pools (SV_F, SV_S) is shown. The bifurcation of the two curves T_F and SV_F is due to the contribution of the fast vesicles recruited during the activity: the former continues growing whereas the latter remains almost constant and near its maximum value of 50. The slow pool (V_S) is emptied more slowly (see the time course of SV_S) and the contribution to the release comes from the underlying refilling process (compare the behavior of SV_S and T_S).

The second row shows the effects of a plateau-like Ca^{2+} un-caging lasting about 17 ms. In a depolarization of assigned duration, it is possible to follow the depletion and the refilling of vesicle pools: after a release the system comes back to the initial conditions with time courses compatible to the experimental observations (actually, the time scale in Fig. 6 has been accelerated for the sake of presentation, while in reality the refilling lasts for hundreds of ms).

The introduced concept of available space represents a real and effective constraint, often overlooked in most of the models of synaptic function. Moreover, we have modeled fast vesicles recruitment through local Ca^{2+} by a few abstract intracellular process; other authors [31] instead needed to invoke global Ca^{2+} concentrations.

Implementation

The extended model above encompasses dynamics that go beyond the modeling of mere bio-chemical events. Issues with a spatial and temporal flavor have been addressed, yet in a simple manner.

Bounding the number of the elements of certain species can be thought as a *spatial* or structural property, like a saturation phenomenon of the available area. Labeled species, e.g. $V_S^{\mathbf{50}}$, have been implemented by assuming a set of active elements, containing as many elements as the number of elements allowed, e.g. 50 for V_S. The generation of a constrained element, e.g. $V_{DSP}^{\mathbf{100}}$ or $V_S^{\mathbf{50}}$, happens through interaction with one of the active elements of the relative set, if any available exists, and binds it, making it unavailable for further interactions.

```
v_rp() =                vd_docked() =
   !d_ch1; vd_docked()     do !d_ch2; ( v_rp() |
                               vd_docking_space() )
                           or !d_ch3; ( sv()    |
                               vd_docking_space() )
vd_docking_space() =
   do ?d_ch1; ()        D_vd_undocking() = ?d_ch2;
                        D_vd_undocking()
   or ?d_ch4; ()
                        sv() =
vs_docking_space() =       do !d_ch4;  ( vd_docked() |
                           vs_docking_space() )
   ?d_ch3; ()              ...
```

A vesicle of the reservoir pool `r_vp()` can become docked `vd_docked()` only by interacting with `vd_docking_space()`, the relative “spatial” element, through channel `d_ch1`. The spatial element is then lost and it is restored when the vesicle is released. Hence, no more specie elements can be simultaneously present than those allowed by the constraint. A spatial element is also needed for a slow vesicle `sv()` to go back to the docked position via `d_ch4`, restoring one `vs_docking_space()`, while a docked vesicle can freely go back to the unconstrained `v_rp()` (`d_ch2`), also restoring one docking slot; analogously for a docking vesicle maturing to a slow one (`d_ch3`).

The Ca^{2+} mediated facilitation determines *temporal* properties of the recruiting of fast vesicles. In the implementation of this process, first `ca()` activates `clmd()` into `ca_clmd()`, then either the reaction is inverted or a fast vesicle is maturated via a constrained mechanism (`d_ch6`). Both `ca()` and `clmd()` are released:

```
clmd() =                       ca_clmd() =
   !d_ch5;ca_clmd()               do !d_ch6; ( v() | ca() | clmd() )
                                  or !d_ch7; ( ca() | clmd() )
 vf_docking_space() =
   ?d_ch6; ()
```

It is important to note that rather than a precise bio-chemical interaction of `ca()`, we are here describing an abstract, perhaps less understood, facilitator role of it, which leaves `ca()` and `clmd()` unaltered. This kind of abstract processes may be useful for summarizing a set of bio-chemical reactions, of which only the resulting macro behavior is known. These processes can be easily modeled within the language, but require a proper tuning of the stochastic parameters for matching the expected behavior, in this case the known time scale of vesicle recruitment. The varying amount of `ca()` determines the strength of the reaction, according to SSA, implementing a sort of variable stochastic rates.

Note also that in the second step, i.e. the maturation of a fast vesicle via `d_ch6`, no vesicle from the reservoir pool is involved, since both `ca_clmd` and the constraining mechanism already participate to the binary interaction. This simplification is acceptable, since vesicles are continuously made available by the pool and the tuning of parameters recovers again the time course of this abstract process. In general, whenever one might want to model abstract processes involving more than two components, binary interaction alone can result in a limitation.

4.4 Post-synaptic Membrane Receptor Activity

We have modeled the membrane activity in the post-synaptic terminal, related to the pre-synaptic mechanisms previously described. This has been done in [6] by devising a stochastic model for the post-synaptic terminal in isolation from the model in [16, 25].

The equations are those in Sect. 3.5 read stochastically. We recall that $\bar{T}$ stands for each single neurotransmitter molecule that binds to the receptor-gated channels, while in the pre-synaptic model T has been used to represent the whole content released by a single vesicle.

The synaptic response is determined, on the post-synaptic side, by the time course of the rising phase of the synaptic ion current. This is a function of the opening rate of the receptor-gated channels and of the neurotransmitter concentration. We monitor the numbers of open channels O_1, O_2, and de-sensitized channels D.

We have determined the number of the $\bar{T}$, which are stochastically assumed to be released in the synaptic cleft, in accordance with the amount stochastically produced by a single calcium wave in our experiments on the pre-synaptic terminal (about 3 to 6 vesicles). Each "pre-synaptic" T corresponds to a wave of neurotransmitters $\bar{T}$. The definition of the wave duration and amplitude required some considerations. First, to determine the "activation" rate of the receptor r_b, we not only need to estimate the *volume Vol* occupied by the neurotransmitter in the cleft, but also the effective *interaction volume I Vol* where the interactions happen against the post-synaptic *surface*. We have computed *Vol* by considering how many neurotransmitters are contained into a single vesicle (6000–8000) and by considering the value of their concentration when they have spread into the synaptic cleft: 10^{-3} M. We obtained Vol $= 10^{-17}$ liter. Following [55], we estimated the effective IVol as Vol/160, obtaining IVol $= 6 \times 10^{-20}$ liter, which was needed for converting from the deterministic to the stochastic rate value. Also, IVol permits us to estimate the number of neurotransmitters involved: about 50 $\bar{T}$, as peak value.

Figure 7 shows a sample of the simulations performed: the time course of the neurotransmitter $\bar{T}$, and the channel states O_1, O_2, D, together with other intermediate channel states; the logarithmic scale of them; and a focus on the time course of O_1, O_2, D. Note the very short duration of the $\bar{T}$ waves and the relatively slower kinetics of the post-synaptic activated channels. This corresponds to the prolonged time course of the synaptic currents observed experimentally [12, 16]. Moreover, observe that for each $\bar{T}$ wave, i.e. for each vesicle released according to our assumptions, the number of open channels is of the order of tens, fitting with experimental observations [75]. Finally, the buildup of the desensitization is also evident, again in accordance with experimental data.

4.5 A Whole Synapse Model

A comprehensive model of the whole synapse has been devised by building upon the independently developed models of the pre- and post-synaptic processes, which have been considered in isolation in Sects. 4.1 and 4.4. Details about the overall model can be found in [6]. What is worth underlining here is that the construction is done in a strongly compositional manner. This has been facilitated by the "spatial" separation of the two component models and by the presence of a well-defined "point of contact", the neurotransmitters, which is quite a natural interface between the two models.

Indeed, the calcium pulse induces the release of vesicles, represented abstractly in the pre-synaptic model as the release of a *single* neurotransmitter T by each

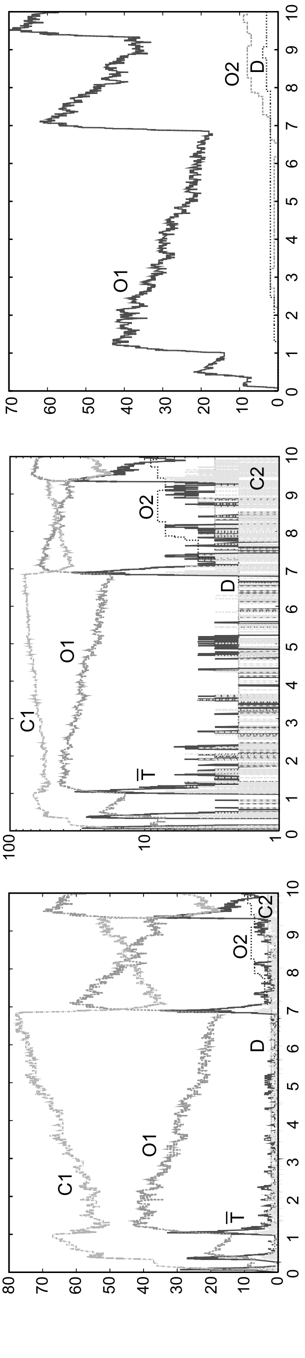


Fig. 7 Post-synaptic channel activation

vesicle. In the post-synaptic model, each T has been interpreted as a wave of neurotransmitters, with amplitude and duration determined as above, as sketched by the following equations, where $\equiv$ puts into relations the release of a vesicle T and the generation of a neurotransmitter wave $\bar{T}$.

$$\ldots \xrightarrow{\gamma} T, \qquad T \equiv \bar{T}, \qquad C_0 + \bar{T} \xrightleftharpoons[r_u^1]{r_b} C_1 \ldots .$$

Often, in the study of synaptic transmission many of the elements we considered are often neglected, e.g. typically the stochasticity of the vesicle release and the real amount of neurotransmitter sensed by the post-synaptic receptors. It is also worth noting that the calcium wave now induces stochastically the $\bar{T}$ waves we supplied by hand when describing the membrane receptor dynamic in isolation.

Implementation

The needed interface can be simply implemented by re-defining vesicle behavior as

```
    vstar() = .... ; tt(1)
```

and defining `tt( n:int )` as a suitable `t()` wave generator, as usual. Then `t()` has the same definition given to each neurotransmitter in the post-synaptic model.

4.6 An Experiment on the Whole Synapse

Figure 8 shows some results of a virtual experiment addressing plasticity events in the whole synapse. Indeed, the interaction between pre- and post-synaptic mechanisms generates short-term plasticity events in a way not yet fully understood [73]. Besides describing some events of short-term plasticity, our model helps distinguishing the pre-synaptic and post-synaptic influence in them. It is know that in the process of short-term synaptic depression, vesicle depletion and post-synaptic receptor desensitization play distinct roles. It has also been shown that desensitization has a role in the synaptic depression only when the frequency of the action potentials is above 10 Hz [73].

The displayed results are on a train of calcium waves at a frequency of about 100 Hz, which mimics action potentials like neural signals. The left part of the figure displays calcium waves, and other values like pump occupancy and channel intermediate states, shown in the central part in logarithmic scale. This makes clear the dynamics of the many processes involved, e.g. the neurotransmitter dynamics and the number of open channels. Remarkably, in the right part, a buildup of desensitization is visible starting from the third wave (see the increase of value D in

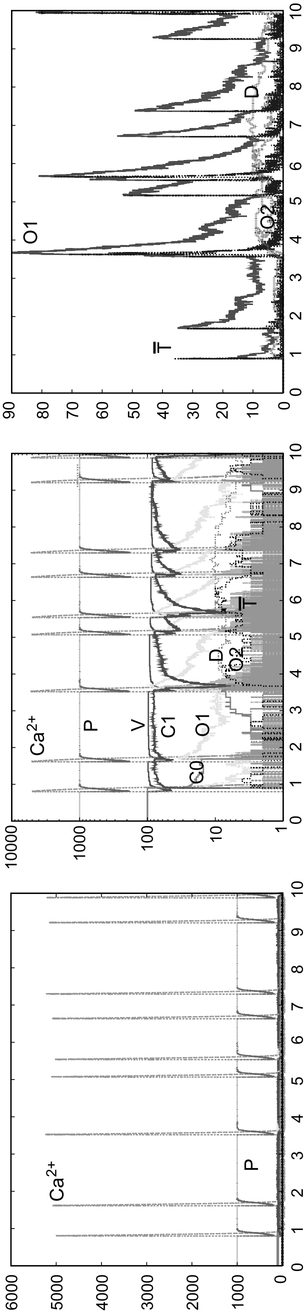


Fig. 8 Whole synapse plasticity: 100 Hz pulse depression

between the third and fourth wave). Note, also, that the same amount of neurotransmitter causes different effects as far as the number of opened channels is concerned (compare the ratio between $O1$ and $\bar{T}$ in the first two waves with the same ratio in the sixth and the seventh wave). One feature of our approach is the possibility of quantifying the number of vesicles stochastically associated to each action potential (recall that each vesicle corresponds to a $\bar{T}$ wave with a peak of about 50 neurotransmitters and it is possible to see superpositions of some of them). Another important aspect is that at the same time one can measure the number of receptor channels open or desensitized. This means that it is possible to distinguish effects due to vesicle depletion, when it builds up, from the effects due to receptor channels.

5 Concluding Remarks

In this paper, we have surveyed a selection of some recent formal models developed for elucidating transmission and plasticity processes at the *calyx of Held* synapse. Moreover, we have provided a brief overview of the biological and formal backgrounds of the topic.

We have focused first on deterministic models in Sect. 3 describing the Ca^{2+} triggered release process and related plastic events, for which elements of vesicles trafficking and post-synaptic membrane have been introduced. All these models fit the experimental findings fairly well, for averaged variable quantities concerning the synapse as a whole. Nevertheless, these approaches have some methodological limitations, due to the break down of the continuous hypothesis, when the processes investigated need a focus on small volumes and concentrations, such as plasticity. This is particularly clear for this synapse in which, for example, small values of Ca^{2+} concentrations play a relevant role. Indeed, absolutely small quantities at the volume of interaction, which are much smaller than in the majority of the other synapses, are able to affect the release processes, as it happens in the facilitation processes. Moreover, these approaches present some computational limitations because they rely on complex sets of ODEs for describing relatively simple Ca^{2+} time courses.

We have then described in Sect. 4 models of our own [5–7], consisting of stochastic representations and extensions of the models presented in Sect. 3. By incrementally building upon the core models of the synaptic terminals, we have covered aspects as step and wave-like Ca^{2+} un-caging, facilitation due to repeated activity, spontaneous release, potentiation due to activation parameters strengthening. We have also presented a quite precise, abstract, and new representation of vesicle depletion and refilling under prolonged depolarization, which is able to exhibit dynamic stability. By the compositional property of our approach, we have connected the initial models of the pre- and post-synaptic terminals in a model of signal traversal of the whole synapse, on which we have carried out experiments about plasticity of the synapse. The linguistic abstraction chosen has allowed us to model processes at different levels of abstraction in a uniform way, such as biochemical dynamics and more abstract mechanisms, used to represent processes whose full details are still not understood.

The main ingredients of our approach are: a *stochastic* model, which is more suitable than other approaches traditionally adopted in the context addressed; a *process calculus* as a representation language, which has demonstrated good properties in terms of expressiveness, modularity, compositionality, and adequate levels of abstraction; a *computational tool*, joining together the above two ingredients, to perform efficient and precise *in silico* simulations.

From the *biological* viewpoint, the surveyed models have been assembled out of experimental data, by fitting of parameters and hypotheses on non-fully understood mechanisms. They provide a coherent representation of the reality they represent. Examples from the stochastic approach are the sensitivity to Ca^{2+} in the release process and in the facilitation phenomena, and the adequacy of the spatial and kinetic hypotheses on vesicle dynamics (e.g. time courses of depletion and refilling under prolonged de-polarizations, see [45]). The surveyed models have to be considered as virtual environments where, in perspective, one may precisely verify and build hypotheses and explanations by *in silico* experiments. The more complete the model, the more informative the virtual experiments.

The presented multi-disciplinary study has also given us some insights from the *computer science* viewpoint. The adequacy of process calculi for modeling life systems has been further tested. These experimentations also suggest possible future development, such as a possible role for non-binary synchronization in supporting processes beyond the mere biochemical dynamics; a finer control of event time; and the embedding of quantities, like simulation time or the number of elements of a given specie, as data in the calculus. Also, of interest, is the possibility of using variable stochastic rates for linking, e.g., reaction speed to temperature. Many of these extensions rise the issue of their integration within the stochastic semantics as determined by SSA. For instance, a notion of locality could be used to overcome the assumption of spatial uniformity (well-stirred space). Indeed, for synaptic transmission and plasticity it is recognized that the introduction of a description of spatial structures could improve the quality of experiments [3, 22, 30, 74, 76], e.g. for handling diffusion processes. In this sense, "location-aware" calculi, like the Brane calculi [9] or spatial Pi-calculus [41], seem of interest. Finally, compositionality also needs further investigations: the stochastic interplay of different processes makes compositionality hard in the general case, since a local behavior in a component can perturb the overall system to which the component is added. Compositionality is instead easier to achieve when components are quite separate, for instance when they refer to disjoint volumes or have a clear and limited interface through which their molecules can interact, as it happens in the composition of the pre- and post-synaptic models [6].

Acknowledgements This research has been partially supported by MIUR PRIN *Bisca*, and by EU-FETPI Global Computing Project IST-2005-16004 *Sensoria* (*Software Engineering for Service-Oriented Overlay Computers*).

References

1. Arbib MA (ed) (1995) The handbook of brain theory and neural networks. MIT Press, Cambridge
2. Ascoli GA (ed) (2002) Computational neuroanatomy. Humana, Totowa
3. Atwood HL, Karunanithi S (2002) Diversification of synaptic strength: presynaptic elements. Nat Rev 3:497–516
4. Bollmann JH, Sakmann B (2005) Control of synaptic strength and timing by the release-site Ca^{2+} signal. Nat Neurosci 8:426–434
5. Bracciali A, Brunelli M, Cataldo E, Degano P (2007) Expressive models for synaptic plasticity. In: Calder M, Gilmore P (eds) Computational methods in system biology (CMSB'07). LNBI, Springer, Edinburgh, pp 152–167
6. Bracciali A, Brunelli M, Cataldo E, Degano P (2008) Stochastic models for the *in silico* simulation of synaptic processes. BMC Bioinform 9(4):S7
7. Bracciali A, Brunelli M, Cataldo E, Degano P (2008) Synapses as stochastic concurrent systems. Theor Comput Sci 408(1):66–82
8. Calder M, Gilmore S, Hillston J (2006) Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra pepa. Trans Comput Syst Biol VII 4230:1–23
9. Cardelli L (2004) Brane calculi-interactions of biological membranes. In: Vincent V, Schachter V (eds) Proceedings of computational methods in systems biology. LNCS, vol 3082. Springer, Paris, pp 257–280
10. Cardelli L (2008) On process rate semantics. Theor Comput Sci 391(3):190–215
11. Cassman M, Arkin A, Doyle F, Katagiri F, Lauffenburger D, Stokes C (2007) System biology—international research and development. Springer, The Netherlands
12. Clements JD (1996) Transmitter timecourse in the synaptic cleft: its role in central synaptic function. Trends Neurosci 19:163–171
13. Cooper LN, Intrator N, Blais BS, Shouval HZ (2004) Theory of cortical platicity. World Scientific, New Jersey
14. Dayan P, Abbot LF (2001) Theoretical neuroscience—computational and mathematical modeling of neural systems. MIT Press, Cambridge
15. De Schutter E (2000) Computational neuroscience: more math is needed to understand the human brain. In: Engquist B, Schmid W (eds) Mathematics unlimited—2001 and beyond, 1st edn. Springer, Berlin, pp 381–391
16. Destexhe A, Mainen ZF, Sejnowski TJ (1998) Kinetic models of synaptic transmission. In: Koch C, Segev I (eds) Methods in neuronal modeling. MIT Press, Cambridge, pp 1–25
17. Destexhe A, Mainen ZF, Sejnowski TJ (1994) Synthesis of models for excitable membrane, synaptic transmission and neuromodulation using a common kinetic formulation. J Comput Neurosci 1:195–231
18. Ermentrout B (2002) Simulating, analyzing, and animating dynamical systems. SIAM, Philadelphia
19. Fall CP, Marland ES, Wagner JM, Tyson JJ (eds) (2005) Computational cell biology. Springer, New York
20. Felmy F, Neher E, Schneggenburger R (2003) Probing the intracellular calcium sensitivity of transmitter release during synaptic facilitation. Neuron 37:801–811
21. Gardiner CW (2001) Handbook of stochastic methods—for physics, chemistry and the natural science. Springer, Berlin
22. Ghijsen WEJM, Leenders AGM (2005) Differential signaling in presynaptic neurotransmitter release. Cell Mol Life Sci 62:937–954
23. Gillespie DT (1977) Exact stochastic simulation of coupled chemical reactions. J Phys Chem 81:2340–2361
24. Gillespie DT, Petzold LR (2006) Numerical simulation for biochemical kinetics. In: Szallasi Z, Stelling J, Perival V (eds) System modeling in cellular biology, 1st edn. MIT Press, Cambridge, pp 331–354

25. Graham BP, Wong AYC, Forsythe ID (2001) A computational model of synaptic transmission at the calyx of Held. Neurocomputing 38(40):37–42
26. Graham BP, Wong AYC, Forsythe ID (2004) A multi-component model of depression at the calyx of Held. Neurocomputing 58(60):449–554
27. Hennig MH, Postlethwaite M, Forsythe ID, Graham BP (2007) A biophysical model of short-term plasticity at the calyx of Held. Neurocomputing 70:1626–1629
28. Hertz J, Krogh A, Palmer RG (1991) Introduction to the theory of neural computation. Westview, Santa Fe
29. Hillston J (1996) A compositional approach to performance modeling. Cambridge University Press, Cambridge
30. Holmes WR (2005) Calcium signaling in dendritic spines. In: Reeke GN, Poznanski RR, Lindsay KA, Rosenberg JR, Sporns O (eds) Modeling in the neuroscience—from biological systems to neuromimetic robotics, 2nd edn. CRC Press, New York, pp 25–60
31. Hosoi N, Sakaba T, Neher E (2007) Quantitative analysis of calcium-dependent vesicle recruitment and its functional role at the calyx of Held synapse. J Neurosci 27:14286–14298
32. Izhikevich EM (2006) Dynamical system in neuroscience: the geometry of excitability and bursting. MIT Press, Cambridge
33. Van Kampen NG (1992) Stochastic processes in physics and in chemistry. Elsevier, Amsterdam
34. Kierzek AM (2002) Stocks: stochastic kinetic simulations of biochemical system with gillespie algorithm. Bioinformatics 18:470–481
35. Kitano H (2002) Systems biology: a brief overview. Theor Comput Sci 295(5560):1662–1664
36. Koch C (1999) Biophysics of computation—information processing in single neuron. Oxford University Press, New York
37. Koch C, Segev I (eds) (1998) Methods in neuronal modeling. MIT Press, Cambridge
38. Lecca P, Priami C, Quaglia P, Rossi B, Laudanna C, Costantin G (2004) A stochastic process algebra approach to simulation of autoreactive lymphocyte recruitment. SIMULATION: Trans Soc Model Simul Int 80(4):273–288
39. Lou X, Scheuss V, Schneggenburger R (2005) Allosteric modulation of the presynaptic Ca^{2+} sensor for vesicle fusion. Nature 435:497–501
40. Lytton WW (2002) From computer to brain—foundations of computational neuroscience. Springer, New York
41. Mathias J, Ewald R, Uhrmacher AM (2008) A spatial extension to the π calculus. Electron Notes Theor Comput Sci 194(3):133–148
42. Milner R (1999) Communicating and mobile systems: the π-calculus. Cambridge University Press, Cambridge
43. Mitra PP, Bokil H (2008) Observed brain dynamics. Oxford University Press, Oxford
44. Nagasaki M, Onami S, Miyano S, Kitano H (1999) Bio-calculus: its concept and molecular interaction. Genome Inform 10:133–143
45. Neher E (2006) A comparison between exocytic control mechanisms in adrenal chromaffin cells and a glutamatergic synapse. Eur J Physiol 453:261–268
46. Neher E (2007) Short-term plasticity turns plastic. Focus on synaptic transmission at the calyx of Held under in vivo-like activity levels. J Neurophysiol 98:577–578
47. Nicholls JG, Martin AR, Wallace BG, Fuchs PA (2001) From neuron to brain. Sinauer Associates, Sunderland
48. Phillips A, Cardelli L (2007) Efficient, correct simulation of biological processes in the stochastic pi-calculus. In: Calder M, Gilmore S (eds) Proceedings of computational methods in systems biology. LNCS, vol 4695. Springer, Edinburgh, pp 184–199
49. Priami C (1995) Stochastic π-calculus. Comput J 36(6):578–589
50. Priami C, Regev A, Shapiro E, Silvermann W (2004) Application of a stochastic name-passing calculus to representation and simulation of molecular processes. Theor Comput Sci 325(1):141–167
51. Regev A, Panina E, Silverman W, Cardelli L, Shapiro E (2004) Bioambients: an abstraction for biological compartements. Theor Comput Sci 325(1):141–167

52. Regev A, Shapiro E (2002) Cellular abstractions: cells as computation. Nature 419:343
53. Rieke F, Warland D, von Steveninck RR, Bialek W (1997) Spikes—exploring the neural codes. MIT Press, Cambridge
54. Sakaba T, Stein A, Jahn R, Neher E (2005) Distinct kinetic changes in neurotransmitter release after snare protein cleavage. Science 309:491–494
55. Savtchenko LP, Rusakov DA (2007) The optimal height of the synaptic cleft. Proc Natl Acad Sci 104:1823–1828
56. Schneggenburger R, Forsythe ID (2006) The calxy of Held. Cell Tissue Res 326:311–337
57. Schneggenburger R, Neher E (2000) Intracellular calcium dependence of transmitter release rates at a fast central synapse. Nature 46:889–893
58. Schneggenburger R, Neher E (2005) Presynaptic calcium and control of vesicle fusion. Curr Opin Neurobiol 15:266–274
59. Schneggenburger R, Sakaba T, Neher E (2002) Vescicle pools and short-term synaptic depression: lessons from a large synapse. Trends Neurosci 25:206–212
60. De Schutter E (ed) (2001) Computational neuroscience—realistic modeling for experimentalist. CRC Press, Boca Raton
61. Segel LA (1987) Modeling dynamic phenomena in molecular and cellular biology. Cambridge University Press, Cambridge
62. Smith GD (2005) Modeling the stochastic gating of ion channels. In: Fall CP, Marland ES, Wagner JM, Tyson JJ (eds) Computational cell biology, 2nd edn. Springer, New York, pp 285–319
63. Smolen PD, Baxter DA, Byrne JH (2004) Mathematical modeling and analysis of intracellular signaling pathways. In: Byrne JH, Roberts JL (eds) From molecules to networks—an introduction to cellular and molecular neuroscience. Elsevier/Academic Press, Amsterdam, pp 391–429
64. Sorensen JB (2004) Formation, stabilization and fusion of the readily releasable pool of secretory vescicles. Eur J Neurosci 448:347–362
65. Sudhof TC (2004) The synaptic vesicle cycle. Annu Rev Neurosci 27:509–547
66. Ullah M, Wolkenhauer O (2007) Family tree of Markov models in Systems Biology. IET Syst Biol 1:247–254
67. Voit EO (2000) Computational analysis of biochemical systems—a practical guide for biochemists and molecular biologists. Cambridge University Press, Cambridge
68. Weis S, Schneggenburger R, Neher E (1999) Properties of a model of Ca^{2+}-dependent vesicle pool dynamics and short term synaptic depression. Biophys J 77:2418–2429
69. Wilkinson DJ (2006) Stochastic modeling for System Biology. Chapman and Hall/CRC Press, London
70. Wolfel M, Lou X, Schneggenburger R (2007) A mechanism intrinsic to the vesicle fusion machinery determines fast and slow transmitter release at a large cns synapse. J Neurosci 27:3198–3210
71. Wolkenhauer O (2009) System Biology—dynamic pathway modeling (to appear)
72. Wolkenhauer O, Ullah M, Kolch W, Cho K-H (2004) Modeling and simulation of intracellular dynamics: choosing an appropriate framework. IEEE Trans Nanobiosci 3:200–207
73. Wong AYC, Graham BP, Billups B, Forsythe ID (2003) Distinguishing between presynaptic and postsynaptic mechanisms of short-term depression during action potentials trains. J Neurosci 23:4868–4877
74. Xu-Friedman MA, Regehr WG (2004) Structural contribution to short-term synaptic plasticity. Physiol Rev 84:69–85
75. Zucker RS, Kullmann DM, Schwartz TL (2004) Release of neurotransmitters. In: Byrne JH, Roberts JL (eds) From molecules to networks—an introduction to cellular and molecular neuroscience. Elsevier/Academic Press, San Diego, pp 197–244
76. Zucker RS, Regehr WG (2002) Short-term synaptic plasticity. Annu Rev Physiol 64:355–405

Understanding Network Behavior by Structured Representations of Transition Invariants

A Petri Net Perspective on Systems and Synthetic Biology

Monika Heiner

Abstract Petri nets offer a bipartite and concurrent paradigm, and consequently represent a natural choice for modeling and analyzing biochemical networks. We introduce a Petri net structuring technique contributing to a better understanding of the network behavior and requiring static analysis only. We determine a classification of the transitions into abstract dependent transition sets, which induce connected subnets overlapping in interface places only. This classification allows a structured representation of the transition invariants by network coarsening. The whole approach is algorithmically defined, and thus does not involve human interaction. This structuring technique is especially helpful for analyzing biochemically interpreted Petri nets, where it supports model validation of biochemical reaction systems reflecting current comprehension and assumptions of what has been designed by natural evolution.

1 Motivation

Systems and synthetic biology are concerned with understanding biochemical processes (pathways) in biological systems ranging in size from a single pathway to a whole organism, and varying in the chosen abstraction level from gene regulatory networks via signal transduction networks to metabolic networks.

Independently of size and abstraction level, all pathways and, therefore, their models, too, exhibit inherently rather complex network structures. These structures reflect the causal interplay of the basic actions and employ all the patterns well known in computer engineering, such as sequence, branching, repetition, and concurrency. However, opposite to technical networks, biochemical networks tend to be very dense and apparently unstructured making the understandability of the full network of interactions difficult and, therefore, error-prone.

M. Heiner (✉)
Department of Computer Science, Brandenburg University of Technology,
Postbox 101344, 03013 Cottbus, Germany
e-mail: monika.heiner@tu-cottbus.de
Fax: +49-355-693587

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_19, © Springer-Verlag Berlin Heidelberg 2009

Getting a survey on the current state of knowledge about a particular pathway requires a lot of reading and search through several data bases, including the creative interpretation of various graphical representations. These pieces of separate understanding have to be assembled to get a comprehensive as well as consistent knowledge representation. For this purpose, a readable and executable language with a formal, and hence unambiguous semantics would obviously be of great help as a common intermediate representation language. Formal models open the door to mathematically founded analyses. The transformation from an informal to a formal model involves the resolution of any ambiguities, which must not necessarily happen in the right way. Therefore, the next step in a sound model-based technology should be devoted to model validation.

Model validation aims basically at increasing our confidence in the constructed model. There is no doubt that this should be a prerequisite before raising more sophisticated questions, where the answers are supposed to be found by help of the model and where we are usually ready to trust the answers we get. So, before thinking about model-based behavior prediction, we are concerned with model validation.

For model validation, we introduce a qualitative model as a supplementary intermediate step, at least from the viewpoint of the biochemist accustomed to continuous modeling only. One of the benefits of using the qualitative approach is that systems can be modeled and analyzed without any quantitative parameters.

This model-driven perspective is equally helpful in the setting of systems biology as well as synthetic biology. In systems biology, models help us in formalizing our understanding of what has been created by natural evolution. So first of all, models serve as an unambiguous representation of the acquired knowledge and help to design new wetlab experiments to sharpen our comprehension. In synthetic biology, models help us to make the engineering of biology easier and more reliable. Models serve as blueprint for novel synthetic biological systems. Their employment is highly recommended to guide the design and construction in order to ensure that the behavior of the synthetic biological systems is reliable and robust under a variety of conditions.

Computer science has generated quite a number of modeling formalisms, which are used in the scenario sketched so far. In this paper, we apply the Petri net formalism. Biochemical reaction systems and Petri nets share two distinctive characteristics. Both are inherently bipartite, and both are inherently concurrent. Thus, Petri nets seem to be a natural choice for modeling biochemical networks. Petri nets are known to combine an intuitive and executable modeling style with mathematically founded analysis techniques, comprising qualitative as well as quantitative ones, complemented by reliable tool support.

This paper is based on a typical static analysis technique, the invariants. Place and transition invariants are a popular validation technique for technical as well as biochemical networks. One approach of acquiring a deeper understanding of the network behavior consists in understanding all its basic executions, which correspond to the minimal transition invariants. We go one step further by guiding the hierarchical structuring (coarsening) of a given network to support its comprehension. We determine a classification of the transitions into abstract dependent transition sets, which induce connected subnets overlapping in interface places only. This classification allows a structured representation of the transition invariants by network coarsening.

The whole approach is algorithmically defined, and thus does not involve human interaction. This structuring technique is especially helpful for analyzing biochemically interpreted Petri nets, where it supports model validation of biochemical reaction systems reflecting current comprehension and assumptions.

This paper is organized as follows. In the next section, we recapitulate the relevant Petri net notions, before we motivate a biochemical interpretation of Petri nets. Thereafter, we introduce a new structuring method, sketch the computation of its two main features, and demonstrate the gained structuring effect by three smaller cases studies, which are also provided on our web pages. Finally, we refer to related work, before concluding with a short summary of the essential aspects.

2 Preliminaries

To be self-contained, we give the formal definitions of the Petri net notions relevant for this paper. As usual, we denote the set of non-negative integers including zero by $\mathbb{N}_0$, and the set of integers by $\mathbb{Z}$. $|S|$ denotes the number of elements in a set S.

To allow formal reasoning, we are going to represent biochemical networks by Petri nets, which enjoy formal semantics amenable to mathematically sound analysis techniques. The first two definitions introduce the standard notion of place/transition Petri nets, which is the basic class in the ample family of Petri net models.

Definition 1 (Petri Net, Syntax) A Petri net is a quadruple $\mathcal{N} = (P, T, f, m_0)$, where

- P and T are finite sets with $P \cup T \neq \emptyset$, $P \cap T = \emptyset$,
- $f : ((P \times T) \cup (T \times P)) \to \mathbb{N}_0$,
- $m_0 : P \to \mathbb{N}_0$.

Thus, Petri nets (or nets for short) are weighted, directed, bipartite graphs. The elements of the set P are called *places*, graphically represented by circles, while the elements of the set T are called *transitions*, represented by rectangles. The function f defines the set of directed *arcs*, weighted by non-negative integers. The (pseudo) arc weight 0 stands for the absence of an arc. The arc weight 1 is the default value and is usually not given explicitly. A place carries an arbitrary number of *tokens*, represented as black dots or a natural number. The number zero is the default value and usually not given explicitly. $m(p)$ yields the number of tokens on place p in the marking m, and m_0 specifies the *initial marking*.

We introduce the following notions and notations for a node $x \in P \cup T$.

- ${}^{\bullet}x := \{y \in P \cup T \mid f(y, x) \neq 0\}$ is the preset of x.

- $x^{\bullet} := \{y \in P \cup T \mid f(x, y) \neq 0\}$ is the postset of x.
- x is called input node of the net if ${}^{\bullet}x = \emptyset$.
- x is called output node of the net if $x^{\bullet} = \emptyset$.
- x is called boundary node of the net if it is either input or output node.

Additionally, we extend the first two notions to a set of nodes $X \subseteq P \cup T$ and define the set of all pre-nodes ${}^{\bullet}X := \bigcup_{x \in X} {}^{\bullet}x$, and the set of all post-nodes $X^{\bullet} := \bigcup_{x \in X} x^{\bullet}$.

Up to now, we have introduced the static aspects of a Petri net only. The behavior of a net is defined by the firing rule, which basically consists of two parts: the precondition and the firing itself.

Definition 2 (Firing Rule) Let $\mathcal{N} = (P, T, f, m_0)$ be a Petri net.

- A transition t is enabled in a marking m, written as $m[t\rangle$, if $\forall p \in^{\bullet} t : m(p) \geq f(p, t)$, else disabled.
- A transition t, which is enabled in m, may fire.
- When t in m fires, a new marking m' is reached, written as $m[t\rangle m'$, with $\forall p \in P : m'(p) = m(p) - f(p, t) + f(t, p)$.
- The firing happens atomically and does not consume any time.

According to this *may* firing rule, a transition is never forced to fire. Figuratively, the firing of a transition moves tokens from its pre-places to its post-places, while possibly changing the number of tokens; compare Fig. 1. Generally, the firing of a transition changes the formerly current marking to a new reachable one, where some transitions are not enabled anymore while others get enabled. The repeated firing of transitions establishes the behavior of the net. The whole net behavior consists of all possible partially ordered firing sequences (partial order semantics) or all possible totally ordered firing sequences (interleaving semantics), respectively.

Every marking m is defined by the given token situation in all places, i.e. $m \in \mathbb{N}_0^{|P|}$. All markings, which can be reached from a given marking m by any firing sequence of arbitrary length, constitute the *set of reachable markings* $[m\rangle$. The set of markings $[m_0\rangle$ reachable from the initial marking is said to be the *state space* of a given system. However, in this paper, we confine ourselves deliberately to analysis techniques, which do not require the generation of the state space. So, the presented approach works also for nets with infinite state spaces, i.e. for unbounded Petri nets.

To open the door to analysis techniques based on linear algebra (or better: discrete computational geometry), we represent the net structure by a matrix, called incidence matrix in the Petri net community and stoichiometric matrix in systems biology. We briefly recall the essential technical terms.

Definition 3 (P-Invariants, T-Invariants) Let $\mathcal{N} = (P, T, f, m_0)$ be a Petri net.

- The incidence matrix of $\mathcal{N}$ is a matrix $\mathbb{C} : P \times T \to \mathbb{Z}$, indexed by P and T, such that $\mathbb{C}(p, t) = f(t, p) - f(p, t)$.
- A place vector (transition vector) is a vector $x : P \to \mathbb{Z}$, indexed by P $(y : T \to \mathbb{Z}$, indexed by $T)$.

- A place vector (transition vector) is called a P-invariant (T-invariant) if it is a non-trivial non-negative integer solution of the homogeneous linear equation system $x \cdot \mathbb{C} = 0$ ($\mathbb{C} \cdot y = 0$).
- The set of nodes corresponding to an invariant's non-zero entries are called the support of this invariant x, written as $supp(x)$.
- An invariant x is called minimal if $\nexists$ invariant $z : supp(z) \subset supp(x)$, i.e. its support does not contain the support of any other invariant z, and the greatest common divisor of all non-zero entries of x is 1.
- A net is covered by P-invariants, shortly CPI (covered by T-invariants, shortly CTI) if every place (transition) belongs to a P-invariant (T-invariant).

Invariants are vectors over natural numbers, which can be read as specifications of multisets. Contrary, supports are sets, which can technically be specified as vectors over Booleans, which allows the access to the ith entry by indexing.

The set X of all minimal P-invariants (T-invariants) x_i of a given net is unique and represents a generating system for all P-invariants (T-invariants). All invariants x can be computed as non-negative linear combinations: $n \cdot x = \sum(a_i \cdot x_i)$, with $n, a_i \in \mathbb{N}_0$, i.e. the allowed operations are addition, multiplication by a natural number, and division by a common divisor.

3 Biochemically Interpreted Petri Nets

The idea to use Petri nets for the representation of biochemical networks is rather intuitive and has been mentioned by Carl Adam Petri himself in one of his internal research reports on interpretation of net theory in the seventies. It has also been used as the very first introductory example in one of the early survey papers [28]. We follow this approach; see Fig. 1.

Places usually model passive system components like conditions, species, or any kind of chemical compounds, e.g. proteins or proteins complexes, playing the role of precursors, products, or enzymes of chemical reactions. Occasionally, we want to differentiate between primary and secondary compounds. The latter ones are often assumed to be ubiquitous and available in sufficient amount.

Complementary, transitions stand usually for active system components like atomic actions or any kind of chemical reactions, e.g. association, dissociation, phosphorylation, or dephosphorylation, transforming precursors into products, possibly controlled by enzymes. A reversible chemical reaction is modeled by two opposite transitions; compare Fig. 2.

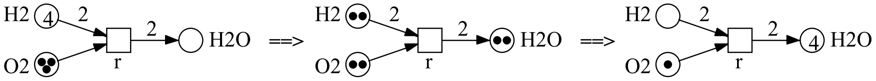


Fig. 1 The Petri net for the well-known chemical reaction r: $2H_2 + O_2 \rightarrow 2H_2O$ and three of its markings (states), connected each by a firing of the transition r. The transition is not enabled anymore in the marking reached after these two single-firing steps

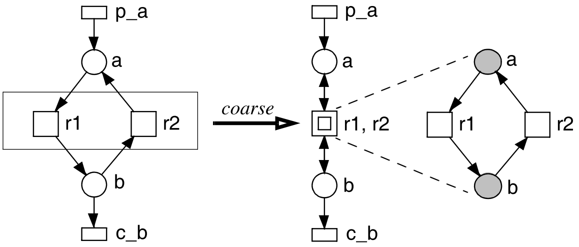


Fig. 2 Hierarchical structuring by use of macro transitions, which are drawn as two centric squares. The flat net (*left*) and the hierarchical net (*right*) are identical—from an analysis point of view. Both nets model a reversible reaction $a \rightleftharpoons b$ with its producing and consuming environment. The nodes colored in *gray* may be considered as logical nodes, automatically generated by the drawing tool. They connect the transition-bordered subnet on the lower hierarchy level with its environment on the next higher hierarchy level

The arcs go from precursors to reactions (ingoing arcs), and from reactions to products (outgoing arcs). In other words, the pre-places of a transition correspond to the reaction's precursors, and its post-places to the reaction's products. Enzymes establish side conditions and are connected in both directions with the reaction they catalyze – we get read arcs; compare place O2 in Fig. 6.

Arc weights may be read as the multiplicity of the arc, reflecting known stoichiometries. Tokens can be interpreted as the available amount of a given species in number of molecules or moles, or any abstract, i.e. discrete concentration level.

We adopt the following drawing conventions; compare Fig. 2.

- Input/output transitions are generally drawn as flat rectangles to highlight their special meaning for the net behavior.
- Logical nodes (fusion nodes) are colored in gray. All logical nodes with the same name are identical, at least from an analysis point of view. They are commonly used for compounds involved in many reactions, e.g. secondary compounds.
- Transition-bordered subnets can be hidden in macro transitions, drawn as two centric squares. This allows an hierarchical structuring of larger nets. We are going to apply this technique to coarsen a given net according to its minimal T-invariants' inherent structure; see Sect. 4.

Invariants are a beneficial technique in model validation, and the challenge is to check all invariants for their biological plausibility.

A *P-invariant* x is a non-zero and non-negative integer place vector such that $x \cdot \mathbb{C} = 0$; in words, for each transition it holds that: multiplying the P-invariant with the transition's column vector yields zero. Thus, the total effect of each transition on the P-invariant is zero, which explains its interpretation as a token conservation component. A P-invariant stands for a set of places over which the weighted sum of tokens is constant and independent of any firing, i.e. for any markings m_1, m_2, which are reachable by the firing of transitions, it holds that $x \cdot m_1 = x \cdot m_2$. In the context of metabolic networks, P-invariants reflect substrate conservations, while in signal transduction or gene regulatory networks P-invariants often correspond to the several states of a given species (protein or protein complex) or gene. A place belonging to a P-invariant is obviously bounded, and CPI causes structural boundedness, i.e. boundedness for any initial marking.

Analogously, a *T-invariant y* is a non-zero and non-negative integer transition vector such that $\mathbb{C} \cdot y = 0$; in words, for each place it holds that: multiplying the place's row with the T-invariant yields zero. Thus, the total effect of the T-invariant on a marking is zero. A T-invariant has two interpretations in the given biochemical context.

– The entries of a T-invariant specify a multi-set of transitions, which by their partially ordered firing reproduce a given marking, i.e. basically occurring one after the other. This partial order sequence of the T-invariant's transitions may contribute to a deeper understanding of the net behavior. A T-invariant is called *feasible* if such a behavior is actually possible in the given marking situation.
– The entries of a T-invariant may also be read as the relative firing rates of the transitions involved, all of them occurring permanently and concurrently. This activity level corresponds to the steady state behavior.

The two opposite transitions modeling the two directions of a reversible reaction always make a minimal T-invariant; thus, they are called *trivial T-invariants*. A net which is covered by non-trivial T-invariants is said to be *strongly covered by T-invariants* (SCTI). Transitions not covered by non-trivial T-invariants are candidates for model reduction, e.g. if the model analysis is concerned with steady state analysis only.

The automatic identification of non-trivial minimal T-invariants is in general useful as a method to highlight important parts of a network, and hence aid its comprehension by biochemists, especially when the entire network is too complex to easily comprehend.

We are especially interested in a network's input/output behavior, which we are going to characterize by input/output T-invariants (I/O T-invariants), i.e. such T-

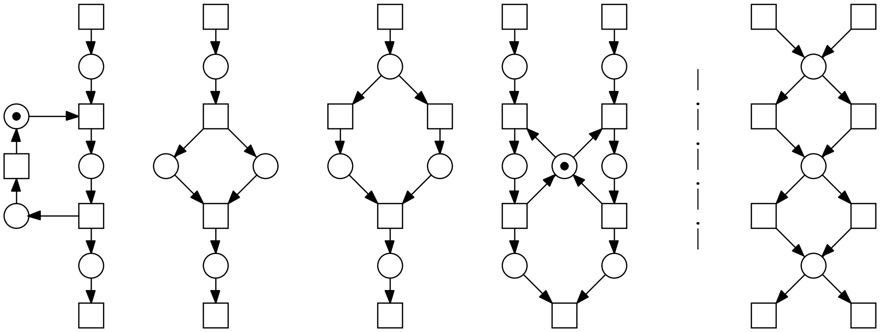


Fig. 3 The four nets on the left are each covered by one minimal T-invariant. Invariants can contain any structures (*from left to right*): cycles, forward/backward branching transitions, forward branching places, backward branching places. Generally, invariants overlap, and in the worst-case there are exponentially many of them; the net on the far-right has 2^4 T-invariants

invariants, involving input and output transitions. These special T-invariants can often be read as alternative, self-contained pathways within a given network under consideration.

A minimal P-invariant (T-invariant) defines a connected subnet, consisting of its support, its pre- and post-transitions (pre- and post-places), and all arcs in between. There are no structural limitations for such subnets induced by minimal invariants, compare Fig. 3, but they are always connected, however, not necessarily strongly connected. These minimal self-contained subnets may be read as a decomposition into token preserving or state repeating modules, which should have an enclosed biological meaning.

Minimal invariants generally overlap; the combinatorial effect causes an explosion of the number of minimal invariants. There are exponentially many of them in the worst-case; compare Fig. 3, far-right. Therefore, we are going to apply a structured representation of a given set of invariants.

4 Structuring Method

The following discussion concentrates on T-invariants. Likewise, the presented technique can be applied to P-invariants due to the given symmetry of the two notions.

We define a dependency relation based on a set of minimal T-invariants. It can be equally applied to the full set of all minimal T-invariants as well as to a subset, e.g. the set of non-trivial T-invariants.

Definition 4 (Dependency Relation) Let $\mathcal{N} = (P, T, f, m_0)$ be a Petri net, and let Y denote a set of minimal T-invariants y of $\mathcal{N}$. Two transitions $i, j \in T$ depend on each other, $i \bowtie j$ for short, if

$$\forall y \in Y : i \in supp(y) \Leftrightarrow j \in supp(y).$$

This is an abstract dependency, defined on the T-invariants' support only. Dependent transitions appear always together in the given set of minimal T-invariants. The drop out of one transition prevents the whole set of transitions depending on each other to accomplish their common function.

The dependency relation fulfills the following properties:

- reflexivity: $i \bowtie i$;
- a transition depends on its own.
- symmetry: $i \bowtie j \Leftrightarrow j \bowtie i$;
  the dependency of i on j implies the dependency of j on i, and vice versa.
- transitivity: $i \bowtie j \wedge j \bowtie k \Rightarrow i \bowtie k$;
  if i depends on j, and j depends on k, then i depends also on k.

Thus, it is an equivalence relation in the transition set T, leading to a partition of T. We call the equivalence classes A_i with

$$A_i \subseteq T \wedge \cup A_i = T \wedge \forall i, j : i \neq j \Rightarrow A_i \cap A_j = \emptyset$$

maximal *abstract dependent transition sets* (ADT sets), and it holds

$$\forall A_i,\ \forall y \in Y : A_i \subseteq supp(y) \vee A_i \cap supp(y) = \emptyset.$$

ADT sets can be read as the smallest biologically meaningful functional units (building blocks). Contrary to T-invariants, which generally overlap, ADT sets induce by definition subnets overlapping in interface places $p_{if} \in P_{IF}$ only, with

$$P_{IF} = \bigcup_{\forall i,j,i\neq j} ({}^{\bullet}A_i \cup A_i{}^{\bullet}) \cap ({}^{\bullet}A_j \cup A_j{}^{\bullet}).$$

These subnets represent a possible structural decomposition of biochemical networks into smaller subnets. Notably, the decomposition is based on statically decidable properties only.

Following the idea of hierarchical structuring of larger networks, we are going to hide building blocks within macro transitions. However, ADT sets are not necessarily connected, as we will see in Sect. 6. Hence, a further decomposition into connected ADT sets is generally needed, possibly according to primary compound flow only, i.e. neglecting connections by secondary compounds, and we get non-maximal ADT sets.

Having a decomposition of the transition set T into ADT sets inducing connected subnets, we are able to determine the interface places, and to coarsen automatically a given net according to the minimal T-invariants' inherent structure:

- macro transitions abstract from connected ADT sets, and
- places on the hierarchy's top level correspond to the interface between the ADT sets.

Then the coarse net structure gives a structured representation of all T-invariants, which may contribute to a better understanding of the net behavior. Moreover, the coarse net structure allows to identify sensitive net parts, i.e. interface places; the knock-out of which would switch off a significant part of the whole network or even prevent any output.

Maximal ADT sets support also the efficient design of wetlab experiments by identifying minimal sets of observation points providing coverage of the whole network: each maximal ADT set needs obviously one observation point only.

Finally, ADT sets are likely to be useful for automatic layout algorithms, whereby the differentiation between primary and secondary compounds might be supportive.

5 Computation

For the algorithmic-oriented minds, we sketch the computation of the two main features of which our structuring approach is made.

5.1 Computation of Invariants

Technically, we need to solve a homogeneous linear equation system over non-negative integers. This restriction of the data space establishes—from a strong mathematical point of view—a challenge. There is no closed formula to compute the solutions. However, there are algorithms—actually, a class of algorithms—constructing the solution (to be precise: the generating system for the solution space) by systematically considering all possible candidates.

This algorithm class has been repetitively re-invented over the years. Thus, these algorithms come along with different names. But a closer look always reveals the same underlying principle. All these versions may be classified as "positive Gauss elimination"; the incidence matrix of the Petri net is systematically transformed to a zero matrix by suitable matrix operations.

Before we start, an auxiliary matrix is added to the incidence matrix to log, which matrix operations have been done. The auxiliary matrix is always a quadratic matrix. It is initialized by the identity matrix (diagonal is set to 1, else 0), and it is added to the right for the computation of the P-invariants (then it is a quadratic matrix over the places), or it is added below the incidence matrix for the computation of the T-invariants (then it is a quadratic matrix over the transitions). The matrix operations, compare Algorithm 1, are always applied to the composed matrix, consisting of the incidence matrix and the auxiliary matrix.

The algorithm terminates, when all columns in the incidence matrix are zero. It needs at most as many iterations of the outer loop as we have transitions, because each iteration makes one column to zero.

Algorithm 1 Computation of P-invariants

```
input incidence matrix $\mathbb{C}$, extended by auxiliary matrix;

while there are non-zero columns in $\mathbb{C}$ do
    pick one non-zero column $i$ in $\mathbb{C}$;
    for all pairs of rows with unequally signed entries in this column $i$ do
        add a new row, which is the smallest possible linear combination of this
        pair, making the matrix entry in this column $i$ to zero;
    end for;
    delete all old rows, i.e. those which have been used in creating these linear
    combinations;
    assert $i$ is now a zero column;
    assert if we had $n$ negative entries and $p$ positive entries in column $i$,
           then the number of rows changes by $n \cdot p - (n + p)$
end while;
assert if there is a solution, the incidence matrix is now zero;
assert all rows in the auxiliary matrix are P-invariants,
       among them are all minimal P-invariants;
```

The challenge in implementing this basic algorithm is twofold. First, we need to eliminate efficiently all non-minimal P-invariants. Second, because the algorithm has to consider all possible candidates, all possible linear combinations are constructed, blowing up the number of rows in the intermediate data structure. There are heuristics trying to minimize this effect, e.g. to pick a column, for which we get less new rows. However, as we know, heuristics never work fine for all possible cases. To give some figures: It might be that there are several millions of rows at an intermediate state of the algorithm, and at the end there are just around 100 left.

It is straightforward to adjust this algorithm to compute T-invariants; or the incidence matrix is transposed—P-invariants of the transposed net are the T-invariants of the original net.

5.2 Computation of Dependent Sets

The algorithm is rather straightforward and easily explained. Let us recall, T-invariants are technically transition vectors over natural numbers, i.e. they have as many components as there are transitions in the net, usually given as column vectors. Likewise, their supports can be given as transition vectors over Booleans with *true* if the transition belongs to the set, and *false* else; again written as column vectors. Let us arrange these column vectors of all T-invariants or of their supports side by side. We get a matrix $\mathbb{T}_{\text{inv}}$ with as many rows as we have transitions and as many columns as we have T-invariants.

The dependency relation can now be rephrased in terms of this matrix $\mathbb{T}_{\text{inv}}$: two transitions dependent on each other, i.e. they always occur together, if their rows are identical. Maximal dependent transition sets are now defined by maximal sets of identical rows. To compute them, we execute Algorithm 2.

The algorithm terminates when all rows have been assigned. Because we have a finite set of transition, the number of rows is finite, too. In the worst case, the outer

Algorithm 2 Computation of maximal dependent transition sets

```
input matrix $\mathbb{T}_{\text{inv}}$;

while there are non-assigned rows do
    create a new set $s$;
    let $i$ be the first index of a row, which has not been assigned to a set;
    mark row $i$ as assigned, and put $i$ into $s$;
    for all non-assigned rows $j$ do
        if rows $i$, $j$ are identical, i.e. $\mathbb{T}_{\text{inv}}(i,*) = \mathbb{T}_{\text{inv}}(j,*)$
        then $j$ belongs to the same set as $i$: mark $j$ as assigned, and put $j$ into $s$
        end if
    end for;
    assert $s$ specifies a maximal ADT set;
end while
```

loop is entered as often as there are transitions. Then each transition builds its own set.

6 Case Studies

We present deliberately three smaller case studies, allowing to be easily understood.

6.1 Glycolysis

We start with one of the standard examples of metabolic networks, the combined glycolysis and pentose phosphate pathway in erythrocytes (red blood cell). We use a version based on [38], which is also elaborated in [19]. The network defines the various reactions occurring in the cell under heavy energy load, such as in brisk muscle activity. Glucose serves as precursor, and Lactate as product, involving several secondary compounds (ATP, ADP, NADH+, NADH, Pi) in the stepwise conversion process; compare Fig. 4.

There are three minimal T-invariants, which we give in a short-hand notation, enumerating the non-zero entries only:

$$\begin{aligned}
y_1 = (&\text{p\_Gluc}, 2\cdot\text{p\_ADP}, 2\cdot\text{p\_Pi},\\
&\text{r9}, \text{r10}, \text{r11}, \text{r12}, \text{r13}, 2\cdot\text{r15}, 2\cdot\text{r16}, 2\cdot\text{r17}, 2\cdot\text{r18}, 2\cdot\text{r19}, 2\cdot\text{r20},\\
&2\cdot\text{c\_Lac}, 2\cdot\text{c\_ATP}),\\
y_2 = (&3\cdot\text{p\_Gluc}, 5\cdot\text{p\_ADP}, 5\cdot\text{p\_Pi},\\
&3\cdot\text{r9}, 6\cdot\text{r1}, 6\cdot\text{r2}, 3\cdot\text{r3}, 2\cdot\text{r4}, \text{r5}, \text{r6}, \text{r7}, \text{r8},\\
&2\cdot\text{r11}, 2\cdot\text{r12}, 2\cdot\text{r13}, 5\cdot\text{r15}, 5\cdot\text{r16}, 5\cdot\text{r17}, 5\cdot\text{r18}, 5\cdot\text{r19}, 5\cdot\text{r20},\\
&5\cdot\text{c\_Lac}, 5\cdot\text{c\_ATP}),\\
y_3 = (&\text{r13}, \text{r14}).
\end{aligned}$$

The net is CTI, however, not SCTI, because r14 is involved in a trivial T-invariant only. Considering the two non-trivial minimal T-invariants, y_1 and y_2, we find four maximal ADT sets. The first set contains the intersection of both T-invariants, comprising almost the whole glycolysis

$$\begin{aligned}
\text{A} &= supp(y_1) \cap supp(y_2)\\
&= \{\text{p\_Gluc}, \text{p\_ADP}, \text{p\_Pi},\\
&\quad \text{r9}, \text{r11}, \text{r12}, \text{r13}, \text{r15}, \text{r16}, \text{r17}, \text{r18}, \text{r19}, \text{r20},\\
&\quad \text{c\_Lac}, \text{c\_ATP}\}.
\end{aligned}$$

The knock-out of one of the transitions in A switches off both non-trivial T-invariants. The next two sets contain those transitions, which are specific to one

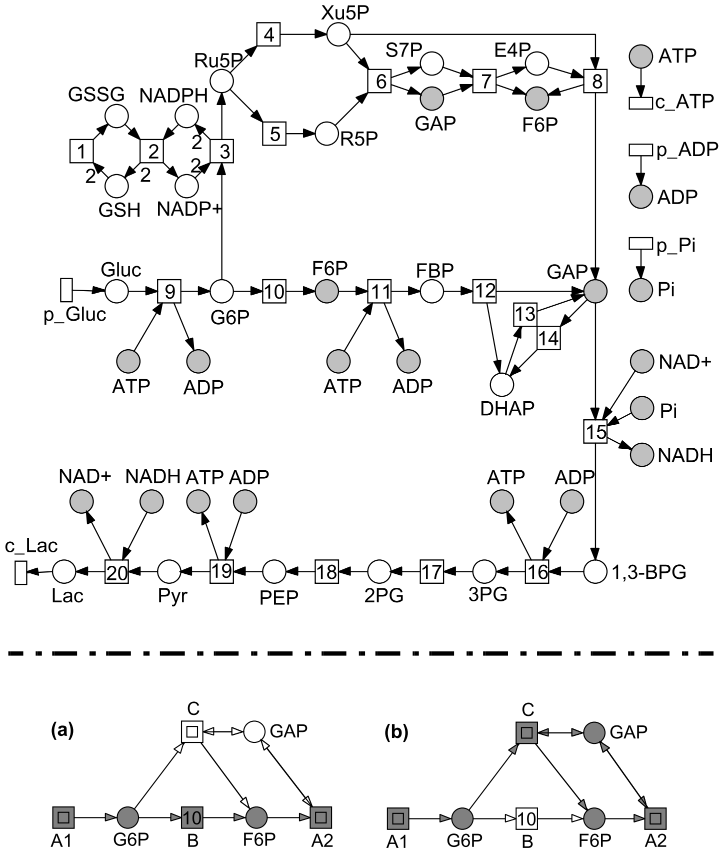


Fig. 4 The Petri net and its coarse structure for the combined glycolysis and pentose phosphate pathway in erythrocytes. The layout of the flat net mimics the hypergraph given in [38]. Nodes colored in gray in the flat net are logical (fusion) nodes. Input and output transitions are drawn as flat rectangles. The two pathways highlighted in the coarse net are: (**a**) glycolysis, and (**b**) pentose phosphate pathway

of the two T-invariants. The specific transition of the T-invariant y_1 belongs to the glycolysis

$$\begin{aligned} \mathrm{B} &= supp(y_1) - supp(y_2) \\ &= \{\mathrm{r10}\}, \end{aligned}$$

and the specific transitions of the T-invariant y_2 cover the pentose phosphate pathway

$$\begin{aligned} \mathrm{C} &= supp(y_2) - supp(y_1) \\ &= \{\mathrm{r1, r2, r3, r4, r5, r6, r7, r8}\}. \end{aligned}$$

The remaining transition belongs to a trivial T-invariant only; it builds an (pseudo) ADT set on its own. This transition does not contribute to the steady state behavior of the two non-trivial T-invariants.

$$\begin{aligned} \mathrm{D} &= \mathrm{T} - supp(y_1) - supp(y_2) \\ &= \{\mathrm{r14}\} \end{aligned}$$

Thus, the main building blocks of the Petri net, and by this way of the underlying biochemical network, are represented by the first three ADT sets, each defining a connected subnet. The two subnets, describing the two pathways, are defined by the union of the first ADT set with the second or third one, respectively. However, if we neglect the connectivity established by secondary compounds, the ADT set *A* breaks down into two subsets:

$$\mathrm{A1} = \{\mathrm{p\_Gluc}, \mathrm{r9}\}, \qquad \mathrm{A2} = \mathrm{A} - \mathrm{A1},$$

which are connected according to the *primary compound flow*, however, not maximal anymore.

We obtain the coarse network structure as given in Fig. 4, lower part, highlighting the structuring principle inherent in the non-trivial minimal T-invariants. Each macro transition stands for a connected subnet defined by a set of transitions, occurring together in all non-trivial minimal T-invariants.

In this example, each elementary (loop-free) macro transition sequence in the coarse net structure corresponds to a non-trivial minimal T-invariant of the whole network. There are two such sequences:

$$\begin{aligned} y_1 &= (\mathrm{A1}; \mathrm{B}; \mathrm{A2}), \\ y_2 &= (\mathrm{A1}; \mathrm{C}; \mathrm{A2}), \end{aligned}$$

sharing the beginning and the end. Thus, the two I/O T-invariants y_1, y_2 are now represented by I/O macro transition sequences. The places shown in the coarse net structure are the boundary places of the subnets, building the interface between the subnets. Please note, only the primary compound flow is represented here.

6.2 Apoptosis

The term apoptosis refers to the genetically programmed cell death, which is an essential part of normal physiology for most metazoan species. Disturbances in the apoptotic process may lead to various diseases. The signal transduction network of apoptosis governs complex mechanisms to control and execute programmed cell death, which are—by the time being—not really well understood. A variety of different cellular signals initiate activation of apoptosis in distinctive ways, depending on the various cell types and their biological states. We consider here a core model

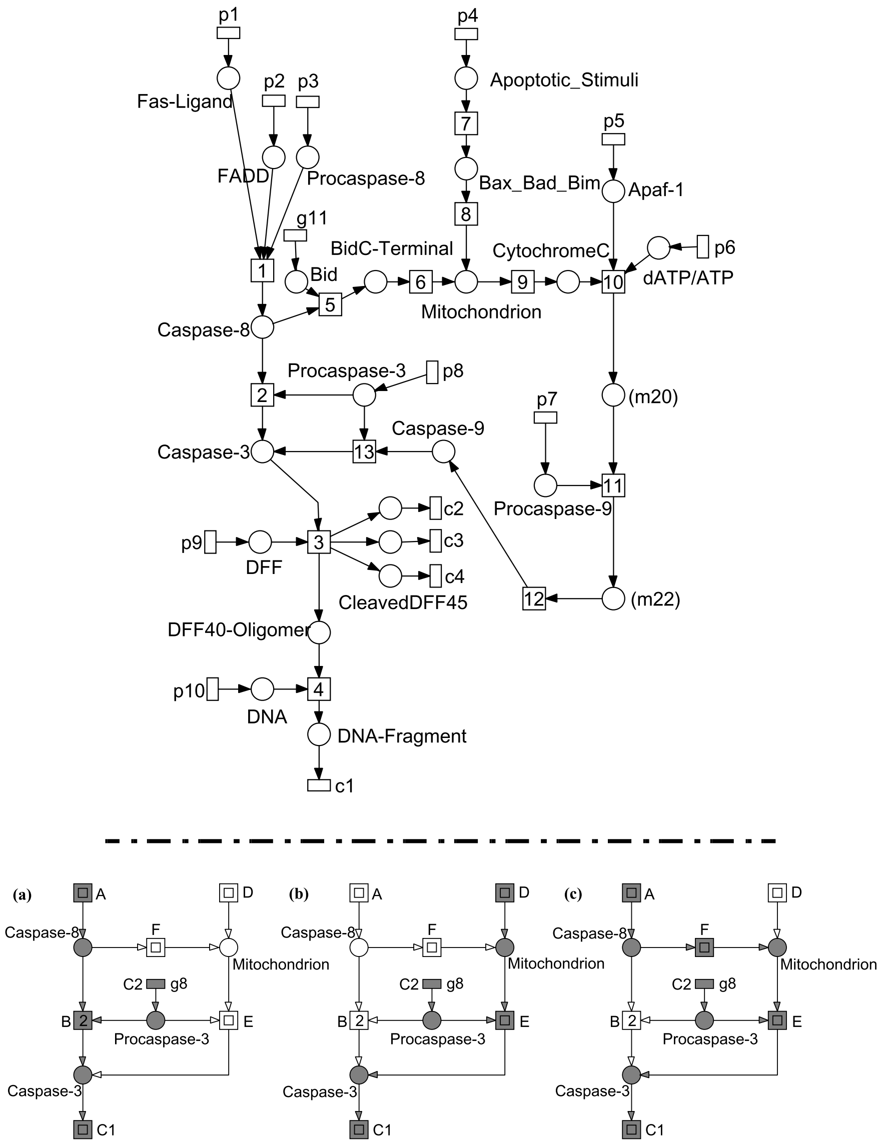


Fig. 5 The Petri net and its coarse structure for a core model of the apoptosis. The layout of the flat net is inspired by the graphical scheme given in [27]. The three pathways highlighted in the coarse net are: (**a**) the Fas receptor pathway, (**b**) the pathway induced by intrinsic apoptotic stimuli, and (**c**) the cross-talk pathway. The ADT sets C1 and C2 are involved in all three pathways

of [16], which is based on [27], comprising the pathways induced by the Fas receptor and the intrinsic apoptotic stimuli, as well as the cross-talk in between; compare Fig. 5.

There are three minimal T-invariants, covering the net:

$$\begin{aligned}
y_1 = (&\mathrm{p1, p2, p3, p8, p9, p10,}\\
&\mathrm{r1, r2, r3, r4,}\\
&\mathrm{c1, c2, c3, c4}),\\
y_2 = (&\mathrm{p4, p5, p6, p7, p8, p9, p10,}\\
&\mathrm{r3, r4, r7, r8, r9, r10, r11, r12, r13,}\\
&\mathrm{c1, c2, c3, c4}),\\
y_2 = (&\mathrm{p1, p2, p3, p5, p6, p7, p8, p9, p10, p11,}\\
&\mathrm{r1, r3, r4, r5, r6, r9, r10, r11, r12, r13,}\\
&\mathrm{c1, c2, c3, c4}).
\end{aligned}$$

There are no trivial T-invariants; so, CTI implies SCTI. We consider all minimal T-invariants, and we get six maximal ADT sets:

$$\begin{aligned}
\mathrm{A} &= \{\mathrm{p1, p2, p3, r1}\},\\
\mathrm{B} &= \{\mathrm{r2}\},\\
\mathrm{C} &= \{\mathrm{p8, p9, p10, r3, r4, c1, c2, c3, c4}\},\\
\mathrm{D} &= \{\mathrm{p4, r7, r8}\},\\
\mathrm{E} &= \{\mathrm{p5, p6, p7, r9, r10, r11, r12, r13}\},\\
\mathrm{F} &= \{\mathrm{p11, r5, r6}\}.
\end{aligned}$$

Notably, the ADT set C is involved in all minimal T-invariants; so, it is vital for the whole network. This set does not induce a connected subnet; therefore, we decompose it into two connected subsets:

$$\begin{aligned}
\mathrm{C1} &= \{\mathrm{p9, p10, r3, r4, c1, c2, c3, c4}\},\\
\mathrm{C2} &= \{\mathrm{p8}\}.
\end{aligned}$$

Consequently, C1 and C2 are not maximal ADT sets anymore. Using these seven ADT sets, we get the coarse net structure as given in Fig. 5, lower part. The three pathways are clearly distinguishable, and—we claim—much better readable than in the flat net. In this example, each minimal I/O T-invariant is represented by a partially ordered I/O macro transition sequence in the coarse net structure (the sign + stands for 'unordered', i.e. concurrent macro transitions):

$$\begin{aligned}
y_1 &= (\mathrm{A + C2; B; C1}),\\
y_2 &= (\mathrm{C2 + D; E; C1}),\\
y_3 &= (\mathrm{(A; F) + C2; E; C1}).
\end{aligned}$$

6.3 Hypoxia

Oxygen is an essential and vital element for the survival of organisms. Lower oxygen content, termed hypoxia, arises under pathophysiological conditions. When there is an imbalance of oxygen content, the organism adapts by restoring normal oxygen content through activation of various genetic and metabolic pathways to compensate for the imbalance. One of the well-studied molecular pathways activated under hypoxia condition is the Hypoxia Induced Factor (HIF) pathway responsible for regulating oxygen-sensitive gene expression. Continuous models in the style of ordinary differential equations (ODEs) have been proposed in [21] and [51]. The Petri net given in Fig. 6 has been derived from these ODEs in order to highlight the ODEs' inherent structure. Reading the given qualitative Petri net as a continuous Petri net, whereby all transitions firing rates follow the mass action kinetics, generates exactly the original ODEs.

Here, we confine ourselves to the very first step—understanding the essential network behavior. We start with the computation of the minimal T-invariants. Besides the expected seven trivial T-invariants for the seven reversible reactions,

$$
\begin{aligned}
&y_1 = (\mathrm{r3}, \mathrm{r4}), && y_2 = (\mathrm{r5}, \mathrm{r6}), && y_3 = (\mathrm{r12}, \mathrm{r13}),\\
&y_4 = (\mathrm{r15}, \mathrm{r16}), && y_5 = (\mathrm{r18}, \mathrm{r19}), && y_6 = (\mathrm{r21}, \mathrm{r22}),\\
&y_7 = (\mathrm{r29}, \mathrm{r30}),
\end{aligned}
$$

there are three non-trivial ones:

$$
\begin{aligned}
y_8 &= (\mathrm{r1}, \mathrm{r2}),\\
y_9 &= (\mathrm{r1}, \mathrm{r12}, \mathrm{r14}, \mathrm{r18}, \mathrm{r20}),\\
y_{10} &= (\mathrm{r1}, \mathrm{r3}, \mathrm{r15}, \mathrm{r17}, \mathrm{r18}, \mathrm{r20}, \mathrm{r22}).
\end{aligned}
$$

Please note, (r1, r2) is not considered to be a trivial T-invariant due to its relevance for the input/output behavior. Determining the maximal ADT sets over all T-invariants yields 17 sets, 15 of them contain just one transition, and the remaining two are {r5, r6} and {r29, r30}, i.e. they correspond to those two trivial T-invariants, the transitions of which are not involved in any of the non-trivial T-invariants. Neglecting the trivial T-invariants in the computation of the maximal ADT sets yields the much more interesting result:

$$
\begin{aligned}
&\mathrm{A} = \{\mathrm{r1}\}, && \mathrm{B} = \{\mathrm{r2}\}, && \mathrm{C} = \{\mathrm{r12}, \mathrm{r14}\}, && \mathrm{D} = \{\mathrm{r18}, \mathrm{r20}\},\\
&\mathrm{E} = \{\mathrm{r3}, \mathrm{r15}, \mathrm{r17}, \mathrm{r22}\},
\end{aligned}
$$

and the pseudo ADT set, containing all remaining transitions of the net, not contributing to the non-trivial T-invariants. The maximal ADT sets A–E induce connected subnets, and we get the coarse net structure as given in Fig. 6, lower part.

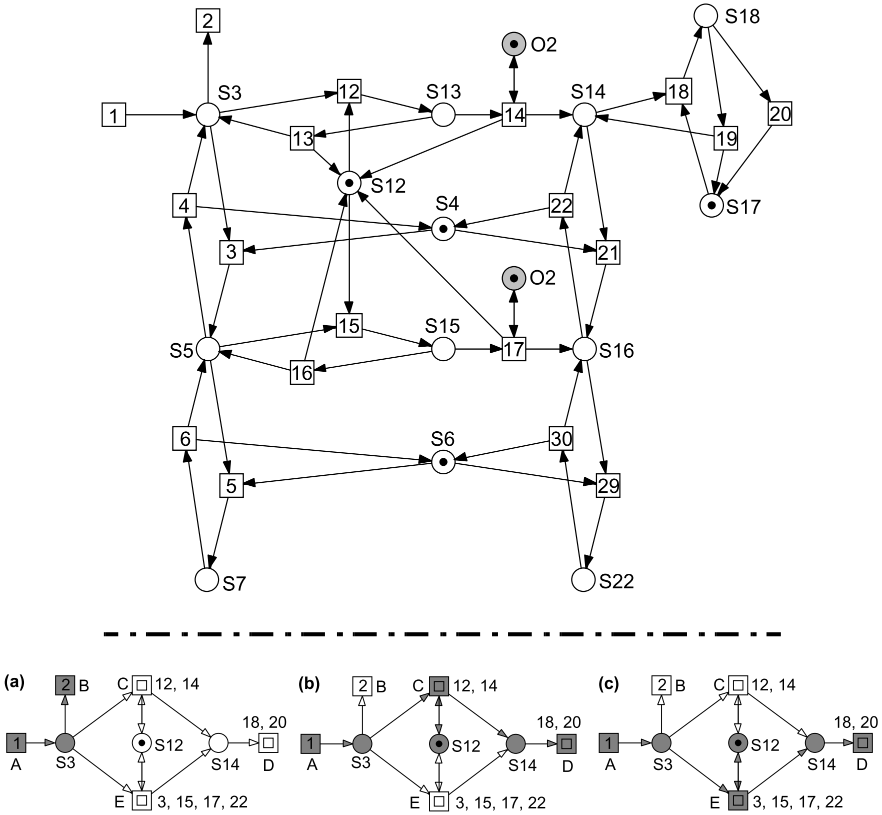


Fig. 6 The Petri net and its coarse structure (when neglecting the trivial T-invariants) for the hypoxia response network based on the ODEs given in [21] and [51]. The three pathways to degrade HIF (S3) highlighted in the coarse net are: (**a**) direct degradation by r2, (**b**) degradation not requiring S4, and (**c**) degradation requiring S4. The knock-out of S12 interrupts both (b) and (c)

The three non-trivial T-invariants are represented by the three macro transition sequences:

$$y_8 = (\mathrm{A}; \mathrm{B}),$$

$$y_9 = (\mathrm{A}; \mathrm{C}; \mathrm{D}),$$

$$y_{10} = (\mathrm{A}; \mathrm{E}; \mathrm{D}).$$

7 Tools

The case studies have been done using Snoopy [18, 45]—a tool to design and animate or simulate hierarchical graphs, among them the qualitative Petri nets as used in this paper. Snoopy provides export to various analysis tools as well as import and export of the Systems Biology Markup Language (SBML) [13].

The T-invariants, ADT sets and their decomposition into connected subnets have been computed with the Petri net analysis tool Charlie [3]. To support result evaluation, node sets, as specified by T-invariants or ADT sets, can be visualized (colored) in Snoopy.

The automatic derivation of the hierarchical Petri net showing the coarse net structure is subject of a running student's project.

The data files of the case studies and the analysis results are available at www-dssz.informatik.tu-cottbus.de/examples/coarsening.

8 Related Work

Please note, the following remarks are not meant to be exhaustive, but to give the interested reader some suggestions where to continue reading.

Petri nets, as we understand them today, have been initiated by concepts proposed by Carl Adam Petri in his Ph.D. thesis in 1962 [33]. The first substantial results making up the still growing body of Petri net theory appeared around 1970. Initial textbooks devoted to Petri nets were issued in the beginning of the 80s [34, 39, 47]. General introductions into Petri net theory can be found, for example, in [1, 6, 28, 48]. An excellent textbook for theoretical issues is [36]. The text [7] might be useful, if you just want to get the general flavor in reasonable time.

Petri nets have been deployed for technical and administrative systems in numerous application domains since the mid-70s. The deployment in systems biology has been first published in [17, 38, 40]. Recent surveys on applying Petri nets for biochemical networks are [2, 26], offering a rich choice of further reading pointers, among them numerous case studies applying various types of Petri nets to biochemical networks, comprising gene regulatory networks, signal transduction networks, metabolic networks, or combinations of them. The majority of these papers deal with one Petri net type only, mostly quantitative Petri nets such as stochastic, continuous, or hybrid Petri nets. A careful qualitative analysis of the combined glycolysis pentose phosphate pathway is exercised in [19]. A framework integrating qualitative, stochastic and continuous Petri nets into a step-wise modeling and analysis process is demonstrated by a running example each in [11, 12, 14].

P- and T-invariants are well-known concepts of Petri net theory since the very beginning [22]. There are corresponding notions in systems biology, called chemical moieties or conservation relations [25, 46], and elementary modes [43] or extreme pathways [44], which are elaborated in the setting of biochemical networks in [30]. In order to reduce the generating system of the solution space, generic pathways (minimal metabolic behavior) have been proposed, which are especially helpful, if there are plenty of reversible reactions [23, 24]. For biochemical systems without reversible reactions, the notions T-invariants, elementary modes, extreme pathways, and generic pathways coincide.

The efficient computation of invariants has been repeatedly examined; for some of the earlier papers, see, e.g. [5, 31, 49], for modular computational approaches, see [4, 32, 52].

Invariants have been applied for validation and verification of Petri net models in many ways. Invariant-based model validation of technical or administrative systems—especially in the context of P-invariants— is one of the standard Petri net techniques; their use to check model consistency is straightforward. The introductory textbooks [29, 39] give examples how P-invariants can be used in mathematical reasoning to prove certain model properties.

The model validation of biochemical networks by help of T-invariants is demonstrated in [15] by three case studies, comprising metabolic as well as signal transduction networks, one of them represented as colored Petri net. The comprehensive textbook [30] is focused on the stoichiometric matrix and related evaluation techniques of reconstructed biochemical networks. It is also a good entry point for the growing body of related literature in systems biology.

The partial order run of I/O T-invariants is considered in [9, 10] to gain deeper insights into the signal response behavior of signal transduction networks. T-invariants are used in [19] to derive adequate environment behavior, transforming an open system into a closed one, in [8] for the identification of functional modules by clustering techniques, and in [37] to obtain time constraints reflecting the steady state behavior.

Finally, a bit of history. The idea to decompose T-invariants into sub-T-invariants is rather intuitive and has already been used in an informal manner in [20] in order to support the validation process for a metabolic network of the potato tuber. The concept of maximal sets of dependent transitions has been introduced in [41] and implemented in Perl to validate the mating pheromone response pathway in *Saccharomyces cerevisiae*. These results are published in [42], which also gives a formal definition of the notion called Maximal Common Transition set (MCT-set), which corresponds to maximal abstract dependent transition sets as introduced in our paper. A generalization of MCT-sets is elaborated in [50], comprising also the foundation for the structuring approach presented in our paper. That is why we adopt the naming convention introduced there. The crucial point for our application scenario is that we generally need a further decomposition of maximal ADT sets into ADT sets inducing connected subnets, which are consequently not maximal anymore.

While writing this paper, and especially compiling this section, we became aware of the notions perfectly/partially/directionally *correlated reaction sets*, abbreviated by co-sets (which would cause confusion in the Petri net community). They are usually introduced verbally as well as by examples; see, e.g. [30]. However, partially correlated reaction sets seem to correspond to (maximal?) abstract dependent transition sets, and perfectly correlated reaction sets to (maximal?) dependent transition sets (not discussed in our paper; see [50] for details). The authors advocate correlated reaction sets for hierarchical thinking in network biology and the unbiased modularization of biochemical networks, and confirm our observation that these sets "can include non-obvious groups of reactions and differ from groupings of reactions based on a visual inspection of the network topology" [35]. There is no better way to conclude this section.

9 Summary

Petri nets provide a concise, executable, and formal modeling paradigm, allowing a unifying view on knowledge originating from different sources, which are usually represented there in various, sometimes even ambiguous styles. The derived models can be validated by checking T-invariants for biological interpretation.

We have presented a structuring technique contributing to a better understanding of the network behavior and requiring static analysis only. The state space is never constructed, thus the technique works even for systems with infinite state spaces, i.e. unbounded Petri nets.

The key notions are T-invariants and ADT sets. Minimal T-invariants induce always connected subnets, which generally overlap. Maximal ADT sets induce always subnets, overlapping in interface places only, but which are not necessarily connected. We determine a classification of the transitions into ADT sets, inducing connected subnets.

This classification defines a structural decomposition into subnets, which can be read as smallest biologically meaningful functional units. Connected ADT sets can be hidden in macro transitions. The derived coarse network provides a structured representation of the given set of minimal T-invariants, and may serve as a shorthand notation. This technique works equally for P-invariants.

The whole approach is algorithmically defined and does not require human interaction. However, the computation of all minimal T-invariants has to be accomplished first, and in the worst-case there are exponentially many of them.

The proposed structuring technique does not rely on the given interpretation of Petri nets. Nevertheless, it seems to be specifically helpful for analyzing biochemically interpreted Petri nets, where it supports the validation of models formalizing our current understanding of what has been created by natural evolution.

In this paper, we have focused on model validation by means of qualitative models, because it is obviously necessary to check at first a model for consistency and correctness of its biological interpretation before starting further analyses, aiming in the long-term at behavior prediction by means of quantitative models. The expected results—justifying the additional expense of preliminary model validation—consist in concise, formal and, therefore, unambiguous models, which are provably self-consistent and more likely to reflect adequately the modeled reality.

Acknowledgements The idea for this paper was born during my 7-month research stay at INRIA Paris-Rocquencourt while on sabbatical leave from my home university in 2007. My thanks go to the project team Constraints for hosting me. In particular, I would like to thank K. Sriram for drawing my attention to the hypoxia example and for the numerous rewarding discussions.

Special thanks to Professor Grzegorz Rozenberg: His professional life has been both inspiring and encouraging to me as to so many others. This paper is an outcome of his faith in the potential of biochemically interpreted Petri nets.

References

1. Bause F, Kritzinger PS (2002) Stochastic Petri nets. Vieweg, Wiesbaden

2. Chaouiya C (2007) Petri net modelling of biological networks. Brief Bioinform 8(4):210–219
3. Charlie Website (2008) A tool for the analysis of place/transition nets. BTU Cottbus. http://www-dssz.informatik.tu-cottbus.de/software/charlie/charlie.html
4. Christensen S, Petrucci L (2000) Modular analysis of Petri nets. Comput J 43(3):224–242
5. Colom JM, Silva M (1991) Convex geometry and semiflows in P/T nets. In: A comparative study of algorithms for computation of minimal P-semiflows. LNCS, vol 483. Springer, Berlin, pp 79–112
6. David R, Alla H (2005) Discrete, continuous, and hybrid Petri nets. Springer, Berlin
7. Desel J, Juhás G (2001) What is a Petri net? In: Unifying Petri nets—advances in Petri nets, Tokyo, 2001. LNCS, vol 2128. Springer, Berlin, pp 1–25
8. Grafahrend-Belau E, Schreiber F, Heiner M, Sackmann A, Junker B, Grunwald S, Speer A, Winder K, Koch I (2008) Modularization of biochemical networks based on classification of Petri net T-invariants. BMC Bioinform 9:90
9. Gilbert D, Heiner M (2006) From Petri nets to differential equations—an integrative approach for biochemical network analysis. In: Proceedings of the ICATPN 2006. LNCS, vol 4024. Springer, Berlin, pp 181–200
10. Gilbert D, Heiner M, Lehrack S (2007) A unifying framework for modelling and analysing biochemical pathways using Petri nets. In: Proceedings of the CMSB 2007. LNCS/LNBI, vol 4695. Springer, Berlin, pp 200–216
11. Gilbert D, Heiner M, Rosser S, Fulton R, Gu X, Trybiło M (2008) A case study in model-driven synthetic biology. In: Proceedings of the 2nd IFIP conference on biologically inspired collaborative computing (BICC), IFIP WCC 2008, Milano, pp 163–175
12. Heiner M, Donaldson R, Gilbert D (2010) Petri nets for systems biology. In: Iyengar MS (ed) Symbolic systems biology: theory and methods. Jones and Bartlett, Boston (to appear)
13. Hucka M, Finney A, Sauro HM, Bolouri H, Doyle JC, Kitano H et al (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. J Bioinform 19:524–531
14. Heiner M, Gilbert D, Donaldson R (2008) Petri nets in systems and synthetic biology. In: Schools on formal methods (SFM). LNCS, vol 5016. Springer, Berlin, pp 215–264
15. Heiner M, Koch I (2004) Petri net based model validation in systems biology. In: Proceedings of the 25th ICATPN 2004. LNCS, vol 3099. Springer, Berlin, pp 216–237
16. Heiner M, Koch I, Will J (2004) Model validation of biological pathways using Petri nets—demonstrated for apoptosis. Biosystems 75:15–28
17. Hofestädt R (1994) A Petri net application of metabolic processes. J Syst Anal Model Simul 16:113–122
18. Heiner M, Richter R, Schwarick M (2008) Snoopy—a tool to design and animate/simulate graph-based formalisms. In: Proceedings of the PNTAP 2008, associated to SIMUTools 2008. ACM digital library
19. Koch I, Heiner M (2008) Petri nets. In: Junker BH, Schreiber F (eds) Biological network analysis. Book series on bioinformatics. Wiley, New York, pp 139–179
20. Koch I, Junker BH, Heiner M (2005) Application of Petri net theory for modeling and validation of the sucrose breakdown pathway in the potato tuber. Bioinformatics 21(7):1219–1226
21. Kohn KW, Riss J, Aprelikova O, Weinstein JN, Pommier Y, Barrett JC (2004) Properties of switch-like bioregulatory networks studied by simulation of the hypoxia response control system. Mol Cell Biol 15:3042–3052
22. Lautenbach K (1973) Exact liveness conditions of a Petri net class. GMD Report 82, Bonn (in German)
23. Larhlimi A, Bockmayr A (2005) Minimal metabolic behaviors and the reversible metabolic space. Preprint No 299, FU Berlin, DFG-Research Center Matheon
24. Larhlimi A, Bockmayr A (2008) On inner and outer descriptions of the steady-state flux cone of a metabolic network. In: Proceedings of the CMSB 2008. LNCS/LNBI, vol 5307. Springer, Berlin, pp 308–327
25. Mendes P (1993) GEPASI: a software package for modelling the dynamics, steady states and control of biochemical and other systems. Comput Appl Biosci 9:563–571

26. Matsuno H, Li C, Miyano S (2006) Petri net based descriptions for systematic understanding of biological pathways. IEICE Trans Fundam Electron Commun Comput Sci E89-A(11):3166–3174
27. Matsuno H, Tanaka Y, Aoshima H, Doi A, Matsui M, Miyano S (2003) Biopathways representation and simulation on hybrid functional Petri net. In: Silico Biol 3(0032)
28. Murata T (1989) Petri nets: properties, analysis and applications. Proc IEEE 77 4:541–580
29. Pagnoni A (1990) Project engineering: computer-oriented planning and operational decision making. Springer, Berlin
30. Palsson BO (2006) Systems biology: properties of reconstructed networks. Cambridge University Press, Cambridge
31. Pascoletti KH (1986) Diophantine systems and solution methods to determine all Petri nets invariants. GMD Report 160, Bonn (in German)
32. Pedersen M (2008) Compositional definitions of minimal flows in Petri nets. In: Proceedings of the CMSB 2008. LNCS/LNBI, vol 5307. Springer, Berlin, pp 288–307
33. Petri CA (1962) Communication with Automata (in German). Schriften des Instituts für Instrumentelle Mathematik, Bonn
34. Peterson JL (1981) Petri net theory and the modeling of systems. Prentice–Hall, New York
35. Papin JA, Reed JL, Palsson PO (2004) Hierarchical thinking in network biology: the unbiased modularization of biochemical networks. Trends Biochem Sci 29(12):641–647
36. Priese L, Wimmel H (2003) Theoretical informatics—Petri nets. Springer, Berlin (in German)
37. Popova-Zeugmann L, Heiner M, Koch I (2005) Time Petri nets for modelling and analysis of biochemical networks. Fundam Inform 67:149–162
38. Reddy VN (1994) Modeling biological pathways: a discrete event systems approach. Master thesis, University of Maryland
39. Reisig W (1982) Petri nets; an introduction. Springer, Berlin
40. Reddy VN, Mavrovouniotis ML, Liebman ML (1993) Petri net representations in metabolic pathways. In Proceedings of the international conference on intelligent systems for molecular biology
41. Sackmann A (2005) Modelling and simulation of signal transduction pathways in saccharomyces cerevisiae using Petri net theory. Diploma thesis, Ernst Moritz Arndt Univ Greifswald (in German)
42. Sackmann A, Heiner M, Koch I (2006) Application of Petri net based analysis techniques to signal transduction pathways. BMC Bioinform 7:482
43. Schuster S, Hilgetag C, Schuster R (1993) Determining elementary modes of functioning in biochemical reaction networks at steady state. In Proceedings of the second Gauss symposium, pp 101–114
44. Schilling CH, Letscher D, Palsson BO (2000) Theory for the systemic definition of metabolic pathways and their use in interpreting metabolic function from a pathway-oriented perspective. Theor Biol 203:229–248
45. Snoopy website (2008) A tool to design and animate/simulate graphs. BTU Cottbus. http://www-dssz.informatik.tu-cottbus.de/software/snoopy.html
46. Schuster S, Pfeiffer T, Moldenhauer F, Koch I, Dandekar T (2002) Exploring the pathway structure of metabolism: decomposition into subnetworks and application to mycoplasma pneumoniae. BioInformatics 18(2):351–361
47. Starke PH (1980) Petri nets: foundations, applications, theory. VEB Deutscher Verlag der Wissenschaften, Berlin (in German)
48. Starke PH (1990) Analysis of Petri net models. Teubner, Stuttgart (in German)
49. Toudic JM (1982) Linear algebra algorithms for the structural analysis of Petri nets. Rev Tech Thomson CSF (France) 14(1):137–156 (in French)
50. Winder K (2006) Invariant-based structural characterization of Petri nets. Diploma thesis. BTU Cottbus, Dep of CS (in German)
51. Yu Y, Wang G, Simha R, Peng W, Turano F, Zeng C (2007) Pathway switching explains the sharp response characteristic of hypoxia response network system. PLos Comput Biol 8(3):1657–1668
52. Zaitsev DA (2005) Functional Petri nets. TR 224, CNRS

Quantitative Verification Techniques for Biological Processes

Marta Kwiatkowska, Gethin Norman, and David Parker

Abstract Probabilistic model checking is a formal verification framework for systems which exhibit stochastic behavior. It has been successfully applied to a wide range of domains, including security and communication protocols, distributed algorithms and power management. In this chapter, we demonstrate its applicability to the analysis of biological pathways and show how it can yield a better understanding of the dynamics of these systems. Through a case study of the Mitogen-Activated Protein (MAP), Kinase cascade, we explain how biological pathways can be modeled in the probabilistic model checker PRISM and how this enables the analysis of a rich selection of quantitative properties.

1 Introduction

Recent research has had considerable success adapting approaches from computer science to the analysis of biological systems and, in particular, biochemical pathways. The fundamental theory behind the majority of this work is the simulation-based techniques for discrete stochastic models originally introduced by Gillespie [9]. This models the evolution of individual molecules, whose rates of interaction are controlled by exponential distributions, and differs from the principal alternative modeling paradigm of pathways, using ordinary differential equations to model the evolution of average molecular concentrations over time. We adopt the stochastic modeling approach but, by employing formal verification techniques, compute *exact* quantitative measures as opposed to taking averages over sets of simulation runs.

In this chapter, we demonstrate how probabilistic model checking [2, 20, 32] and the probabilistic model checker PRISM [14, 27] can be employed as a framework for the modeling and analysis of biological pathways. This approach is motivated by both the fact that PRISM has already been successfully applied to the study of biological pathways; see, for example [4, 11, 30], and previous work which has demonstrated the applicability of probabilistic model checking to the analysis of a wide variety of complex stochastic systems; see, for example [18].

M. Kwiatkowska (✉)
Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, UK
e-mail: marta.kwiatkowska@comlab.ox.ac.uk

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_20, © Springer-Verlag Berlin Heidelberg 2009

This framework inherits many of the advantages of model checking, including the use of a both a formal model and specification of the system under study and the fact that the approach is exhaustive, analysing all possible behaviors of the system. We are also able to re-use existing technology, exploiting the efficient implementations and tool support developed for probabilistic model checkers such as PRISM. The intention is that probabilistic model checking should be used in conjunction with other, well-established approaches for analysing pathways based on simulation and differential equations. In combination, these techniques can offer greater insight into the complex interactions present in biological pathways.

Outline

In the next section, we give an overview of probabilistic model checking and the tool PRISM. Section 3 presents the MAPK cascade, discusses how the pathway can be modeled in the PRISM language, and demonstrates how PRISM can be used to specify and analyse a wide range of quantitative properties. In Sect. 4, we discuss related work and Sect. 5 concludes the chapter.

2 Probabilistic Model Checking

Probabilistic model checking is a formal verification technique for the modeling and analysis of systems that exhibit stochastic behavior. This technique is a variant of *model checking*, a well established and widely-used formal method for ascertaining the correctness of real-life systems. Model checking requires two inputs:

- a description of the system, usually given in some high-level modeling formalism such as a Petri net or process algebraic expression;
- a specification of one or more desired properties of the system, normally using temporal logics such as Computation Tree Logic CTL) or Linear-time Temporal Logic (LTL).

From these inputs, a *model checker* can construct a model of the system, typically a labeled state-transition system in which each state represents a possible configuration and each transition represents an evolution of the system from one configuration to another over time. It is then possible to automatically verify whether or not each property is satisfied, based on a systematic and exhaustive exploration of the constructed state-transition system.

In probabilistic model checking, the models are augmented with quantitative information regarding the likelihood that transitions occur and the times at which they do so. In practice, these models are typically Markov chains or Markov decision processes. To model biological pathways, the appropriate model is *continuous-time Markov chains* (CTMCs), in which transitions between states are assigned (positive, real-valued) rates. These values are interpreted as the rates of negative exponential distributions.

Formally, letting $\mathbb{R}_{\geq 0}$ denote the set of non-negative reals and AP be a fixed, finite set of atomic propositions used to label states with properties of interest, a CTMC is a tuple $(S, \vec{R}, L)$ where:

- S is a finite set of *states*;
- $\vec{R} : (S \times S) \rightarrow \mathbb{R}_{\geq 0}$ is a *transition rate matrix*;
- $L : S \rightarrow 2^{AP}$ is a *labeling* function which associates each state with a set of atomic propositions.

The transition rate matrix $\vec{R}$ assigns rates to each pair of states, which are used as parameters of the exponential distribution. A transition can only occur between states s and s' if $\vec{R}(s, s') > 0$ and, in this case, the probability of the transition being triggered within t time-units equals $1 - \exp(-\vec{R}(s, s') \times t)$. Typically, in a state s, there is more than one state s' for which $\vec{R}(s, s') > 0$; this is known as a *race condition* and the first transition to be triggered determines the next state. The time spent in state s before any such transition occurs is exponentially distributed with the rate $E(s) = \sum_{s' \in S} \vec{R}(s, s')$, called the *exit rate* of stat s. The probability of moving to state s' is given by $\vec{R}(s, s')/E(s)$.

A CTMC can be augmented with *rewards*, attached to states and/or transitions of the model. Formally, a *reward structure* for a CTMC is a pair $(\vec{\rho}, \vec{\iota})$ where:

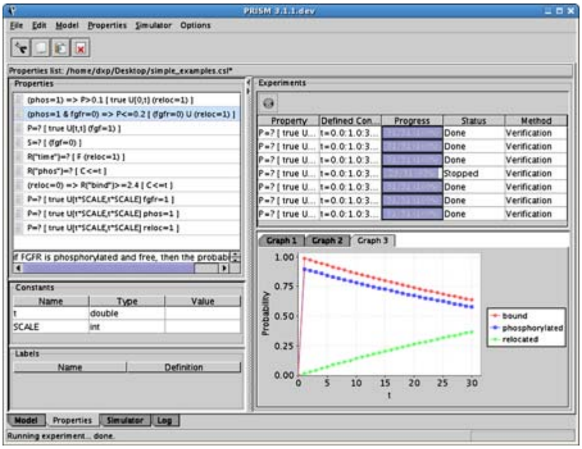
- $\vec{\rho} : S \rightarrow \mathbb{R}_{\geq 0}$ is a *state reward function*;
- $\vec{\iota} : (S \times S) \rightarrow \mathbb{R}_{\geq 0}$ is a *transition reward function*.

State rewards can represent either a quantitative measure of interest at a particular time instant (e.g. the number of phosphorylated proteins in the system) or the rate at which some measure accumulates over time (e.g. energy dissipation). Transition rewards are accumulated each time a transition occurs and can be used to compute, e.g. the number of protein bindings over a particular time period.

Properties of CTMCs are, like in non-probabilistic model checking, expressed in temporal logic, but are now quantitative in nature. For this, we use probabilistic temporal logics such as CSL [1, 2] and its extensions for reward-based properties [20]. For example, rather than verifying that 'the protein always eventually degrades', using CSL allows us to ask 'what is the probability that the protein eventually degrades' or 'what is the probability that the protein degrades within t hours?' Reward-based properties include 'what is the expected number of phosphorylations within the first t time units?' and 'what is the expected time that proteins spend bound before relocation occurs?' For further details on probabilistic model checking of CTMCs, see, for example [2, 20, 32].

PRISM [14, 27] is a probabilistic model checking tool developed at the Universities of Birmingham and Oxford. It provides support for several types of probabilistic models, including CTMCs. Models are specified in a simple, state-based language based on guarded commands. PRISM's notation for specifying properties of CTMCs incorporates the reward-based extension ([20]) of CSL. Figure 1 shows a screenshot of PRISM in action.

Fig. 1 A screenshot of the PRISM graphical user interface



Properties list: /home/dxp/Desktop/simple_examples.csl*

```
(phos=1) => P>0.1 [ true U[0,t] (reloc=1) ]
(phos=1 & fgfr=0) => P<=0.2 [ (fgfr=0) U (reloc=1) ]
P=? [ true U[t,t] (fgf=1) ]
S=? [ (fgf=0) ]
R("time")=? [ F (reloc=1) ]
R("phos")=? [ C<=t ]
(reloc=0) => R("bind")>=2.4 [ C<=t ]
P=? [ true U[t*SCALE,t*SCALE] fgfr=1 ]
P=? [ true U[t*SCALE,t*SCALE] phos=1 ]
P=? [ true U[t*SCALE,t*SCALE] reloc=1 ]
```

if FGFR is phosphorylated and free, then the probab

Property	Defined Con...	Progress	Status	Method
P=? [true U...	t=0.0:1.0:3...		Done	Verification
P=? [true U...	t=0.0:1.0:3...		Done	Verification
P=? [true U...	t=0.0:1.0:3...		Done	Verification
P=? [true U...	t=0.0:1.0:3...		Stopped	Verification
P=? [true U...	t=0.0:1.0:3...		Done	Verification
P=? [true U...	t=0.0:1.0:3...		Done	Verification
P=? [true U...	t=0.0:1.0:3...		Done	Verification

Name	Type	Value
t	double	
SCALE	int	

Name	Definition

Running experiment... done.

The underlying computation in PRISM involves a combination of:

- *graph-theoretical algorithms*, for conventional temporal logic model checking and *qualitative* probabilistic model checking;
- *numerical computation*, for *quantitative* probabilistic model checking, i.e. calculation of probabilities and reward values.

Graph-theoretical algorithms are comparable to the operation of a conventional, non-probabilistic model checker. For numerical computation, PRISM typically solves linear equation systems or performs transient analysis. Due to the size of the models that need to be handled, the tool uses iterative methods rather than direct methods. For solution of linear equation systems, it supports a range of well-known techniques including the Jacobi, Gauss–Seidel, and successive over-relaxation (SOR) methods; for transient analysis of CTMCs, it employs uniformization.

One of the most notable features of PRISM is that it uses state-of-the-art *symbolic* approaches, using data structures based on binary decision diagrams [17, 25]. These allow for compact representation and efficient manipulation of large, structured models by exploiting regularities exhibited in the high-level modeling language descriptions. The tool actually provides three distinct *engines* for numerical solution: the first is purely symbolic; the second uses sparse matrices; and the third is a hybrid, using a combination of the two. The result is a flexible implementation which can be adjusted to improve performance depending on the type of models and properties being analysed.

PRISM also incorporates a discrete-event simulation engine. This allows approximate solutions to be generated for the numerical computations that underlie the model checking process, by applying Monte Carlo methods and sampling. These techniques offer increased scalability, at the expense of numerical accuracy. Using the same underlying engine, PRISM includes a tool to perform manual execution and debugging of models. Other functionality provided by the user interface of the tool includes a graph-plotting component for visualization of numerical results and editors for the model and property specification languages.

3 Case Study: MAPK Cascade

We demonstrate the application of probabilistic model checking to the modeling, specification and analysis of biological pathways through a case study: the MAPK cascade.

The Mitogen-Activated Protein (MAP) Kinases are involved in a pathway through which information is sent to the nucleus. It is one of the most important signaling pathways, playing a pivotal role in the molecular signaling that governs the growth, proliferation, and survival of many cell types. The MAPK cascade consists of a MAPK Kinase Kinase (MAPKKK), a MAPK Kinase (MAPKK), and a MAPK. The cascade is initialized through the phosphorylation of MAPKKK, which then activates MAPKK through phosphorylation at two serine residues. This then activates MAPK through phosphorylation at theronine and tyrosine residues. The initialization of the pathway can be caused by a diverse set of stimuli including growth factors, neurotransmitters and cytokines.

Figure 2 gives an overview of the structure of the pathway and Fig. 3 details the reactions that form the cascade, as taken from [15]. In the reactions presented in Fig. 3, it is assumed that the phosphorylation of both MAPK and MAPKK occur in two distributed steps. For example, when MAPK collides with its activator (MAPKK-PP) the first phosphorylation (MAPK-P) occurs and the activator is released. The phosphorylated MAPK must then collide again with its activator for the second phosphorylation (MAPK-PP) to occur. The deactivation of phosphorylated MAPK and MAPKK is caused by the corresponding phosphatase, while the activation and deactivation of MAPKKK is through the enzymes E1 and E2, respectively. To simplify the presentation in Fig. 3, we denote MAPK, MAPKK, and MAPKKK by K, KK, and KKK, respectively.

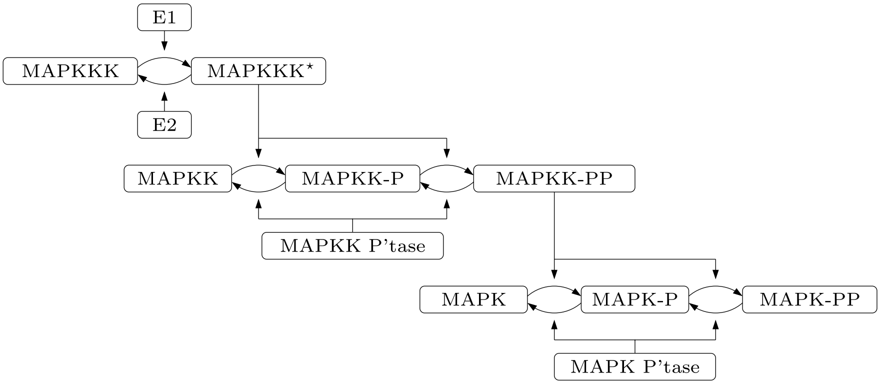


Fig. 2 MAPK cascade pathway

1. MAPKKK is activated through enzyme E1	
$\mathrm{KKK} + \mathrm{E1} \rightarrow \mathrm{KKK{:}E1}$	$a_1=1\,\mathrm{nM}^{-1}\,\mathrm{s}^{-1}$
$\mathrm{KKK} + \mathrm{E1} \leftarrow \mathrm{KKK{:}E1}$	$d_1=150\,\mathrm{s}^{-1}$
$\mathrm{KKK{:}E1} \rightarrow \mathrm{KKK}^\star + \mathrm{E1}$	$k_1=150\,\mathrm{s}^{-1}$
2. MAPKKK is deactivated through enzyme E2	
$\mathrm{KKK}^\star + \mathrm{E2} \rightarrow \mathrm{KKK}^\star\mathrm{{:}E2}$	$a_2=1\,\mathrm{nM}^{-1}\,\mathrm{s}^{-1}$
$\mathrm{KKK}^\star + \mathrm{E2} \leftarrow \mathrm{KKK}^\star\mathrm{{:}E2}$	$d_2=150\,\mathrm{s}^{-1}$
$\mathrm{KKK}^\star\mathrm{{:}E2} \rightarrow \mathrm{KKK} + \mathrm{E2}$	$k_2=150\,\mathrm{s}^{-1}$
3. MAPKK is activated by MAPKKK*	
$\mathrm{KK} + \mathrm{KKK}^\star \rightarrow \mathrm{KK{:}KKK}^\star$	$a_3=1\,\mathrm{nM}^{-1}\,\mathrm{s}^{-1}$
$\mathrm{KK} + \mathrm{KKK}^\star \leftarrow \mathrm{KK{:}KKK}^\star$	$d_3=150\,\mathrm{s}^{-1}$
$\mathrm{KK{:}KKK}^\star \rightarrow \mathrm{KK\text{-}P} + \mathrm{KKK}^\star$	$k_3=150\,\mathrm{s}^{-1}$
4. MAPKK-P is deactivated by MAPKK phosphatase	
$\mathrm{KK\text{-}P} + \mathrm{KK\text{-}Ptase} \rightarrow \mathrm{KK\text{-}P{:}KK\text{-}Ptase}$	$a_4=1\,\mathrm{nM}^{-1}\,\mathrm{s}^{-1}$
$\mathrm{KK\text{-}P} + \mathrm{KK\text{-}Ptase} \leftarrow \mathrm{KK\text{-}P{:}KK\text{-}Ptase}$	$d_4=150\,\mathrm{s}^{-1}$
$\mathrm{KK\text{-}P{:}KK\text{-}Ptase} \rightarrow \mathrm{KK} + \mathrm{KK\text{-}Ptase}$	$k_4=150\,\mathrm{s}^{-1}$
5. MAPKK-P is activated by MAPKKK*	
$\mathrm{KK\text{-}P} + \mathrm{KKK}^\star \rightarrow \mathrm{KK\text{-}P{:}KKK}^\star$	$a_5=1\,\mathrm{nM}^{-1}\,\mathrm{s}^{-1}$
$\mathrm{KK\text{-}P} + \mathrm{KKK}^\star \leftarrow \mathrm{KK\text{-}P{:}KKK}^\star$	$d_5=150\,\mathrm{s}^{-1}$
$\mathrm{KK\text{-}P{:}KKK}^\star \rightarrow \mathrm{KK\text{-}PP} + \mathrm{KKK}^\star$	$k_5=150\,\mathrm{s}^{-1}$
6. MAPKK-PP is deactivated by MAPKK phosphatase	
$\mathrm{KK\text{-}PP} + \mathrm{KK\text{-}Ptase} \rightarrow \mathrm{KK\text{-}PP{:}KK\text{-}Ptase}$	$a_6=1\,\mathrm{nM}^{-1}\,\mathrm{s}^{-1}$
$\mathrm{KK\text{-}PP} + \mathrm{KK\text{-}Ptase} \leftarrow \mathrm{KK\text{-}PP{:}KK\text{-}Ptase}$	$d_6=150\,\mathrm{s}^{-1}$
$\mathrm{KK\text{-}PP{:}KK\text{-}Ptase} \rightarrow \mathrm{KK\text{-}P} + \mathrm{KK\text{-}Ptase}$	$k_6=150\,\mathrm{s}^{-1}$
7. MAPK is activated by MAPKK-PP	
$\mathrm{K} + \mathrm{KK\text{-}PP} \rightarrow \mathrm{K{:}KK\text{-}PP}$	$a_7=1\,\mathrm{nM}^{-1}\,\mathrm{s}^{-1}$
$\mathrm{K} + \mathrm{KK\text{-}PP} \leftarrow \mathrm{K{:}KK\text{-}PP}$	$d_7=150\,\mathrm{s}^{-1}$
$\mathrm{K{:}KK\text{-}PP} \rightarrow \mathrm{K\text{-}P} + \mathrm{KK\text{-}PP}$	$k_7=150\,\mathrm{s}^{-1}$
8. MAPK-P is deactivated by MAPK phosphatase	
$\mathrm{K\text{-}P} + \mathrm{K\text{-}Ptase} \rightarrow \mathrm{K\text{-}P{:}K\text{-}Ptase}$	$a_8=1\,\mathrm{nM}^{-1}\,\mathrm{s}^{-1}$
$\mathrm{K\text{-}P} + \mathrm{K\text{-}Ptase} \leftarrow \mathrm{K\text{-}P{:}K\text{-}Ptase}$	$d_8=150\,\mathrm{s}^{-1}$
$\mathrm{K\text{-}P{:}K\text{-}Ptase} \rightarrow \mathrm{K} + \mathrm{K\text{-}Ptase}$	$k_8=150\,\mathrm{s}^{-1}$
9. MAPK-P is activated by MAPKK-PP	
$\mathrm{K\text{-}P} + \mathrm{KK\text{-}PP} \rightarrow \mathrm{K\text{-}P{:}KK\text{-}PP}$	$a_9=1\,\mathrm{nM}^{-1}\,\mathrm{s}^{-1}$
$\mathrm{K\text{-}P} + \mathrm{KK\text{-}PP} \leftarrow \mathrm{K\text{-}P{:}KK\text{-}PP}$	$d_9=150\,\mathrm{s}^{-1}$
$\mathrm{K\text{-}P{:}KK\text{-}PP} \rightarrow \mathrm{K\text{-}PP} + \mathrm{KK\text{-}PP}$	$k_9=150\,\mathrm{s}^{-1}$
10. MAPK-PP is deactivated by MAPK phosphatase	
$\mathrm{K\text{-}PP} + \mathrm{K\text{-}Ptase} \rightarrow \mathrm{K\text{-}PP{:}K\text{-}Ptase}$	$a_{10}=1\,\mathrm{nM}^{-1}\,\mathrm{s}^{-1}$
$\mathrm{K\text{-}PP} + \mathrm{K\text{-}Ptase} \leftarrow \mathrm{K\text{-}PP{:}K\text{-}Ptase}$	$d_{10}=150\,\mathrm{s}^{-1}$
$\mathrm{K\text{-}PP{:}K\text{-}Ptase} \rightarrow \mathrm{K\text{-}P} + \mathrm{K\text{-}Ptase}$	$k_{10}=150\,\mathrm{s}^{-1}$

Fig. 3 MAPK cascade reactions

The kinetic rates given in Fig. 3 are based on the data presented in [15] where it is assumed that the K_m values ($K_m = (d_m + k_m)/a_m$) for phosphorylation and dephosphorylation of MAPK, MAPKK, and MAPKKK all equal 300 nM.

3.1 Specifying the Model

We now outline how to construct a *discrete stochastic* model of the MAPK cascade reactions from Fig. 3 in the modeling language of the PRISM tool. The applicability of probabilistic model checking and PRISM follows from the fact that the underlying model can be shown to be a CTMC, in which the stochastic rates associated with each transition can be derived from the kinetic rates of the reactions. In the case of unary reactions, the stochastic rate equals the kinetic rate. On the other hand, for binary reactions, if the kinetic rate is given in terms of molar concentrations, then the stochastic rate can be obtained by dividing by $\mathit{Vol} \times \mathcal{N}_A$ where *Vol* is the volume and $\mathcal{N}_A$ is Avogadro's number. For a more detailed discussion of the relationship between kinetic and stochastic rates, see, for example [9, 36].

A model described in the PRISM language comprises a set of *modules*, the state of each being represented by a valuation over a set of finite-ranging *variables*. The global state of the model is determined by a valuation over the union of all variables (denoted V). The atomic propositions of the model are given by predicates over the variables V and the labeling function assigns to each state the predicates that it satisfies.

The behavior of a module, i.e. the changes in state which it can undergo, is specified by a number of *guarded commands* of the form:

$$[\mathit{act}]\ \mathit{guard} \ \rightarrow \ \mathit{rate} \ : \ \mathit{update};$$

where *act* is an (optional) action label, *guard* is a predicate over the variables V, *rate* is a (non-negative) real-valued expression and *update* is of the form:

$$(x_1' = u_1) \ \& \ (x_2' = u_2) \ \& \ \cdots \ \& \ (x_n' = u_n)$$

where $u_1, u_2, \ldots, u_k$ are functions over V and $x_1, x_2, \ldots, x_n$ are variables of the module. Intuitively, in global state s (i.e. a valuation over the variables V) of the PRISM model, the command is enabled if s satisfies the predicate *guard*. If a command is enabled, a transition that updates the module's variables according to *update* (i.e. for $1 \leq i \leq n$ the variable x_i is updated to the value $u_i(s)$) can occur with rate *rate*. When multiple commands with the same update are enabled, the corresponding transitions are combined into a single transition whose rate is the sum of the individual rates.

To model interactions where the state of several modules changes simultaneously, we use *synchronization*, through the action labels that can be included in the guarded commands. The rate of the combined transition is defined as the product of the rates for each command. As we will see below, the rate of the combined transition is often fully specified in one module and rates omitted from the other modules (this yields the correct rate since PRISM assigns a rate of 1 to any command for which none is specified).

When building a PRISM model of a biological pathway, it is possible to construct an *individual-based* model which provides a detailed model of the evolution of individual molecular components. However, taking this approach comes at a cost: it will inevitably suffer from the well-known state-space explosion problem where, as the complexity of the system increases, the state space of the underlying model grows exponentially.

```
const int N; // initial amount of MAPK

// stochastic reaction rates
const double a7=1/N; const double d7=150; const double k7=150;
const double a8=1/N; const double d8=150; const double k8=150;
const double a9=1/N; const double d9=150; const double k9=150;
const double a10=1/N; const double d10=150; const double k10=150;

module MAPK

    k : [0..N] init N; // quantity of MAPK
    k_kkpp : [0..N] init 0; // quantity of MAPK:MAPKK-PP
    kp : [0..N] init 0; // quantity of MAPK-P
    kp_kkpp : [0..N] init 0; // quantity of MAPK-P:MAPKK-PP
    kp_ptase : [0..N] init 0; // quantity of MAPK-P:MAPK phosphatase
    kpp : [0..N] init 0; // quantity of MAPK-PP
    kpp_ptase : [0..N] init 0; // quantity of MAPK-PP:MAPK phosphatase

    // reaction 7 (MAPK is activated by MAPKK-PP)
    [a_k_kk] k>0 & k_kkpp<N
                    → a7 * k : (k_kkpp'=k_kkpp + 1) & (k'=k − 1);
    [d_k_kk] k<N & k_kkpp>0
                    → d7 * k_kkpp : (k_kkpp'=k_kkpp − 1) & (k'=k + 1);
    [k_k_kk] k_kkpp>0 & kp<N
                    → k7 * k_kkpp : (k_kkpp'=k_kkpp − 1) & (kp'=kp + 1);
    // reaction 8 (MAPK-P is deactivated by MAPK phosphatase)
    [a_k_ptase] kp>0 & kp_ptase<N
                    → a8 * kp : (kp_ptase'=kp_ptase + 1) & (kp'=kp − 1);
    [d_k_ptase] kp<N & kp_ptase>0
                    → d8 * kp_ptase : (kp_ptase'=kp_ptase − 1) & (kp'=kp + 1);
    [k_k_ptase] kp_ptase>0 & k<N
                    → k8 * kp_ptase : (kp_ptase'=kp_ptase − 1) & (k'=k + 1);
    // reaction 9 (MAPK-P is activated by MAPKK-PP)
    [a_k_kk] kp>0 & kp_kkpp<N
                    → a9 * kp : (kp_kkpp'=kp_kkpp + 1) & (kp'=kp − 1);
    [d_k_kk] kp<N & kp_kkpp>0
                    → d9 * kp_kkpp : (kp_kkpp'=kp_kkpp − 1) & (kp'=kp + 1);
    [k_k_kk] kp_kkpp>0 & kpp<N
                    → k9 * kp_kkpp : (kp_kkpp'=kp_kkpp − 1) & (kpp'=kpp + 1);
    // reaction 10 (MAPK-PP is deactivated by MAPK phosphatase)
    [a_k_ptase] kpp>0 & kpp_ptase<N
                    → a10 * kpp : (kpp_ptase'=kpp_ptase + 1) & (kpp'=kpp − 1);
    [d_k_ptase] kpp<N & kpp_ptase>0
                    → d10 * kpp_ptase : (kpp_ptase'=kpp_ptase − 1) & (kpp'=kpp + 1);
    [k_k_ptase] kpp_ptase>0 & kp<N
                    → k10 * kpp_ptase : (kpp_ptase'=kpp_ptase − 1) & (kp'=kp + 1);

endmodule
```

Fig. 4 PRISM module representing quantities of species relating to MAPK

An alternative is to employ a *population-based* approach where the number of each type of molecule or species is modeled, rather than the state of each individual component. Such an approach leads to a much smaller state-space (see, for example [11]) while still including sufficient detail to express the properties of interest. For these reasons, it is this approach that we use here.

For the PRISM language, a population-based model can be expressed naturally by using the variables of modules as counters, i.e. there is a variable for each of the

```
const int M; // initial amount of MAPK phosphatase

module KPTASE

    kptase : [0..M] init M; // amount of MAPK phosphatase

    // reactions 8 and 10 (MAPK/MAPK-P is deactivated by MAPK phosphatase)
    [a_k_ptase] kptase>0  →  kptase : (kptase'=kptase − 1);
    [d_k_ptase] kptase<M  →  1 : (kptase'=kptase + 1);
    [k_k_ptase] kptase<M  →  1 : (kptase'=kptase + 1);

endmodule
```

Fig. 5 PRISM module representing quantity of MAPK phosphatase

possible species in the system which keeps count of the number of that species that are currently present.

In Fig. 4, we present the module representing quantities of the species relating to MAPK and, in Fig. 5, the module representing MAPK phosphatase. The whole cascade could have been specified in one single large PRISM module. However, there is a natural separation of the different elements in the cascade (those relating to MAPKKK, MAPKK, MAPK, MAPKK phosphatase, MAPK phosphatase, E1 and E2) and defining the system using individual modules based on this separation makes the description simpler, easier to understand and less prone to modeling errors. This fact can be seen in other PRISM language models of biological pathways; see, for example [4, 11, 27]. The complete PRISM description of the MAPK cascade is available from the case study repository on the PRISM website [27].

As can be seen in Figs. 4 and 5, we have specified that there are initially N inactive MAPKs (the initial value of the variable k is N) and M MAPK phosphatases (the initial value of *kptase* is M). The actual values of N and M have been left undefined since, as will be seen later, this allows these parameters to be varied during model checking.

The values for stochastic reaction rates of the system are defined as constants (see the top of Fig. 4). Notice that the stochastic rates of the binary reactions (i.e. those specified by the constants *a7*, *a8*, *a9*, and *a10*) are obtained from the kinetic rates by dividing by the initial number of MAPKs (i.e. N). This is because (recall the discussion of computing stochastic reaction rates earlier in this section) we make the assumption that the volume of the system is proportional to the initial number of MAPKs. It would also have been possible to leave some of the constants for the stochastic rates unspecified and then vary these during verification.

Figures 4 and 5 also show that the modules for MAPK and MAPK phosphatase synchronize through the actions *a_k_ptase*, *d_k_ptase*, and *k_k_ptase*, which correspond to the deactivation of MAPK (as described in reactions 8 and 10 of Fig. 3). The actions *a_k_kk*, *d_k_kk*, and *k_k_kk*, which appear in the module for MAPK (Fig. 4), correspond to the activation of MAPK by MAPKK-PP (see reactions 7 and 9 of Fig. 3), and there are corresponding commands in the module for MAPKK.

When using a population-based approach, we must ensure that the rates of the CTMC take into account the different possible interactions that can occur. For example, if there are three activated MAPKs (k_pp_1, k_pp_2, and k_pp_3) and two MAPK phosphatases ($kptase_1$ and $kptase_2$) then there are six different species that can be formed: $k\_pp_1{:}kptase_1$, $k\_pp_1{:}kptase_2$, $k\_pp_2{:}kptase_1$, $k\_pp_2{:}kptase_2$, $k\_pp_3{:}kptase_1$ and $k\_pp_3{:}kptase_2$. The reaction rate is thus proportional to both the number of activated MAPKs and the number of MAPK phosphatases. This is straightforward to achieve in the PRISM modeling language since PRISM multiplies rates when modules synchronize: in this case, we set the rates to $a10 \times kpp$ and *kptase* in the modules *MAPK* (Fig. 4) and *KPTASE* (Fig. 5), respectively.

3.2 Specifying Rewards

Rewards are PRISM's mechanism for describing additional quantitative measures of probabilistic models. In this section, we explain how to specify reward structures for the PRISM model of the MAPK cascade presented in the previous section. Reward structures in PRISM are described using the construct:

`rewards` "*reward_name*" ... `endrewards`

comprising one or more state-reward items of the form:

guard : *reward*;

and/or transition-reward items of the form:

[*act*] *guard* : *reward*;

where *guard* is a predicate (over the variables V of the model), *act* is an action label appearing in the commands of the model and *reward* is a real-valued expression (which can contain variables and constants from the model). A state-reward item assigns a state reward of *reward* to all states satisfying *guard* and a transition-reward item assigns a transition reward of *reward* to all *act*-labeled transitions from states satisfying *guard*. Multiple rewards (from different reward items) for a single state or transition are summed and states or transitions with no assigned reward are assumed to have reward 0.

In Fig. 6, we present four different reward structures for the PRISM model of the cascade. The first reward structure ("*activated*") assigns a state reward equal to the amount of MAPK that is activated while the second reward structure ("*percentage*")

```
rewards "activated"

    true : kpp;

endrewards
```

```
rewards "percentage"

    true : 100*(kpp/N);

endrewards
```

```
rewards "reactions"
    [a_k_kk] true : 1;
    [d_k_kk] true : 1;
    [k_k_kk] true : 1;
endrewards
```

```
rewards "time"

    true : 1;

endrewards
```

Fig. 6 Reward structures for the cascade

assigns a state reward equal to the percentage of MAPK that is activated. These can be used to compute the expected amount/percentage of activated MAPK at some time instant or in the long run. The third reward structure "*reactions*" assigns a reward of 1 to all transitions which correspond to a reaction between MAPK and MAPKK. This can be used to compute the expected number of such reactions within a particular period of time or on average (in the long run). The final reward structure ("*time*") simply assigns a state reward of 1 to all states in the model which can be used, for example, to analyse the total expected time before an event/reaction occurs or a certain configuration is reached.

3.3 Specifying Properties

The temporal logic CSL, originally introduced by Aziz et al. [1] and since extended by Baier et al. [2], is based on the temporal logics CTL [5] and PCTL [10]. It provides a powerful means of specifying a variety of performance measures on CTMCs. PRISM use an extended version [20] which also allows for the specification of reward properties. We now give a number of examples of such specifications relating to the PRISM model and reward structures for the MAPK cascade presented in the previous sections. Recall that, in a PRISM model, atomic propositions are given by predicates over the variables of the model.

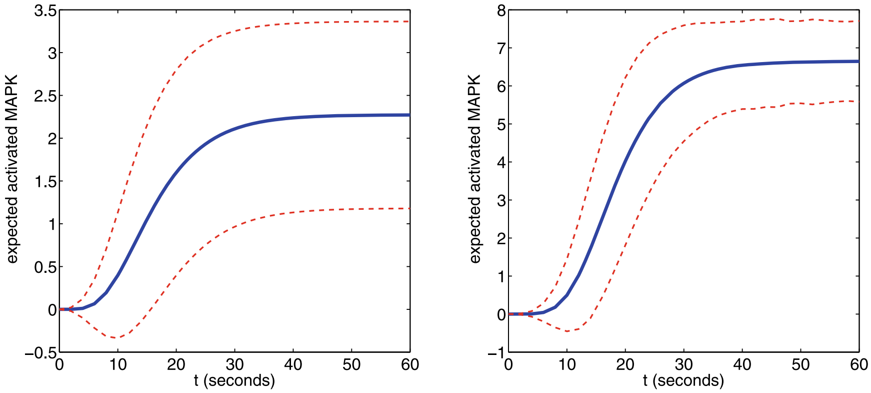
- $(kkpp = N \wedge kpp = 0) \rightarrow \mathcal{P}_{\geq 0.12}[(kkpp > 0)\mathcal{U}(kpp > 0)]$—if all MAPKKs are activated and none of the MAPKs are activated, then the probability that, while some MAPKKs remain activated, a MAPK becomes activated is at least 0.12.
- $\mathcal{P}_{=?}[\,\texttt{true}\,\mathcal{U}^{[t,t]}((kpp + kkpp) = l)]$—what is the probability that the total number of MAPKs and MAPKKs activated at time instant t equals l?
- $(kkkp > 0 \wedge kpp = 0) \rightarrow \mathcal{P}_{\leq 0.7}[(kpp = 0)\mathcal{U}^{[t_1,t_2]}(kpp > 0)]$—if some MAPKKKs are activated and no MAPKs are activated, then the probability that the first time a MAPK gets activated is within the time interval $[t_1, t_2]$ is at most 0.7.
- $(k = 0) \rightarrow \mathcal{P}_{\leq 0.01}[(k = 0)\mathcal{U}^{[t,\infty)}(k > 0)]$—if there are no inactive MAPKs, then the probability that some MAPK is deactivated for the first time after time t is at most 0.01.
- $\mathcal{S}_{=?}[(kpp = l)]$—what is the probability that in the long run there are precisely l MAPKs activated?
- $\mathcal{R}_{\{\text{“reactions”}\}=?}[\mathcal{C}^{\leq t}]$—what is the expected number of reactions between MAPKs and MAPKKs during the first t seconds?
- $(kpp = N) \rightarrow \mathcal{R}_{\{\text{“activated”}\}\geq N/2}[\mathcal{I}^{=t}]$—if all MAPKs are activated, then after t seconds the expected number of activated MAPK is at least half of the total number of MAPK.
- $\mathcal{R}_{\{\text{“reactions”}\}=?}[\mathcal{F}(kpp = N)]$—what is the expected number of reactions between MAPK and MAPKK before all MAPKs are activated at the same time instant?

- $(kpp > 0) \rightarrow \mathcal{R}_{\{\text{“time”}\}\leq 120}[\mathcal{F}\ (k = N)\ ]$—if some MAPKs are activated, the expected time until all of the MAPKs become deactivated at the same time instant at most 120 seconds.
- $\mathcal{R}_{\{\text{“percentage”}\}\geq 98}[\mathcal{S}]$—in the long run, at least 98% of MAPK is activated.
- $\mathcal{R}_{\{\text{“reactions”}\}=?}[\mathcal{S}]$—what is the long-run average rate of reactions between MAPK and MAPKK?

3.4 Results and Analysis

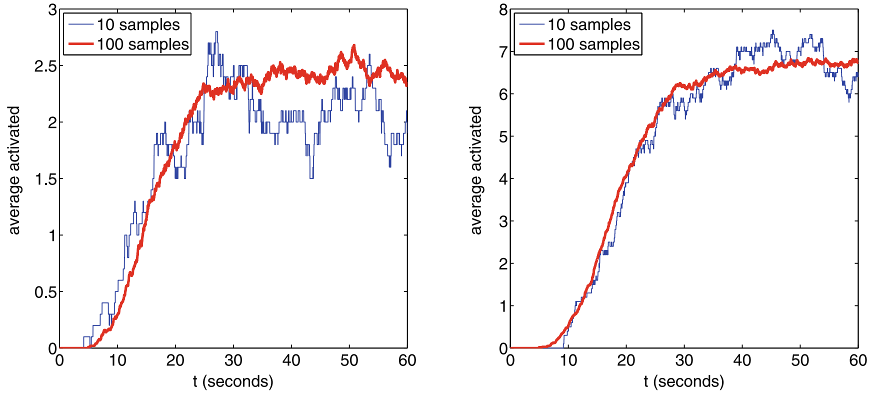
When analysing quantitative properties such as those listed above, it is often beneficial to study trends resulting from variations in parameters either from the model (e.g. initial species concentrations or reaction rates) or from the property specification (e.g. a time bound). Performing analysis in this way is more likely to provide insight into the dynamics of the model or to identify interesting or anomalous behavior.

To illustrate this, Fig. 7 shows results obtained with PRISM for the MAPK cascade case study when considering the expected amount of activated MAPK at time instant t, as t varies. The initial quantities of MAPK, MAPKK, and MAPKKK (denoted N) are 4 for Fig. 7(a) and 8 for Fig. 7(b). The initial quantity of all remaining species in the cascade (the enzymes E1 and E2 and the phosphatases for MAPK and MAPKK) is 1. The plots in Fig. 7 also show the standard deviation of the random variable for the amount of activated MAPK at time t, drawn as a pair of dotted lines. Since, the standard deviation of a random variable X equals the square root of its variance which equals $\mathbf{E}(X^2) - \mathbf{E}(X)^2$, the standard deviation (and variance) is calculated by additionally computing the expected value at time t for the reward



(a) $N = 4$ (b) $N = 8$

Fig. 7 Expected activated MAPK at time t ($\mathcal{R}_{\{\text{“activated”}\}=?}[\mathcal{I}^{=t}]$)



(a) $N = 4$

(b) $N = 8$

Fig. 8 Simulation results for amount of activated MAPK at time t

structure:

```
rewards "activated_squared"
    true : kpp * kpp;
endrewards
```

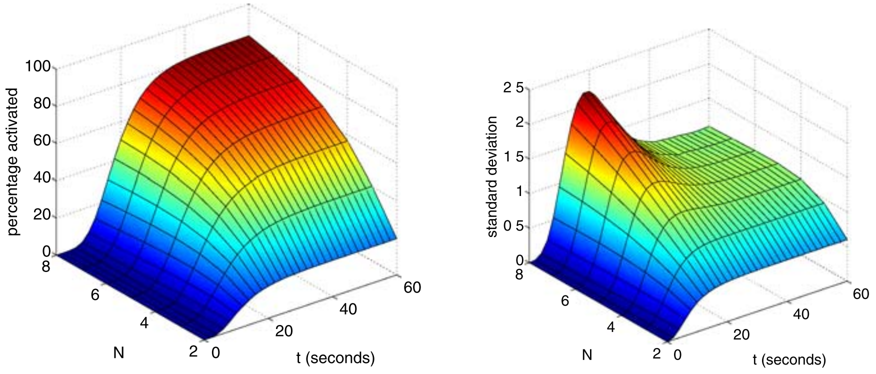
i.e. the square of the reward structure "*activated*" given in Fig. 6.

For the purposes of comparison, we also show results for the expected amount of activated MAPK computed using PRISM's discrete-event simulation engine. These results are presented in Fig. 8 (for the same initial configurations as those used in Fig. 7). These are generated using very small numbers of simulation runs (10 and 100). Smoother approximations for the plots from Fig. 7 can be obtained with higher numbers of runs.

Since it is also easy to change the initial amount N of MAPK, MAPKK, and MAPKKK in our model, we also show how the expected amount of activated MAPK over time varies for different values of N. Figure 9(a) shows the expected percentage of activated MAPK at time t for values of N from 2 up to 8, and Fig. 9(b) the standard deviation for the amount of MAPK over the same parameters.

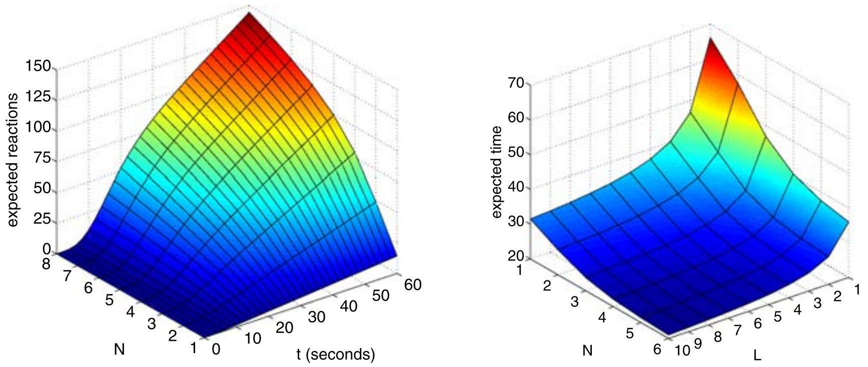
Using the other reward structures from Fig. 6, we also present results for the expected number of reactions between MAPK and MAPKK up until time t (Fig. 10(a)) and the expected time until all MAPKs are activated at the same time (Fig. 10(b)). In both cases, we vary the initial amount N of MAPK, MAPKK, and MAPKKK and, in Fig. 10(b), we also vary the initial quantity (denoted L) of the enzyme E1.

The results demonstrate that as N grows, the percentage of MAPK that is eventually activated increases and the time until all MAPKs are activated decreases. They also show the (expected) dynamics that raising species quantities increases the number of reactions that occur between them. We also observe that as N increases, the behavior of the PRISM model demonstrates the same behavior as that presented in [15] (computed through ODEs and the reactions given in Fig. 3) where, in response to an external stimulus (E1), the cascade acts as a switch for the activation of MAPK.



(a) Percentage (b) Standard devation

Fig. 9 Expected activated MAPK at time t and corresponding standard deviation values



(a) MAPK MAPKK reactions (b) Time until MAPK activated

Fig. 10 Expected MAPK–MAPK reactions by t and time until all MAPK activated

4 Related Work

In this section, we briefly review some other applications of probabilistic verification techniques to systems biology. We also describe the connections that exist between these approaches and the PRISM tool. Figure 11 illustrates the ways in which PRISM can interact with other tools and specification formalisms.

PRISM has been applied to a variety of biological case studies. In [11], it is used to study a model of the Fibroblast Growth Factor (FGF) signaling pathway. The model corresponds to a single instance of the pathway, i.e. there is at most one of each molecule or species, which has the advantage that the resulting state space is relatively small. However, the model is still highly complex due to the large number of different interactions that can occur in the pathway and is sufficiently rich to explain the roles of each component and how they interact. In [4], PRISM is used to

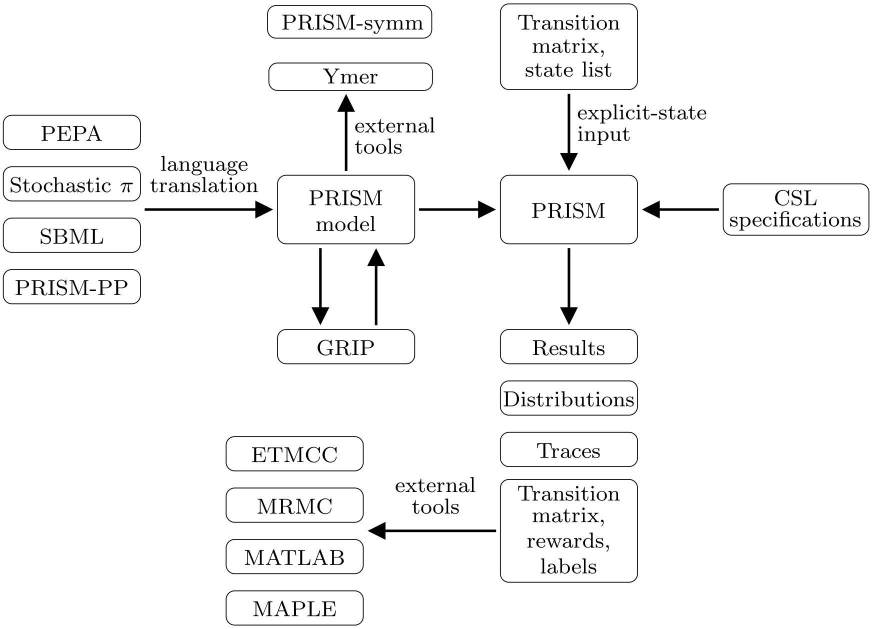


Fig. 11 Language and tool connections for PRISM

model the RKIP-inhibited ERK pathway where concentrations of each protein are modeled as discrete abstract quantities. Through comparisons with simulations for a traditional differential equation model, the authors show that accurate results can be obtained with relatively small sets of discrete values. PRISM is used in [30] to model codon bias, studying a range of quantitative properties of the system. Finally, [31] uses PRISM, in combination with several other tools, to analyse gene expression modeled using P-Systems.

Another formalism that has proved popular for modeling biological systems is *stochastic process algebra*. For example, PEPA [13] is used in [3] to study the effect of RKIP on the ERK signaling pathway. The stochastic π-calculus [28], an extension of the π-calculus with CTMC semantics, has been used to model many systems; see, for example [21, 29]. Various tools for construction and verification of PEPA models are available and, for the stochastic π-calculus, simulators such as BioSpi [29] and the Stochastic Pi-Machine (SPiM) [26] have been developed, but no model checkers. Both formalisms can also be used in conjunction with PRISM, through language translators. The PEPA translator is part of PRISM [27] and a prototype stochastic π-calculus translator has been built based on the techniques in [24].

An alternative format for representing biological models is Systems Biology Markup Language (SBML) [34], a computer-readable language based on XML. This is intended to facilitate exchanging models between different systems biology software tools. Biochemical reaction networks are described by specifying the set of

Fig. 12 Fragment of SBML-shorthand code for the MAPK cascade of Fig. 3

```
@model:2.3.1=MAPK "MAPK"
@compartments
 cell=1
@species
 cell:e1=1 "Enzyme E1"
 cell:kkk=3 "MAPKKK"
 cell:kkk_e1=0 "MAPKKK:E1"
 cell:kkkp=0 "MAPKKK*"
 ...
@parameters
 a1=0.3333333333333333
 d1=150
 k1=150
 ...
@reactions
@r=r1a "MAPKKK is activated through enzyme E1 - 1a"
 kkk+e1 -> kkk_e1
 a1*kkk*e1
@r=r1d "MAPKKK is activated through enzyme E1 - 1d"
 kkk_e1 -> kkk+e1
 d1*kkk_e1
@r=r1k "MAPKKK is activated through enzyme E1 - 1k"
 kkk_e1-> kkkp+e1
 k1*kkk_e1
...
```

species in the system, the reactions they can undergo, and the kinetic laws and parameters which govern these reactions. Again, support for PRISM is provided through a language translator [33]. For illustration, Fig. 12 shows a fragment of the "SBML-shorthand" [35] code which describes the set of MAPK reactions used throughout this chapter. This simple textual language can be automatically translated [35] into SBML. When the SBML model produced is then converted into PRISM code [33], the resulting CTMC is identical to the one used in this chapter.

Further mechanisms are also available for input of models into PRISM. The tool includes a simple pre-processing language (PRISM-PP) which can be used to automatically generate model and property specifications that contain a lot of repetition. Markov chains can also be imported directly (through an explicit list of their states, transitions, and rates) allowing models to be generated in other tools and then analysed in PRISM.

Conversely, it is also possible to use external tools to analyse PRISM models. One example is the statistical based model-checker Ymer [37], which performs approximate CSL model checking of CTMCs expressed as PRISM models, using discrete-event simulation and sequential acceptance sampling (for a detailed comparison of the merits of this approach and the probabilistic model checking techniques used by PRISM, see [38]). Another example is the tool GRIP (Generic Representatives In PRISM) [7], which performs language-level symmetry reduction of PRISM models based on the generic representatives approach of [8]. Further support for symmetry reduction is provided by PRISM-symm [19], a prototype extension of PRISM which uses an efficient symbolic (MTBDD-based) implementation.

Finally, models that have been specified in the PRISM modeling language can be constructed in PRISM, and then exported to an explicit representation of the Markov chain for analysis in other tools. In particular, this output can be customized for the probabilistic model checkers Markov Reward Model Checker (MRMC) [16] and the Erlangen–Twente Markov Chain Checker (ETMCC) [12] which can both be used for verifying CTMCs against CSL specifications. MRMC also supports rewards-based property specifications through the logic CSRL [6]. Models, in addition to other PRISM outputs such as numerical results or simulation traces, can be imported into more general-purpose tools such as MATLAB [23] and MAPLE [22].

5 Conclusions

We have illustrated how probabilistic model checking and, in particular, the probabilistic model checker PRISM can be employed as a framework for the analysis of biological pathways. One of the key strengths of this approach is that it allows for the computation of exact quantitative measures relating to the evolution of the system over time. Since as we have demonstrated, it is possible to specify and verify a wide variety of such measures, a detailed, quantitative analysis of the interactions between the components of a pathway is possible.

The principal challenge remaining for the application of probabilistic model checking to biological systems, as in so many other domains, is the scalability of the techniques to ever larger systems and models. There is hope that some of the techniques that have already been developed in the field of formal verification, such as symmetry reduction, bi-simulation minimization and abstraction, will prove beneficial in this area. For further details on such approaches and pointers to related work; see, for example [11].

Acknowledgements This work was supported in part by EPSRC grants EP/D07956X and EP/D076625.

References

1. Aziz A, Sanwal K, Singhal V, Brayton R (2000) Model checking continuous time Markov chains. ACM Trans Comput Log 1(1):162–170
2. Baier C, Haverkort B, Hermanns H, Katoen J-P (2003) Model-checking algorithms for continuous-time Markov chains. IEEE Trans Softw Eng 29(6):524–541
3. Calder M, Gilmore S, Hillston J (2006) Modelling the influence of RKIP on the ERK signaling pathway using the stochastic process algebra PEPA. Trans Comput Syst Biol 7:1–23
4. Calder M, Vyshemirsky V, Gilbert D, Orton R (2006) Analysis of signaling pathways using continuous time Markov chains. Trans Comput Syst Biol 4:44–67
5. Clarke E, Emerson E, Sistla A (1986) Automatic verification of finite-state concurrent systems using temporal logics. ACM Trans Program Lang Syst 8(2):244–263
6. Cloth L, Katoen J-P, Khattri M, Pulungan R (2005) Model checking Markov reward models with impulse rewards. In: Proceedings of the international conference on dependable systems and networks (DSN'05). IEEE Computer Society, Los Alamitos, pp 722–731

7. Donaldson A, Miller A, Parker D (2007) GRIP: Generic representatives in PRISM. In: Proceedings of the 4th international conference on quantitative evaluation of systems (QEST'07). IEEE Computer Society, Los Alamitos, pp 115–116
8. Emerson E, Wahl T (2003) On combining symmetry reduction and symbolic representation for efficient model checking. In: Geist D, Tronci E (eds) Proceedings of the 12th conference on correct hardware design and verification methods (CHARME 2003). Lecture notes in computer science, vol 2860. Springer, Berlin, pp 216–230
9. Gillespie D (1977) Exact stochastic simulation of coupled chemical reactions. J Phys Chem 81(25):2340–2361
10. Hansson H, Jonsson B (1994) A logic for reasoning about time and reliability. Form Asp Comput 6(5):512–535
11. Heath J, Kwiatkowska M, Norman G, Parker D, Tymchyshyn O (2008) Probabilistic model checking of complex biological pathways. Theor Comput Sci 319:239–257
12. Hermanns H, Katoen J-P, Meyer-Kayser J, Siegle M (2003) A tool for model checking Markov chains. Softw Tools Technol Transf 4(2):153–172
13. Hillston J (1996) A compositional approach to performance modeling. Cambridge University Press, Cambridge
14. Hinton A, Kwiatkowska M, Norman G, Parker D (2006) PRISM: A tool for automatic verification of probabilistic systems. In: Hermanns H, Palsberg J (eds) Proceedings of the 12th international conference on tools and algorithms for the construction and analysis of systems (TACAS'06). Lecture notes in computer science, vol 3920. Springer, Berlin, pp 441–444
15. Huang C, Ferrell J (1996) Ultrasensitivity in the mitogen-activated protein kinase cascade. Proc Natl Acad Sci 93:10078–10083
16. Katoen J-P, Khattri M, Zapreev I (2005) A Markov reward model checker. In: Proceedings of the 2nd international conference on quantitative evaluation of systems (QEST'05). IEEE Computer Society, Los Alamitos, pp 243–244
17. Kwiatkowska M, Norman G, Parker D (2004) Probabilistic symbolic model checking with PRISM: a hybrid approach. Softw Tools Technol Transf 6(2):128–142
18. Kwiatkowska M, Norman G, Parker D (2005) Probabilistic model checking in practice: case studies with PRISM. ACM SIGMETRICS Perform Eval Rev 32(4):16–21
19. Kwiatkowska M, Norman G, Parker D (2006) Symmetry reduction for probabilistic model checking. In: Ball T, Jones R (eds) Proceedings of the 18th international conference on computer aided verification (CAV'06). Lecture notes in computer science, vol 4114. Springer, Berlin, pp 234–248
20. Kwiatkowska M, Norman G, Parker D (2007) Stochastic model checking. In: Bernardo M, Hillston J (eds) Formal methods for the design of computer, communication and software systems: performance evaluation (SFM'07). Lecture notes in computer science, vol 4486. Springer, Berlin, pp 220–270
21. Lecca P, Priami C (2003) Cell cycle control in eukaryotes: a BioSpi model. In: Proceedings of the workshop concurrent models in molecular biology (BioConcur'03). Electronic notes in theoretical computer science
22. MAPLE. www.maplesoft.com/products/maple/
23. MATLAB. http://www.mathworks.com/products/matlab/
24. Norman G, Palamidessi C, Parker D, Wu P (2007) Model checking the probabilistic π-calculus. In: Proceedings of the 4th international conference on quantitative evaluation of systems (QEST'07). IEEE Computer Society, Los Alamitos, pp 169–178
25. Parker D (2002) Implementation of symbolic model checking for probabilistic systems. PhD thesis, University of Birmingham
26. Phillips A, Cardelli L (2007) Efficient, correct simulation of biological processes in the stochastic π-calculus. In: Calder M, Gilmore S (eds) Proceedings of the 5th international conference on computational methods in systems biology (CMSB'07). Lecture notes in bioinformatics, vol 4695. Springer, Berlin, pp 184–199
27. PRISM web site. http://www.prismmodelchecker.org/
28. Priami C (1995) Stochastic π-calculus. Comput J 38(7):578–589

29. Priami C, Regev A, Silverman W, Shapiro E (2001) Application of a stochastic name passing calculus to representation and simulation of molecular processes. Inf Process Lett 80:25–31
30. Pronk T, de Vink E, Bosnacki D, Breit T (2007) Stochastic modeling of codon bias with PRISM. In: Linden I, Talcott C (eds) Proceedings of the 3rd international workshop on methods and tools for coordinating concurrent, distributed and mobile systems (MTCoord 2007). Computer Science Department, University of Cyprus, Nicosia
31. Romero-Campero F, Gheorghe M, Bianco L, Pescini D, Pérez-Jiménez M, Ceterchi R (2006) Towards probabilistic model checking on P systems using PRISM. In: Hoogeboom H, Păun G, Rozenberg G, Salomaa A (eds) Proceedings of the 7th international workshop on membrane computing (WMC06). Lecture notes in computer science, vol 4361. Springer, Berlin, pp 477–495
32. Rutten J, Kwiatkowska M, Norman G, Parker D (2004) Mathematical techniques for analyzing concurrent and probabilistic systems. CRM monograph series, vol 23. AMS, Providence
33. SBML-to-PRISM translator. http://www.prismmodelchecker.org/sbml/
34. SBML web site. http://sbml.org/
35. SBML-shorthand. http://www.staff.ncl.ac.uk/d.j.wilkinson/software/sbml-sh/
36. Wolkenhauer O, Ullah M, Kolch W, Cho K-H (2004) Modeling and simulation of intracellular dynamics: choosing an appropriate framework. IEEE Trans Nanobiosci 3:200–207
37. Younes H (2005) Ymer: A statistical model checker. In: Etessami K, Rajamani S (eds) Proceedings of the 17th international conference on computer aided verification (CAV'05). Lecture notes in computer science, vol 3576. Springer, Berlin, pp 429–433
38. Younes H, Kwiatkowska M, Norman G, Parker D (2006) Numerical vs. statistical probabilistic model checking. Softw Tools Technol Transf 8(3):216–228

A New Mathematical Model for the Heat Shock Response

Ion Petre, Andrzej Mizera, Claire L. Hyder,
Andrey Mikhailov, John E. Eriksson,
Lea Sistonen, and Ralph-Johan Back

Abstract We present in this paper a novel molecular model for the gene regulatory network responsible for the eukaryotic heat shock response. Our model includes the temperature-induced protein misfolding, the chaperone activity of the heat shock proteins, and the backregulation of their gene transcription. We then build a mathematical model for it, based on ordinary differential equations. Finally, we discuss the parameter fit and the implications of the sensitivity analysis for our model.

1 Introduction

One of the most impressive algorithmic-like bioprocesses in living cells, crucial for the very survival of cells is the *heat shock response*: the reaction of the cell to elevated temperatures. One of the effects of raised temperature in the environment is that proteins get misfolded, with a rate that is exponentially dependent on the temperature. As an effect of their hydrophobic core being exposed, misfolded proteins tend to form bigger and bigger aggregates, with disastrous consequences for the cell; see [1]. To survive, the cell needs to increase quickly the level of chaperons (proteins that are assisting in the folding or refolding of other proteins). Once the heat shock is removed, the cell eventually reestablishes the original level of chaperons; see [10, 18, 22].

The heat shock response has been subject of intense research in the last few years, for at least three reasons. First, it is a well-conserved mechanism across all eukaryotes, while bacteria exhibit only a slightly different response; see [5, 12, 23]. As such, it is a good candidate for studying the engineering principle of gene regulatory networks; see [4, 5, 12, 25]. Second, it is a tempting mechanism to model mathematically, since it involves only very few reactants, at least in a simplified presentation; see [18, 19, 22]. Third, the heat shock proteins (the main chaperons involved in the eukaryotic heat shock response) play a central role in a large number of regulatory

I. Petre (✉)
Department of Information Technologies, Åbo Akademi University, Turku 20520,
Finland
e-mail: ipetre@abo.fi

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_21, © Springer-Verlag Berlin Heidelberg 2009

and of inflammatory processes, as well as in signaling; see [9, 20]. Moreover, they contribute to the resilience of cancer cells, which makes them attractive as targets for cancer treatment; see [3, 15, 16, 27].

We focus in this paper on a new molecular model for the heat shock response, proposed in [19]. We consider here a slight extension of the model in [19] where, among others, the chaperons are also subject to misfolding. After introducing the molecular model in Sect. 2, we build a mathematical model in Sect. 3, including the fitting of the model with respect to experimental data. We discuss in Sect. 4 the results of the sensitivity analysis of the model, including its biological implications.

2 A New Molecular Model for the Eukaryotic Heat Shock Response

The heat shock proteins (hsp) play the key role in the heat shock response. They act as chaperons, helping misfolded proteins (mfp) to refold. The response is controlled in our model through the regulation of the transactivation of the hsp-encoding genes. The transcription of the gene is promoted by some proteins called heat shock factors (hsf) that trimerize and then bind to a specific DNA sequence called heat shock element (hse), upstream of the hsp-encoding gene. Once the hsf trimer is bound to the heat shock element, the gene is transactivated and the synthesis of hsp is thus switched on (for the sake of simplicity, the role of RNA is ignored in our model). Once the level of hsp is high enough, the cell has an ingenious mechanism to switch off the hsp synthesis. For this, hsp bind to free hsf, as well as break the hsf trimers (including those bound to hse, promoting the gene activation), thus effectively halting the hsp synthesis.

Under elevated temperatures, some of the proteins (prot) in the cell get misfolded. The heat shock response is then quickly switched on simply because the heat shock proteins become more and more active in the refolding process, thus leaving the heat shock factors free and able to promote the synthesis of more heat shock proteins. Note that several types of heat shock proteins exist in an eukaryotic cell. We treat them all uniformly in our model, with hsp70 as common denominator. The same comment applies also to the heat shock factors.

Our molecular model for the eukaryotic heat shock response consists of the following molecular reactions:

1. $2\,\mathsf{hsf} \leftrightarrows \mathsf{hsf}_2$
2. $\mathsf{hsf} + \mathsf{hsf}_2 \leftrightarrows \mathsf{hsf}_3$
3. $\mathsf{hsf}_3 + \mathsf{hse} \leftrightarrows \mathsf{hsf}_3 : \mathsf{hse}$
4. $\mathsf{hsf}_3 : \mathsf{hse} \to \mathsf{hsf}_3 : \mathsf{hse} + \mathsf{mhsp}$
5. $\mathsf{hsp} + \mathsf{hsf} \leftrightarrows \mathsf{hsp} : \mathsf{hsf}$
6. $\mathsf{hsp} + \mathsf{hsf}_2 \to \mathsf{hsp} : \mathsf{hsf} + \mathsf{hsf}$
7. $\mathsf{hsp} + \mathsf{hsf}_3 \to \mathsf{hsp} : \mathsf{hsf} + 2\,\mathsf{hsf}$
8. $\mathsf{hsp} + \mathsf{hsf}_3 : \mathsf{hse} \to \mathsf{hsp} : \mathsf{hsf} + 2\,\mathsf{hsf} + \mathsf{hse}$
9. $\mathsf{hsp} \to \emptyset$

10. $\mathsf{prot} \rightarrow \mathsf{mfp}$
11. $\mathsf{hsp} + \mathsf{mfp} \leftrightarrows \mathsf{hsp}:\mathsf{mfp}$
12. $\mathsf{hsp}:\mathsf{mfp} \rightarrow \mathsf{hsp} + \mathsf{prot}$
13. $\mathsf{hsf} \rightarrow \mathsf{mhsf}$
14. $\mathsf{hsp} \rightarrow \mathsf{mhsp}$
15. $\mathsf{hsp} + \mathsf{mhsf} \leftrightarrows \mathsf{hsp}:\mathsf{mhsf}$
16. $\mathsf{hsp}:\mathsf{mhsf} \rightarrow \mathsf{hsp} + \mathsf{hsf}$
17. $\mathsf{hsp} + \mathsf{mhsp} \leftrightarrows \mathsf{hsp}:\mathsf{mhsp}$
18. $\mathsf{hsp}:\mathsf{mhsp} \rightarrow 2\,\mathsf{hsp}$

It is important to note that the main addition we consider here with respect to the model in [19] is to include the misfolding of hsp and hsf. This is, in principle, no minor extension since in the current model the repairing mechanism is subject to failure, but it is capable to fix itself.

Several criteria were followed when introducing this molecular model:

(i) as few reactions and reactants as possible;
(ii) include the temperature-induced protein misfolding;
(iii) include hsf in all its three forms: monomers, dimers, and trimers;
(iv) include the hsp-backregulation of the transactivation of the hsp-encoding gene;
(v) include the chaperon activity of hsp;
(vi) include only well-documented, textbook-like reactions and reactants.

For the sake of keeping the model as simple as possible, we are ignoring a number of details. For example, note that there is no notion of locality in our model: we make no distinction between the place where gene transcription takes place (inside nucleus) and the place where protein synthesis takes place (outside nucleus). Note also that protein synthesis and gene transcription are greatly simplified in reaction 4: we only indicate that once the gene is transactivated, protein synthesis is also switched on. On the other hand, reaction 4 is faithful to the biological reality; see [1] in indicating that newly synthesized proteins often need chaperons to form their native fold.

As far as protein degradation is concerned, we only consider it in the model for hsp. If we considered it also for hsf and prot, then we should also consider the compensating mechanism of protein synthesis, including its control. For the sake of simplicity and also based on experimental evidence that the total amount of hsf and of prot is somewhat constant, we ignore the details of synthesis and degradation for hsf and prot.

3 The Mathematical Model

We build in this section a mathematical model associated to the molecular model 1–18. Our mathematical model is in terms of coupled ordinary differential equations and its formulation is based on the principle of mass-action.

3.1 The Principle of Mass-Action

The mass-action law is widely used in formulating mathematical models in physics, chemistry, and engineering. Introduced in [6, 7], it can be briefly summarized as follows: *the rate of each reaction is proportional to the concentration of reactants*. In turn, the rate of each reaction gives the rate of consuming the reactants and the rate of producing the products. For example, for a reaction

$$R_1 : A + B \rightarrow C,$$

the rate according to the principle of mass action is $f_1(t) = kA(t)B(t)$, where $k \geq 0$ is a constant and $A(t)$, $B(t)$ are functions of time giving the level of the reactants A and B, respectively. Consequently, the rate of consuming A and B, and the rate of producing C is expressed by the following differential equations:

$$\frac{dA}{dt} = \frac{dB}{dt} = -k\,A(t)\,B(t), \qquad \frac{dC}{dt} = k\,A(t)\,B(t).$$

For a reversible reaction,

$$R_2 : A + B \leftrightarrows C,$$

the rate is $f_2(t) = k_1 A(t)B(t) - k_2 C(t)$, for some constants $k_1, k_2 \geq 0$. The differential equations are written in a similar way:

$$\frac{dA}{dt} = \frac{dB}{dt} = -f_2(t), \qquad \frac{dC}{dt} = f_2(t). \tag{$*$}$$

For a set of coupled reactions, the differential equations capture the combined rate of consuming and producing each reactant as an effect of all reactions taking place simultaneously. For example, for reactions

$$R_3 : A + B \leftrightarrows C, \qquad R_4 : B + C \leftrightarrows A, \qquad R_5 : A + C \leftrightarrows B,$$

the associated system of differential equations is

$$\begin{aligned} dA/dt &= -f_3(t) + f_4(t) - f_5(t), \\ dB/dt &= -f_3(t) - f_4(t) + f_5(t), \\ dC/dt &= f_3(t) - f_4(t) - f_5(t), \end{aligned}$$

where $f_i(t)$ is the rate of reaction R_i, for all $3 \leq i \leq 5$, formulated according to the principle of mass action.

We recall that for a system of differential equations

$$\frac{dX_1}{dt} = f_1(X_1, \ldots, X_n),$$

$$\vdots$$

$$\frac{dX_n}{dt} = f_n(X_1, \ldots, X_n),$$

we say that $(x_1, x_2, \ldots, x_n)$ is a *steady states* (also called *equilibrium points*) if it is a solution of the algebraic system of equations $f_i(X_1, \ldots, X_n) = 0$, for all $1 \leq i \leq n$, see [24, 28]. Steady states are particularly interesting because they characterize situations where although reactions may have nonzero rates, their combined effect is zero. In other words, the concentration of all reactants and of all products are constant.

We refer to [11, 17, 29] for more details on the principle of mass action and its formulation based on ordinary differential equations.

3.2 Our Mathematical Model

Let $\mathbb{R}_+$ be the set of all positive real numbers and $\mathbb{R}_+^n$ the set of all n-tuples of positive real numbers, for $n \geq 2$. We denote each reactant and bond between them in the molecular model 1–18 according to the convention in Table 1. We also denote by $\kappa \in \mathbb{R}_+^{17}$ the vector with all reaction rate constants as its components; see Table 2: $\kappa = (k_1^+, k_1^-, k_2^+, k_2^-, k_3^+, k_3^-, k_4, k_5^+, k_5^-, k_6, k_7, k_8, k_9, k_{11}^+, k_{11}^-, k_{12}, k_{13}^+, k_{13}^-, k_{14}, k_{15}^+, k_{15}^-, k_{16})$.

The mass action-based formulation of the associated mathematical model in terms of differential equations is straightforward, leading to the following system

Table 1 The list of variables in the mathematical model, their initial values, and their values in one of the steady states of the system, for $T = 42$. Note that the initial values give one of the steady states of the system for $T = 37$

Metabolite	Variable	Initial value	A steady state ($T = 42$)
hsf	X_1	0.669	0.669
hsf$_2$	X_2	8.73×10^{-4}	8.73×10^{-4}
hsf$_3$	X_3	1.23×10^{-4}	1.23×10^{-4}
hsf$_3$: hse	X_4	2.956	2.956
mhsf	X_5	3.01×10^{-6}	2.69×10^{-5}
hse	X_6	29.733	29.733
hsp	X_7	766.875	766.875
mhsp	X_8	3.45×10^{-3}	4.35×10^{-2}
hsp : hsf	X_9	1403.13	1403.13
hsp : mhsf	X_{10}	4.17×10^{-7}	3.72×10^{-6}
hsp : mhsp	X_{11}	4.78×10^{-4}	6.03×10^{-3}
hsp : mfp	X_{12}	71.647	640.471
prot	X_{13}	1.14×10^{8}	1.14×10^{8}
mfp	X_{14}	517.352	4624.72

Table 2 The numerical values for the fitted model

Kinetic constant	Reaction	Numerical value
k_1^+	(1), forward	3.49091
k_1^-	(1), backward	0.189539
k_2^+	(2), forward	1.06518
k_2^-	(2), backward	1×10^{-9}
k_3^+	(3), forward	0.169044
k_3^-	(3), backward	1.21209×10^{-6}
k_4	(4)	0.00830045
k_5^+	(5), forward	9.73665
k_5^-	(5), backward	3.56223
k_6	(6)	2.33366
k_7	(7)	4.30924×10^{-5}
k_8	(8)	2.72689×10^{-7}
k_9	(9)	3.2×10^{-5}
k_{11}^+	(11), forward	0.00331898
k_{11}^-	(11), backward	4.43952
k_{12}	(12)	13.9392
k_{13}^+	(15), forward	0.00331898
k_{13}^-	(15), backward	4.43952
k_{14}	(16)	13.9392
k_{15}^+	(17), forward	0.00331898
k_{15}^-	(17), backward	4.43952
k_{16}	(18)	13.9392

of equations:

$$dX_1/dt = f_1(X_1, X_2, \dots, X_{14}, \kappa), \tag{1}$$

$$dX_2/dt = f_2(X_1, X_2, \dots, X_{14}, \kappa), \tag{2}$$

$$dX_3/dt = f_3(X_1, X_2, \dots, X_{14}, \kappa), \tag{3}$$

$$dX_4/dt = f_4(X_1, X_2, \dots, X_{14}, \kappa), \tag{4}$$

$$dX_5/dt = f_5(X_1, X_2, \dots, X_{14}, \kappa), \tag{5}$$

$$dX_6/dt = f_6(X_1, X_2, \dots, X_{14}, \kappa), \tag{6}$$

$$dX_7/dt = f_7(X_1, X_2, \dots, X_{14}, \kappa), \tag{7}$$

$$dX_8/dt = f_8(X_1, X_2, \dots, X_{14}, \kappa), \tag{8}$$

$$dX_9/dt = f_9(X_1, X_2, \dots, X_{14}, \kappa), \tag{9}$$

$$dX_{10}/dt = f_{10}(X_1, X_2, \dots, X_{14}, \kappa), \tag{10}$$

$$dX_{11}/dt = f_{11}(X_1, X_2, \dots, X_{14}, \kappa), \tag{11}$$

$$dX_{12}/dt = f_{12}(X_1, X_2, \dots, X_{14}, \kappa), \tag{12}$$

$$dX_{13}/dt = f_{13}(X_1, X_2, \dots, X_{14}, \kappa), \tag{13}$$

$$dX_{14}/dt = f_{14}(X_1, X_2, \dots, X_{14}, \kappa), \tag{14}$$

where

$$\begin{aligned}
f_1 &= -k_2^+ X_1 X_2 + k_2^- X_3 - k_5^+ X_1 X_7 + k_5^- X_9 + 2k_8 X_4 X_7 + k_6 X_2 X_7 \\
&\quad - \varphi(T) X_1 + k_{14} X_{10} + 2k_7 X_3 X_7 - 2k_1^+ X_1^2 + 2k_1^- X_2, \\
f_2 &= -k_2^+ X_1 X_2 + k_2^+ X_3 - k_6 X_2 X_7 + k_1^+ X_1^2 - k_1^- X_2, \\
f_3 &= -k_3^+ X_3 X_6 + k_2^+ X_1 X_2 - k_2^- X_3 + k_3^- X_4 - k_7 X_3 X_7, \\
f_4 &= k_3^+ X_3 X_6 - k_3^- X_4 - k_8 X_4 X_7, \\
f_5 &= \varphi(T) X_1 - k_{13}^+ X_5 X_7 + k_{13}^- X_{10}, \\
f_6 &= -k_3^+ X_3 X_6 + k_3^- X_4 + k_8 X_4 X_7, \\
f_7 &= -k_5^+ X_1 X_7 + k_5^- X_9 - k_{11}^+ X_7 X_{14} + k_{11}^- X_{12} - k_8 X_4 X_7 - k_6 X_2 X_7 \\
&\quad - k_{13}^+ X_5 X_7 + \left(k_{13}^- + k_{14}\right) X_{10} - \left(\varphi(T) + k_9\right) X_7 - k_{15}^+ X_7 X_8 \\
&\quad - k_7 X_3 X_7 + \left(k_{15}^- + 2k_{16}\right) X_{11} + k_{12} X_{12}, \\
f_8 &= k_4 X_4 + \varphi(T) X_7 - k_{15}^+ X_7 X_8 + k_{15}^- X_{11}, \\
f_9 &= k_5^+ X_1 X_7 - k_5^- X_9 + k_8 X_4 X_7 + k_6 X_2 X_7 + k_7 X_3 X_7, \\
f_{10} &= k_{13}^+ X_5 X_7 - \left(k_{13}^- + k_{14}\right) X_{10}, \\
f_{11} &= k_{15}^+ X_7 X_8 - \left(k_{15}^- + k_{16}\right) X_{11}, \\
f_{12} &= k_{11}^+ X_7 X_{14} - \left(k_{11}^- + k_{12}\right) X_{12}, \\
f_{13} &= k_{12} X_{12} - \varphi(T) X_{13}, \\
f_{14} &= -k_{11}^+ X_7 X_{14} + k_{11}^- X_{12} + \varphi(T) X_{13}.
\end{aligned}$$

The rate of protein misfolding $\varphi(T)$ with respect to temperature T has been investigated experimentally in [13, 14], and a mathematical expression for it has been proposed in [18]. We have adapted the formula in [18] to obtain the following misfolding rate per second:

$$\varphi(T) = \left(1 - \frac{0.4}{e^{T-37}}\right) \times 0.8401033733 \times 10^{-6} \times 1.4^{T-37}\ \mathrm{s}^{-1},$$

where T is the temperature of the environment in Celsius degrees, with the formula being valid for $37 \le T \le 45$.

The following result gives three mass-conservation relations for our model.

Theorem 1 *There exists* $K_1, K_2, K_3 \geq 0$ *such that*

(i) $X_1(t) + 2X_2(t) + 3X_3(t) + 3X_4(t) + X_5(t) + X_9(t) = K_1$,
(ii) $X_4(t) + X_6(t) = K_2$,
(iii) $X_{13}(t) + X_{14}(t) + X_{12}(t) = K_3$,

for all $t \geq 0$.

Proof We only prove here part (ii), as the others may be proved analogously. For this, note that from (4) and (6), it follows that

$$\frac{d(X_4 + X_6)}{dt} = (f_4 + f_6)(X_1, \ldots, X_{14}, \kappa, t) = 0,$$

i.e., $(X_4 + X_6)(t)$ is a constant function. □

The steady states of the model (1)–(14) satisfy the following algebraic relations, where x_i is the numerical value of X_i in the steady state, for all $1 \leq i \leq 14$.

$$\begin{aligned} 0 = {} & -k_2^+ x_1 x_2 + k_2^- x_3 - k_5^+ x_1 x_7 + k_5^- x_9 + 2k_8 x_4 x_7 + k_6 x_2 x_7 \\ & - \varphi(T) x_1 + k_{14} x_{10} + 2k_7 x_3 x_7 - 2k_1^+ x_1^2 + 2k_1^- x_2, \end{aligned} \tag{15}$$

$$0 = -k_2^+ x_1 x_2 + k_2^+ x_3 - k_6 x_2 x_7 + k_1^+ x_1^2 - k_1^- x_2, \tag{16}$$

$$0 = -k_3^+ x_3 x_6 + k_2^+ x_1 x_2 - k_2^- x_3 + k_3^- x_4 - k_7 x_3 x_7, \tag{17}$$

$$0 = k_3^+ x_3 x_6 - k_3^- x_4 - k_8 x_4 x_7, \tag{18}$$

$$0 = \varphi(T) x_1 - k_{13}^+ x_5 x_7 + k_{13}^- x_{10}, \tag{19}$$

$$0 = -k_3^+ x_3 x_6 + k_3^- x_4 + k_8 x_4 x_7, \tag{20}$$

$$\begin{aligned} 0 = {} & -k_5^+ x_1 x_7 + k_5^- x_9 - k_{11}^+ x_7 x_{14} + k_{11}^- x_{12} - k_8 x_4 x_7 - k_6 x_2 x_7 \\ & - k_{13}^+ x_5 x_7 + \big(k_{13}^- + k_{14}\big) x_{10} - \big(\varphi(T) + k_9\big) x_7 - k_{15}^+ x_7 x_8 - k_7 x_3 x_7 \\ & + \big(k_{15}^- + 2k_{16}\big) x_{11} + k_{12} x_{12}, \end{aligned} \tag{21}$$

$$0 = k_4 x_4 + \varphi(T) x_7 - k_{15}^+ x_7 x_8 + k_{15}^- x_{11}, \tag{22}$$

$$0 = k_5^+ x_1 x_7 - k_5^- x_9 + k_8 x_4 x_7 + k_6 x_2 x_7 + k_7 x_3 x_7, \tag{23}$$

$$0 = k_{13}^+ x_5 x_7 - \big(k_{13}^- + k_{14}\big) x_{10}, \tag{24}$$

$$0 = k_{15}^+ x_7 x_8 - \big(k_{15}^- + k_{16}\big) x_{11}, \tag{25}$$

$$0 = k_{11}^+ x_7 x_{14} - \big(k_{11}^- + k_{12}\big) x_{12}, \tag{26}$$

$$0 = k_{12} x_{12} - \varphi(T) x_{13}, \tag{27}$$

$$0 = -k_{11}^+ x_7 x_{14} + k_{11}^- x_{12} + \varphi(T) x_{13}. \tag{28}$$

It follows from Theorem 1 that only eleven of the relations above are independent. For example, relations (15)–(17), (19), (21)–(27) are independent. The system consisting of the corresponding differential equations is called the *reduced system* of (1)–(14).

3.3 Fitting the Model to Experimental Data

The experimental data available for the parameter fit is from [10] and reflects the level of DNA binding, i.e., variable X_4 in our model, for various time points up to 4 hours, with continuous heat shock at 42°C. Additionally, we require that the initial value of the variables of the model is a steady state for temperature set to 37°C. This is a natural condition since the model is supposed to reflect the reaction to temperatures raised above 37°C.

Mathematically, the problem we need to solve is one of global optimization, as formulated below. For each 17-tuple κ of positive numerical values for all kinetic constants, and for each 14-tuple α of positive initial values for all variables in the model, the function $X_4(t)$ is uniquely defined for a fixed temperature T. We denote the value of this function at time point τ, with parameters κ and α by $x_4^T(\kappa, \alpha, \tau)$. Note that this property holds for all the other variables in the model and it is valid in general for any mathematical model based on ordinary differential equations (one calls such models *deterministic*). We denote the set of experimental data in [10] by

$$E_n = \left\{(t_i, r_i) \mid t_i, r_i > 0, 1 \leq i \leq N\right\},$$

where $N \geq 1$ is the number of observations, t_i is the time point of each observation and r_i is the value of the reading.

With this setup, we can now formulate our optimization problem as follows: find $\kappa \in \mathbb{R}_+^{17}$ and $\alpha \in \mathbb{R}_+^{14}$ such that

(i) $f(\kappa, \alpha) = \frac{1}{N} \sum_{i=1}^{N} (x_4^{42}(\kappa, \alpha, t_i) - r_i)^2$ is minimal and
(ii) α is a steady state of the model for $T = 37$ and parameter values given by κ.

The function $f(\kappa, \alpha)$ is a cost function (in this case *least mean squares*), indicating numerically how the function $x_4^T(\kappa, \alpha, t), t \geq 0$, compares with the experimental data.

Note that in our optimization problem, not all 31 variables (the components of κ and α) are independent. On one hand, we have the three algebraic relations given by Theorem 1. On the other hand, we have eleven more independent algebraic relations given by the steady state equations (15)–(17), (19), (21)–(27). Consequently, we have 17 independent variables in our optimization problem.

Given the high degree of the system (1)–(14), finding the analytical form of the minimum points of $f(\kappa, \alpha)$ is very challenging. This is a typical problem when the system of equations is nonlinear. Adding to the difficulty of the problem is the fact that the eleven independent steady state equations cannot be solved analytically, given their high overall degree.

Since an analytical solution to the model fitting problem is often intractable, the practical approach to such problems is to give a numerical simulation of a solution.

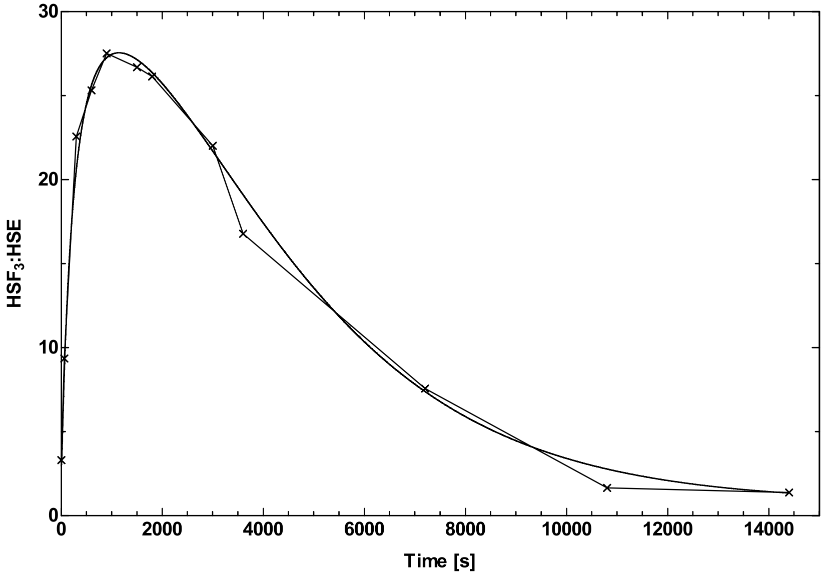


Fig. 1 The *continuous line* shows a numerical estimation of function $X_4(t)$, standing for DNA binding, for the initial data in Table 1 and the parameter values in Table 2. With crossed points, we indicated the experimental data of [10]

Several methods exist for this; see [2, 21]. The trade-off with all these methods is that typically they offer an estimate of a *local* optimum, with no guarantee of it being a *global* optimum.

Obtaining a numerical estimation of a local optimum for (i) is not difficult. However, such a solution may not satisfy (ii). To solve this problem, for a given local optimum $(\kappa_0, \alpha_0) \in \mathbb{R}_+^{17} \times \mathbb{R}_+^{14}$ one may numerically estimate a steady state $\alpha_1 \in \mathbb{R}_+^{14}$ for $T = 37$. Then the pair (κ_0, α_1) satisfies (ii). Unfortunately, (κ_0, α_1) may not be close to a local optimum of the cost function in (i).

Another approach is to replace the algebraic relations implicitly given by (ii) with an optimization problem similar to that in (i). Formally, we replace all algebraic relations $R_i = 0$, $1 \le i \le 11$, given by (ii) with the condition that

$$g(\kappa, \alpha) = \frac{1}{M} \sum_{j=1}^{M} R_i^2(\kappa, \alpha, \delta_j)$$

is minimal, where $0 < \delta_1 < \cdots < \delta_M$ are some arbitrary (but fixed) time points. Our problem thus becomes one of optimization with cost function (f, g), with respect to the order relation $(a, b) \le (c, d)$ if and only if $a \le c$ and $b \le d$. The numerical values in Table 2 give one solution to this problem obtained based on Copasi [8]. The plot in Fig. 1 shows the time evolution of function $X_4(t)$ up to $t = 4$ hours, with the experimental data of [10] indicated with crosses.

The solution in Table 2 has been compared with a number of other available experimental data (such as behavior at 41°C and at 43°C), as well as against qualitative, nonnumerical data. The results were satisfactory and better than those of previous models reported in the literature, such as [18, 22]. For details on the model validation analysis, we refer to [19].

Note that the steady state of the system of differential equations (1)–(14), for the initial values in Table 1 and the parameter values in Table 2 is *asymptotically stable*. To prove it, it is enough to consider its associated *Jacobian*:

$$J(t) = \begin{pmatrix} \partial f_1/\partial X_1 & \partial f_1/\partial X_2 & \dots & \partial f_1/\partial X_{14} \\ \partial f_2/\partial X_1 & \partial f_2/\partial X_2 & \dots & \partial f_2/\partial X_{14} \\ \vdots & \vdots & & \vdots \\ \partial f_{14}/\partial X_1 & \partial f_{14}/\partial X_2 & \dots & \partial f_{14}/\partial X_{14} \end{pmatrix}.$$

As it is well known, see [24, 28], a steady state is asymptotically stable if and only if all eigenvalues of the Jacobian at the steady state have negative real parts. A numerical estimation done with *Copasi* [8] shows that the steady state for $T = 42$; see Table 1, is indeed asymptotically stable.

4 Sensitivity Analysis

Sensitivity analysis is a method to estimate the changes brought into the system through small changes in the parameters of the model. In this way, one may estimate both the robustness of the model against small changes in the model, as well as identify possibilities for bringing a certain desired changed in the system. For example, one question that is often asked of a biochemical model is what changes should be done to the model so that the new steady state satisfies certain properties. In our case, we are interested in changing some of the parameters of the model so that the level of mfp in the new steady state of the system is smaller than in the standard model, thus presumably making it easier for the cell to cope with the heat shock. We also analyze a scenario in which we are interested in increasing the level of mfp in the new steady state, thus increasing the chances of the cell not being able to cope with the heat shock. Such a scenario is especially meaningful in relation with cancer cells that exhibit the properties of an excited cell, with increased levels of hsp; see [3, 15, 16, 27]. In this section, we follow in part a presentation of sensitivity analysis due to [26].

We consider the partial derivatives of the solution of the system with respect to the parameters of the system. These are called *first-order local concentration sensitivity coefficients*. Second- or higher-order sensitivity analysis considering the simultaneous change of two or more parameters is also possible. If we denote $X(t,\kappa) = (X_1(t,\kappa), X_2(t,\kappa), \dots, X_{14}(t,\kappa))$, the solution of the system (1)–(14) with respect to the parameter vector κ, then the concentration sensitivity coefficients are the time functions $\partial X_i/\partial \kappa_j(t)$, for all $1 \leq i \leq 14$, $1 \leq j \leq 17$. Differentiating the system (1)–(14) with respect to κ_j yields the following set of *sensitivity equations*:

$$\frac{d}{dt}\frac{\partial X}{\kappa_j} = J(t)\frac{\partial X}{\partial \kappa_j} + \frac{\partial f(t)}{\partial \kappa_j}, \quad \text{for all } 1 \le j \le 17, \tag{29}$$

where $\partial X/\partial \kappa_j = (\partial X_1/\partial \kappa_j, \ldots, \partial X_{14}/\partial \kappa_j)$ is the component-wise vector of partial derivatives, $f = (f_1, \ldots, f_{14})$ is the model function in (1)–(14), and $J(t)$ is the corresponding Jacobian. The initial condition for the system (29) is that $\partial X/\partial \kappa_j(0) = 0$, for all $1 \le j \le 17$.

The solution of the system (29) can be numerically integrated, thus obtaining a numerical approximation of the time evolution of the sensitivity coefficients. Very often, however, the focus is on sensitivity analysis around steady states. If the considered steady state is asymptotically stable, then one may consider the limit $\lim_{t\to\infty}(\partial X/\partial \kappa_j)(t)$, called *stationary sensitivity coefficients*. They reflect the dependency of the steady state on the parameters of the model. Mathematically, they are given by a set of algebraic equations obtained from (29) by setting $d/dt(\partial X/\kappa_j) = 0$. We then obtain the following algebraic equations:

$$\left(\frac{\partial X}{\partial \kappa_j}\right) = -J^{-1}F_j, \quad \text{for all } 1 \le j \le 17, \tag{30}$$

where J is the value of the Jacobian at the steady state and F_j is the jth column of the matrix $F = (\partial f_r/\partial \kappa_s)_{r,s}$ computed at the steady state.

When used for comparing the relative effect of a parameter change in two or more variables, the sensitivity coefficients must have the same physical dimension or be dimensionless; see [26]. Most often, one simply considers the matrix S' of (dimensionless) *normalized* (also called *scaled*) sensitivity coefficients:

$$S'_{ij} = \frac{\kappa_j}{X_i(t,\kappa)} \cdot \frac{\partial X_i(t,\kappa)}{\partial \kappa_j} = \frac{\partial \ln X_i(t,\kappa)}{\partial \ln \kappa_j}.$$

Numerical estimations of the normalized sensitivity coefficients for a steady state may be obtained, e.g. with Copasi. For X_{14} (standing for the level of mfp in the model), the most significant (with the largest module) sensitivity coefficients are the following:

- $\partial \ln(X_{14})/\partial \ln(T) = 14.24$,
- $\partial \ln(X_{14})/\partial \ln(k_1^+) = -0.16$,
- $\partial \ln(X_{14})/\partial \ln(k_2^+) = -0.16$,
- $\partial \ln(X_{14})/\partial \ln(k_5^+) = 0.49$,
- $\partial \ln(X_{14})/\partial \ln(k_5^-) = -0.49$,
- $\partial \ln(X_{14})/\partial \ln(k_6) = 0.16$,
- $\partial \ln(X_{14})/\partial \ln(k_9) = 0.15$,
- $\partial \ln(X_{14})/\partial \ln(k_{11}^+) = -0.99$,
- $\partial \ln(X_{14})/\partial \ln(k_{11}^-) = 0.24$,
- $\partial \ln(X_{14})/\partial \ln(k_{12}) = -0.24$.

These coefficients being most significant is consistent with the biological intuition that the level of mfp in the model is most dependant on the temperature (parameter T), on the rate of mfp being sequestered by hsp (parameters k_{11}^+ and k_{11}^-) and the rate of protein refolding (parameter k_{12}). However, the sensitivity coefficients also reveal less intuitive, but significant dependencies such as the one on the reaction rate of hsf being sequestered by hsp (parameters k_5^+ and k_5^-), on the rate of dissipation of hsf dimers (parameter k_6), or on the rate of dimer- and trimer-formation (parameters k_1^+ and k_2^+).

Note that the sensitivity coefficients reflect the changes in the steady state for *small* changes in the parameter. For example, increasing the temperature from 42 with 0.1% yields an increase in the level of mfp with 1.43%, roughly as predicted by $\partial \ln(X_{14})/\partial \ln(T) = 14.24$. An increase of the temperature from 42 with 10% yields, however, an increase in the level of mfp of 311.93%.

A similar sensitivity analysis may also be performed with respect to the initial conditions; see [26]. If we denote by $X^{(0)} = X(0,\kappa)$, the initial values of the vector X, for parameters κ, then the *initial concentration sensitivity coefficients* are obtained by differentiating system (1)–(14) with respect to $X^{(0)}$:

$$\frac{d}{dt}\frac{\partial X}{\partial X^{(0)}} = J(t)\frac{\partial X}{\partial X^{(0)}}(t), \tag{31}$$

with the initial condition that $\partial X/\partial X^{(0)}(0)$ is the identity matrix. Similarly, as for the parameter-based sensitivity coefficients, it is often useful to consider the normalized, dimensionless coefficients

$$\frac{\partial X_i}{\partial X^{(0)}{}_j}(t)\cdot\frac{X^{(0)}{}_j(t)}{X_i(t)} = \frac{\partial \ln(X_i)}{\partial \ln(X^{(0)}{}_j)}.$$

A numerical estimation of the initial concentration sensitivity coefficient of mfp around the steady state given in Table 2 for $T = 42$, shows that all are negligible except for the following two coefficients: $\partial \ln(X_{14})/\partial \ln(X_9^{(0)}) = -0.497748$ and $\partial \ln(X_{14})/\partial \ln(X_{13}^{(0)}) = 0.99$. While the biological significance of the dependency of mfp on the initial level of prot is obvious, its dependency on the initial level of hsp : hsf is perhaps not. Moreover, it turns out that several other variables have a significant dependency on the initial level of hsp : hsf:

- ○ $\partial \ln(X_1)/\partial \ln(X_9(0)) = 0.49$,
- ○ $\partial \ln(X_2)/\partial \ln(X_9(0)) = 0.49$,
- ○ $\partial \ln(X_3)/\partial \ln(X_9(0)) = 1.04$,
- ○ $\partial \ln(X_4)/\partial \ln(X_9(0)) = 0.49$,
- ○ $\partial \ln(X_{10})/\partial \ln(X_9(0)) = 0.49$,
- ○ $\partial \ln(X_6)/\partial \ln(X_9(0)) = -0.04$,
- ○ $\partial \ln(X_7)/\partial \ln(X_9(0)) = 0.49$,
- ○ $\partial \ln(X_9)/\partial \ln(X_9(0)) = 0.99$,
- ○ $\partial \ln(X_{14})/\partial \ln(X_9(0)) = -0.49$,
- ○ $\partial \ln(X_{11})/\partial \ln(X_9(0)) = 0.49$,

For example, increasing $X_9^{(0)}$ by 1% increases the steady state values of X_7 by 0.49% and decreases the level of X_{14} by 0.49%. Increasing $X_9^{(0)}$ by 10% increases the steady state values of X_7 by 4.85% and decreases the level of X_{14} by 4.63%.

The biological interpretation of this significant dependency of the model on the initial level of hsp : hsf is based on two arguments. On one hand, the most significant part (about two thirds) of the initial available molecules of hsp in our model are present in bonds with hsf. On the other hand, the vast majority of hsf molecules are initially bound to hsp. Thus, changes in the initial level of hsp : hsf have an immediate influence on the two main drivers of the heat shock response: hsp and hsf. Interestingly, the dependency of the model on the initial levels of either hsp or hsf is negligible.

Acknowledgements This work has been partially supported by the following grants from Academy of Finland: project 108421 and 203667 (to I.P.), the Center of Excellence on Formal Methods in Programming (to R-J.B.).

References

1. Alberts B, Johnson A, Lewis J, Raff M, Roberts K, Walter P (2004) Essential cell biology, 2nd edn. Garland Science, London
2. Burden RL, Douglas Faires J (1996) Numerical analysis. Thomson Brooks/Cole, Pacific Grove
3. Ciocca DR, Calderwood SK (2005) Heat shock proteins in cancer: diagnostic, prognostic, predictive, and treatment implications. Cell Stress Chaperones 10(2):86–103
4. El-Samad H, Kurata H, Doyle J, Gross CA, Khamash M (2005) Surviving heat shock: control strategies for robustness and performance. Proc Natl Acad Sci 102(8):2736–2741
5. El-Samad H, Prajna S, Papachristodoulu A, Khamash M, Doyle J (2003) Model validation and robust stability analysis of the bacterial heat shock response using sostools. In: Proceedings of the 42nd IEEE conference on decision and control, pp 3766–3741
6. Guldberg CM, Waage P (1864) Studies concerning affinity. In: Forhandlinger CM (ed) Videnskabs-Selskabet i Christiana, p. 35
7. Guldberg CM, Waage P (1879) Concerning chemical affinity. Erdmann's J Pract Chem 127:69–114
8. Hoops S, Sahle S, Gauges R, Lee C, Pahle J, Simus N, Singhal M, Xu L, Mendes P, Kummer U (2006) Copasi—a complex pathway simulator. Bioinformatics 22(24):3067–3074
9. Kampinga HK (1993) Thermotolerance in mammalian cells: protein denaturation and aggregation, and stress proteins. J Cell Sci 104:11–17
10. Kline MP, Morimoto RI (1997) Repression of the heat shock factor 1 transcriptional activation domain is modulated by constitutive phosphorylation. Mol Cell Biol 17(4):2107–2115
11. Klipp E, Herwig R, Kowald A, Wierling C, Lehrach H (2006) Systems biology in practice. Wiley–VCH, New York
12. Kurata H, El-Samad H, Yi TM, Khamash M, Doyle J (2001) Feedback regulation of the heat shock response in e.coli. In: Proceedings of the 40th IEEE conference on decision and control, pp 837–842
13. Lepock JR, Frey HE, Ritchie KP (1993) Protein denaturation in intact hepatocytes and isolated cellular organelles during heat shock. J Cell Biol 122(6):1267–1276
14. Lepock JR, Frey HE, Rodahl AM, Kruuv J (1988) Thermal analysis of chl v79 cells using differential scanning calorimetry: Implications for hyperthermic cell killing and the heat shock response. J Cell Physiol 137(1):14–24
15. Liu B, DeFilippo AM, Li Z (2002) Overcomming immune tolerance to cancer by heat shock protein vaccines. Mol Cancer Ther 1:1147–1151
16. Lukacs KV, Pardo OE, Colston MJ, Geddes DM, Alton EWFW (2000) Heat shock proteins in cancer therapy. In: Habib (ed) Cancer gene therapy: past achievements and future challenges. Kluwer, Dordrecht, pp 363–368
17. Nelson DL, Cox MM (2000) Principles of biochemistry, 3rd edn. Worth Publishers, New York
18. Peper A, Grimbergent CA, Spaan JAE, Souren JEM, van Wijk R (1997) A mathematical model of the hsp70 regulation in the cell. Int J Hyperthermia 14:97–124

19. Petre I, Hyder CL, Mizera A, Mikhailov A, Eriksson JE, Sistonen L, Back R-J (2008) Two metabolites are enough to drive the eukaryotic heat shock response. Manuscript
20. Pockley AG (2003) Heat shock proteins as regulators of the immune response. Lancet 362(9382):469–476
21. Press WH, Teukolsky SA, Vetterling WT, Flammery BP (2007) Numerical recipes: the art of scientific computing. Cambridge University Press, Cambridge
22. Rieger TR, Morimoto RI, Hatzimanikatis V (2005) Mathematical modeling of the eukaryotic heat shock response: dynamics of the hsp70 promoter. Biophys J 88(3):1646–1658
23. Srivastava R, Peterson MS, Bentley WE (2001) Stochastic kinetic analysis of the escherichia coli stres circuit using σ^{32}-targeted antisense. Biotechnol Bioeng 75(1):120–129
24. Taubes CH (2001) Modeling differential equations in biology. Cambridge University Press, Cambridge
25. Tomlin CJ, Axelrod JD (2005) Understanding biology by reverse engineering the control. Proc Natl Acad Sci 102(12):4219–4220
26. Turányi T (1990) Sensitivity analysis of complex kinetic systems. Tools and applications. J Math Chem 5:203–248
27. Workman P, de Billy E (2007) Putting the heat on cancer. Nat Med 13(12):1415–1417
28. Zill DG (2001) A first course in differential equations. Thomson, Tompa
29. Zill DG (2005) A first course in differential equations with modeling applications. Thomson, Tompa

Part VII Process Calculi and Automata

Artificial Biochemistry

Luca Cardelli

Abstract We model chemical and biochemical systems as collectives of interacting stochastic automata, with each automaton representing a molecule that undergoes state transitions. In this *artificial biochemistry*, automata interact by the equivalent of the law of mass action. We investigate several simple but intriguing automata collectives by stochastic simulation and by ODE analysis.

1 Introduction

Macromolecules

Molecular biology investigates the structure and function of biochemical systems starting from their basic building blocks: *macromolecules*. A macromolecule is a large, complex molecule (a protein or a nucleic acid) that usually has inner mutable state and external activity. Informal explanations of biochemical events trace individual macromolecules through their state changes and their interaction histories: a macromolecule is endowed with an *identity* that is retained through its transformations, even through changes in molecular energy and mass. A macromolecule, therefore, is qualitatively different from the *small molecules* of inorganic chemistry. Such molecules are *stateless*: in the standard notation for chemical reactions they are seemingly created and destroyed, and their inner structure is used mainly for the bookkeeping required by the conservation of mass.

Attributing identity and state transitions to molecules provides more than just a different way of looking at a chemical event: it solves a fundamental difficulty with chemical-style descriptions. Each macromolecule can have a huge number of internal states, exponentially with respect to its size, and can join with other macromolecules to from even larger state configurations, corresponding to the product of their states. If each molecular state is to be represented as a stateless chemical species, transformed by chemical reactions, then we have a huge explosion in the number of species and reactions with respect to the number of different macromolecules that actually, physically, exist. Moreover, macromolecules can join to each other indefinitely, resulting in situations corresponding to infinite sets of chemical reactions among infinite sets of different chemical species. In contrast, the description of a biochemical system at the level of macromolecular states and transitions

L. Cardelli (✉)
Microsoft Research, Cambridge, UK
e-mail: luca@microsoft.com

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_22, © Springer-Verlag Berlin Heidelberg 2009

remains finite: the unbounded complexity of the system is implicit in the *potential* molecular interactions, but does not have to be written down explicitly. Molecular biology textbooks widely adopt this finite description style, at least for the purpose of illustration.

Many proposal now exist that aim to formalize the combinatorial complexity of biological systems without a corresponding explosion in the notation. One of the earliest can be found in [4] (which inspired the title of this article), where an artificial formal frameworks is used to get insights into natural systems. More recently, the descriptive paradigm in systems biology has become that of *programs as models* [8, 12, 20, 21]. Macromolecules, in particular, are seen as *stateful concurrent agents* that *interact with each other* through a *dynamic interface*. While this style of descriptions is (like many others) not quite accurate at the atomic level, it forms the basis of a formalized and growing body of biological knowledge.

The complex chemical structure of a macromolecule is thus commonly abstracted into just internal states and potential interactions with the environment. Each macromolecule forms, symmetrically, part of the environment for the other macromolecules, and can be described without having to describe the whole environment. Such an *open system* descriptive style allows modelers to extend systems by composition, and is fundamental to avoid enumerating the whole combinatorial state of the system (as one ends up doing in *closed systems* of chemical reactions). The programs-as-models approach is growing in popularity with the growing modeling ambitions in systems biology, and is, incidentally, the same approach taken in the organization of software systems. The basic problem and the basic solution are similar: programs are finite and compact models of potentially unbounded state spaces.

Molecules as Automata

At the core, we can therefore regard a macromolecule as some kind of automaton, characterized by a set of internal states and a set of discrete transitions between states driven by external interactions. We can thus try to handle molecular automata by some branch of automata theory and its outgrowths: cellular automata, Petri nets, and process algebra. The peculiarities of biochemistry, however, are such that until recently one could not easily pick a suitable piece of automata theory off the shelf.

Many sophisticated approaches have now been developed, and we are particularly fond of stochastic process algebra [18, 19]. In this paper, however, we do our outmost to remain within the bounds of a much simpler theory. We go back, in a sense, to a time before cellular automata, Petri nets, and process algebra, which all arose from the basic intuition that automata should interact with each other. Our main criterion is that, as in finite-state automata, we should be able to easily and separately *draw* the individual automata, both as a visual aid to design and analysis, and to emulate the illustration-based approach found in molecular biology textbooks. As a measure of success, in this paper, we draw a large number of examples.

Technically, we place ourselves within a small fragment of a well-known process algebra (stochastic π-calculus), but the novelty of the application domain, namely the “mass action” behavior of large numbers of “well-mixed” automata, demands a broader outlook. We rely on the work in [2, 3] for foundations and in-depth analysis. In this paper, we aim instead to give a self-contained and accessible presentation of the framework, and to explore by means of examples the richness of emergent and unexpected behavior that can be obtained by large populations of simple automata. Our automata drawings are precise, but the only formalization can be found in the corresponding process algebra scripts in the Auxiliary Materials [1].

Stochastic Automata Collectives

With those aims, we investigate *stochastic automata collectives*. By a *collective*, we mean a *large set of interacting*, *finite state* automata. This is not quite the situation we have in classical automata theory, because we are interested in automata interactions. It is also not quite the situation with cellular automata, because our automata are interacting, but not necessarily on a regular grid. And it is not quite the situation in process algebra, because we are interested in the behavior of collectives, not of individuals. And in contrast to Petri nets, we model separate parts of a system separately. Similar frameworks have been investigated under the headings of collectives [24], sometimes including stochasticity [13]. The broad area of computer network analysis is also relevant; see [9] for a bridge between that and stochastic automata.

By *stochastic*, we mean that automata interactions have rates. These rates induce a quantitative semantics for the behavior of collectives, and allow them to mimic chemical kinetics. Chemical systems are, physically, formed by the *stochastic* interactions of *discrete* particles. For large number of particles, it is usually possible to consider them as formed by *continuous* quantities that evolve according to *deterministic* laws, and to analyze them by ordinary differential equations (ODEs). However, one should keep in mind that continuity is an abstraction, and that sometimes it is not even a correct limit approximation. In biochemistry, the stochastic discrete approach is particularly appropriate because cells often contain very low numbers of molecules of critical species: that is a situation where continuous models may be misleading. Stochastic automata collectives are hence directly inspired by biochemical systems, which are sets of interacting macromolecules, whose stochastic behavior ultimately derives from molecular dynamics. Some examples of the mismatch between discrete and continuous models are discussed at the end of Sect. 3.

Paper Outline

In Sect. 2, we introduce the notion of stochastic interacting automata. In Sect. 3, we explore a number of examples inspired by chemical kinetics, leading to the implementation of basic analog and digital devices. In Sect. 4, we describe more sophisticated automata, which are more suitable for biochemical modeling. In Sect. 5, we discuss further related work, and conclude. The figures are in color (the layout and the narrative offer color-neutral hints, but we encourage reading the digital version of this paper). Auxiliary Materials [1] include magnifiable figures and the editable simulation scripts that generated the figures.

2 Interacting Automata

2.1 Automata Reactions

We begin by focusing on the notion of stochastic interacting automata and their collective behavior. Figure 1 shows a typical situation. We have three separate automata species A, B, C, each with three *possible states* (circles) and each with a *current state* (yellow): initially A_1, B_1, C_1, respectively. *Transitions* change the current state of an automaton to one of its possible states: they are drawn as thick gray arrows (solid or dashed). *Interactions* between separate automata are drawn as thin red dashed arrows. *Collectives* consist of populations of automata, e.g., $100 \times A$, $200 \times B$ and $300 \times C$.

There are two possible kinds of *reactions* that can cause an automaton to take a transition and change its current state; each reaction changes the situation depicted on the left of Fig. 2 to the situation on the right, within some larger context. First, from its current state, an automaton can spontaneously execute a *delay* transition (dashed gray arrow) resulting in a change of state of that automaton. Second, an automaton can jointly execute an *interaction* (thin red dashed arrow) with a separate automaton. In an interaction, one automaton executes an *input* (?), and the other an *output* (!) on a common *channel* (a). A channel is an abstraction (just a name) for any interaction surface or mechanism, and input/output

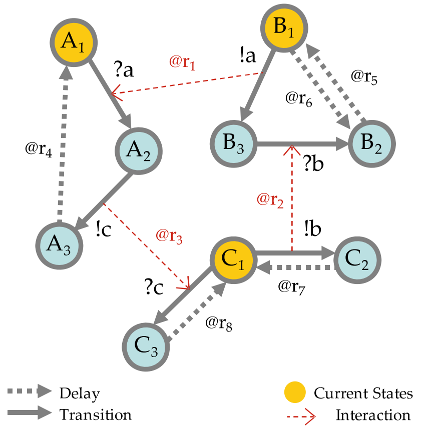
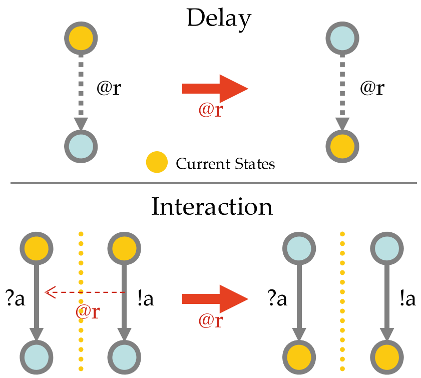


Fig. 1 Interacting automata

Fig. 2 Automata reactions



are abstraction for any kind of interaction complementarity. An actual interaction can happen only if both automata are in a current state such that the interaction is enabled along complementary transitions (solid gray arrows); if the interaction happens, then both automata change state simultaneously. The system of automata in Fig. 1, for example, could go through the following state changes: $A_1, B_1, C_1 \rightarrow_{(a)} A_2, B_3, C_1 \rightarrow_{(b)} A_2, B_2, C_2 \rightarrow A_2, B_2, C_1 \rightarrow_{(c)} A_3, B_2, C_3 \rightarrow A_1, B_2, C_3 \rightarrow A_1, B_2, C_1 \rightarrow A_1, B_1, C_1$.

Each reaction fires at (@) a rate r. In the case of interaction, the rate is associated to a channel (i.e., interactions have rates, but input/output transitions do *not* have rates of their own). Reaction rates determine, stochastically, the choice of the next reaction to execute, and also determine the time spent between reactions [5, 16]. In particular, the probability of an enabled reaction with rate r occurring within time t is given by an exponential distribution $F(t) = 1 - e^{-rt}$ with mean $1/r$. A Continuous Time Markov Chain (CTMC) can be extracted from an automata collective [2]: a state of such a CTMC is a multiset of the automata current states for the population, and a transition in the CTMC has a rate that is the sum of the rates of all the reactions connecting two states.

2.2 *Groupies and Celebrities*

In the rest of this section, we explore a little zoo of simple but surprising automata collectives, before beginning a more systematic study in Sect. 3. We usually set our reaction rates to 1.0 (and in that case omit them in figures) not because rates are unimportant, but because rich behavior can be obtained already by changing the automata structure: rate variation is a further dimension of complexity. The 1.0 rates still determine the pacing of the system in time.

The automaton in Fig. 3 has two possible states, A and B. A single automaton can perform no reaction, because all its reactions are interactions with other automata. Suppose that we have two such automata in state A; they each offer $!a$ and $?a$, hence they can interact on channel a, so that one moves to state B and the other one moves

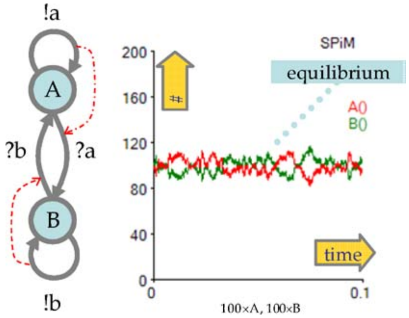


Fig. 3 Celebrity automata

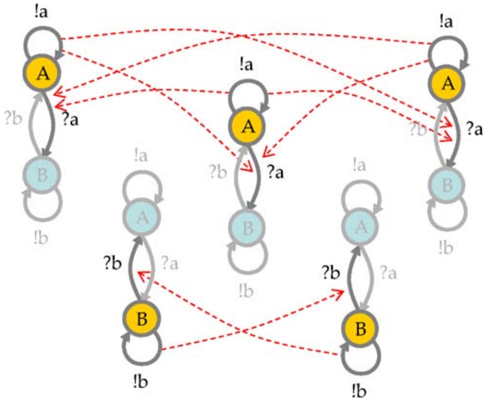


Fig. 4 Possible interactions

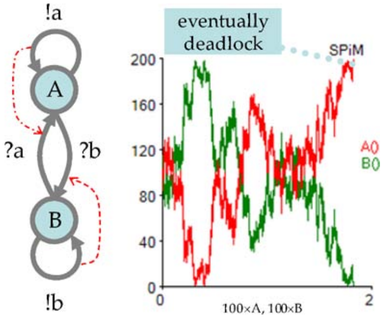
back to state A (either one, since there are two possible symmetric reactions). If we have two automata in state B, one will similarly move to state A. If we have one in state A and one in state B, then no interactions are possible and the system is stable.

We call such automata *celebrities* because they aim to be different: if one of them "sees" another celebrity in the same state, it changes state. How will a population of celebrities behave? Starting with 100 celebrities in state A and 100 in state B, the stochastic simulation in Fig. 3 (obtained by the techniques in [16] and Auxiliary Materials [1]) shows that a 50/50 noisy equilibrium is maintained. Moreover, the system is live: individual celebrities keep changing state.

The possible interactions between celebrities are indicated in Fig. 3 by thin-red-dashed *interaction arrows* between transitions on the same automaton. Remember, however, that an automaton can never interact with itself: this abuse of notation refers unambiguously to interactions between distinct automata in a collective. The possible interactions should more properly be read out from the complementary transition labels. The transition labels emphasize the *open system* interactions with all possible environments, while the interaction arrows emphasize the *closed system* interactions in a given collective.

Figure 4 shows more explicitly the possible interactions in a population of 5 celebrity automata, of which 3 are in state A and 2 are in state B. Note that since there are more "a interactions" than "b interactions," and their rates are equal, at

Fig. 5 Groupie automata



this point in time an "*a* interaction" between a pair of automata is more likely. These considerations about the likelihood of interactions are part of the stochastic simulation algorithm, and also lead to the chemical *law of mass action* for large populations [25].

Let us now consider a different two-state automaton shown in Fig. 5. Again, a single automaton can do nothing. Two automata in state A are stable since they both offer $!a$ and $?b$, and no interactions are possible; similarly, for two automata in state B. If we have one automaton in state A and one in state B, then they offer $!a$ and $?a$, so they can interact on channel a and both move to state A. They also offer $?b$ and $!b$, so they can also interact on channel b and both move to state B.

We call such automata *groupies* because they aim to be similar: two groupies in different states will switch to equal states. How will a population of groupies behave? In Fig. 5, we start with 100 A and 100 B: the system evolves through a bounded random walk, and the outcome remains uncertain until the very end. Eventually, though, the groupies form a single homogeneous population of all A or all B, and the system is then dead: no automaton can change state any further. Different runs of the simulation may randomly produce all A or all B: the system is *bistable*.

2.3 Mixed Populations

Populations of groupies and populations of celebrities have radically different behavior. What will happen if we mix them? It is sufficient to mix a small number of celebrities (1 is enough) with an arbitrarily large number of groupies, to achieve another radical change in system behavior. As shown in Fig. 6, the groupies can still occasionally agree to become, e.g., all A. But then a celebrity moves to state B to differentiate itself from them, and that breaks the deadlock by causing at least one groupie to emulate the celebrity and move to state B. Hence, the whole system now evolves as a bounded random walk with no stable state.

An important lesson here is that an arbitrarily small, but nonzero, number of celebrities can transform the *macroscopic* groupie behavior from a system that always eventually deadlocks, to a system that never deadlocks. We can also replace

Fig. 6 Both together

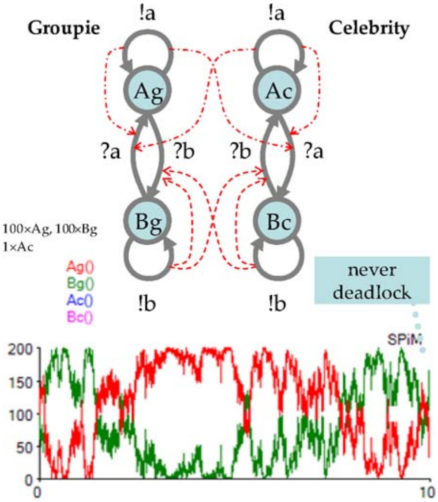
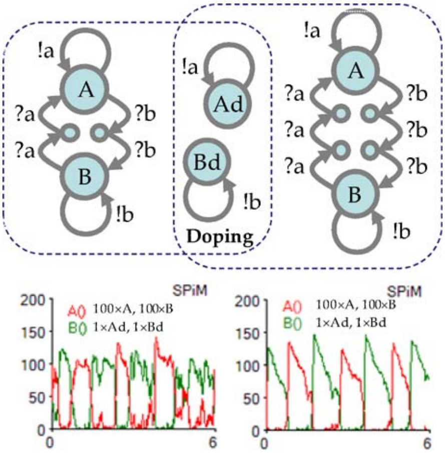


Fig. 7 Hysteric groupies



celebrities with simpler *doping* automata (Fig. 7) that have the same effect of destabilizing groupie collectives.

We now change the structure of the groupie automaton by introducing intermediate states on the transitions between A and B, while still keeping all reactions rates at 1.0. In Fig. 7, each groupie in state A must find two groupies (left) or three groupies (right) in state B to be persuaded to change to state B. Once started, the transition from A to B is irreversible; hence, some hysteresis (history dependence) is introduced. Both systems include doping automata (center) to avoid population deadlocks.

The additional intermediate states produce a striking change in behavior with respect to Fig. 6: from complete randomness to irregular (left) and then regular (right) oscillations. The peaks in the plots of Fig. 7 are stochastic both in height and in width, and occasionally one may observe some miss-steps, but they clearly alternate. The transformation in behavior, obtained by changes in the structure of individual automata, is certainly remarkable (and largely independently on rate values). Moreover, the oscillations depend critically on the tiny perturbations introduced by doping: without them, the system typically stops on its first cycle.

The morale from these examples is that the collective behavior of even the simplest interactive automata can be rich and surprising. Macroscopic behavior "emerges" apparently unpredictably from the structure of the components, and even a tiny number of components can have macroscopic effects. The question then arises, how can we relate the macroscopic behavior to the microscopic structure? In the following section, we continue looking at examples, many of which can be analyzed by continuous methods. We return to discussing the behavior of groupies at the end of Sect. 3.

3 The Chemistry of Automata

3.1 Concentration

The first few chapters of a chemical kinetics textbook [10] usually include a discussion of the *order* of a chemical reaction. In particular, a reaction of the form $A \rightarrow^r C$ is first-order, and a reaction of the form $A + B \rightarrow^r C$ is second-order. The terminology derives from the order of the polynomials of the associated differential equations. In the first case, the *concentration* of A, written $[A]$, follows the exponential decay law $d[A]/dt = -r[A]$, where the right-hand side is a first-order term with coefficient $-r$ (r being the base reaction rate). In the second case, the concentration of A follows the mass action law $d[A]/dt = -r[A][B]$, where the right-hand side is a second-order term. (In the sequel, we use $[A]^\bullet$ for $d[A]/dt$.)

Our automata collectives should match these laws of chemical kinetics, at least in large number approximations. But what should be the meaning of "a concentration of discrete automata states" that would follow such laws? That sounds puzzling, but is really the same question as the meaning of "a concentration of discrete molecules" in a chemical system. One has to first fix a volume of interaction (assumed filled with, e.g., water), and then divide the discrete number of molecules by that volume, obtaining a concentration that is regarded (improperly) as a continuous quantity. Along these lines, a relationship between automata collectives and differential equations is studied formally in [2].

Under biological conditions, common simplifying assumptions are that the volume of interaction is constant that the volume is filled with a liquid solution with constant temperature and pressure, and that the solution is *dilute* and *well mixed*. The dilution requirement is a limit on maximum chemical concentrations: there should be enough water that the collisions of chemicals with water are more frequent than among themselves. The well-mixed requirement means that diffusion effects are not important. Together, these physical assumptions justify the basic mathematical assumption: the probability of any two non-water molecules colliding (and reacting) in the near future is independent of their current position, so that we need to track only the number (or concentration) of molecules, and not their positions. Three-way collisions are too unlikely to matter [5].

Therefore, we assume that our automata interact within a fixed volume V, and we use a scaling factor $\gamma = N_A V$, where N_A is, in chemistry, Avogadro's number. For most purposes, we can set $\gamma = 1.0$, which means that we are considering a volume of size $1/N_A$ liters. However, keeping γ symbolic helps in performing any scaling of volume (and hence of number of automata) that may be needed.

With these basic assumptions, we next analyze the reaction orders that are available to our collectives, and the effect of reaction order on kinetics.

3.2 First-Order Reactions

As we have seen, an automaton in state A can spontaneously move to state A' at a specified rate r, by a stochastic delay. In a population of such automata, each transition decrements the number of automata in state A, and increments the number of automata in state A'. This can be written also as a chemical reaction $A \to^r A'$ (Fig. 8), with first-order rate law $-r[A]$, where $[A]_t = \#A_t/\gamma$ is the concentration of automata in state A at time t, and $\#A_0$ is the initial number of automata in state A. The concentration $[A]$ is a continuous quantity, and is more properly related to the *expectation* of the discrete number $\#A$ having a certain value [25]. The rate of change for the reaction (assuming $A' \neq A$) is then the derivative of $[A]$, written $[A]^\bullet = -r[A]$. The profile of the reaction is an exponential decay at rate $r : [A]_0 e^{-rt}$ (see curve S_1 in Fig. 9).

A sequence of exponential decays produces an Erlang distribution, as seen in Fig. 9 (many biological processes, like the sequential transcription and translation of DNA, behave similarly). Initially, we have $C = 10000$ automata in state S_1. The occupation of the initial state S_1 is an exponential decay, the occupation of the intermediate states S_i is the Erlang distribution of shape parameter i, and the occupation of the final state is the cumulative Erlang distribution of shape parameter 10.

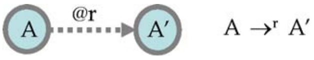


Fig. 8 First-order reactions

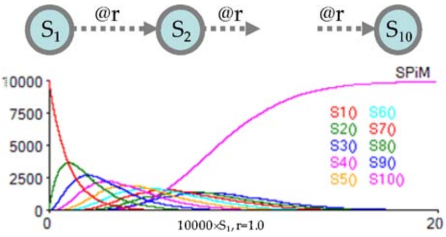


Fig. 9 Sequence of delays

The shape of an exponential distribution is independent of the initial quantity (e.g., the half-life is constant). In general, for first-order reactions, the time course of the reactions is independent of the scaling of the initial quantities. For example, if we start with 10 times as many automata in Fig. 9 and we scale down the vertical axis by a factor of 10, we obtain the same plot, to the original time 20. Meaning that the “speed of the systems” is the same as before, and since there are 10 times more reactions in the same time span, the “execution rate” is 10 times higher.

3.3 Second-Order Reactions

As discussed in Sect. 2, two automata can interact to perform a joint transition on a common channel, each changing its current state. The interaction is synchronous and complementary: one automaton in state A performs an input $?a$ and moves to state A'; the other automaton in state B performs an output $!a$ and moves to state B'.

This interaction can be written as a chemical reaction $A + B \rightarrow^{r\gamma} A' + B'$ (Fig. 10, top), where r is the fixed rate assigned to the interaction channel, and $r\gamma$ is the volume scaling for $A + B$ reactions [25] (to scale the volume γ to $n \times \gamma$, we must scale $\#X_0$ to $n \times \#X_0$ to keep $[X]_0$ the same, and r to r/n to keep rates $(r/n)n \times \gamma$ the same). The rate law, given by the law of mass action, is $-r\gamma[A][B]$, because each automaton in the population of current states $[A]$ can interact with each automaton in the population of current states $[B]$, and the derivatives (assuming A, B, A', B' are distinct) are $[A]^\bullet = [B]^\bullet = -r\gamma[A][B]$.

A different situation arises, though, if the interaction happens within a homogeneous population, e.g., when state A offers both an input $?a$ to transition to state A' and an output $!a$ to transition to state A'' (Fig. 10, bottom). Then every automaton in state A can interact with every other automaton in state A in *two* symmetric ways; hence, the rate r must be doubled to $2r$. The volume scaling for $A + A$ reactions is $(2r)\gamma/2 = r\gamma$ [25]. The chemical reaction is then $A + A \rightarrow^{r\gamma} A' + A''$, whose rate law is $-r\gamma[A]^2$. The rate of change of $[A]$ (assuming $A' \neq A \neq A''$) is $[A]^\bullet = -2r\gamma[A]^2$, since two A are lost each time.

In Fig. 11, we show an automaton that exhibits a first-order reaction and one of each kind of second-order reactions. Its collective behavior is determined by the corresponding chemical reactions. This shows that the dynamics of all orders of reactions can be intermingled in one automaton.

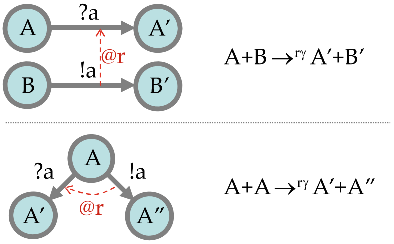


Fig. 10 Second-order reactions

Fig. 11 All 3 reactions

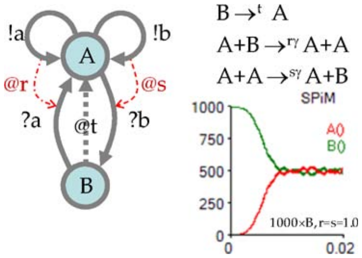


Fig. 12 Same behavior

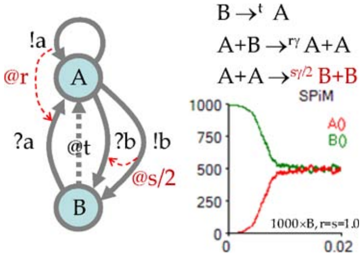
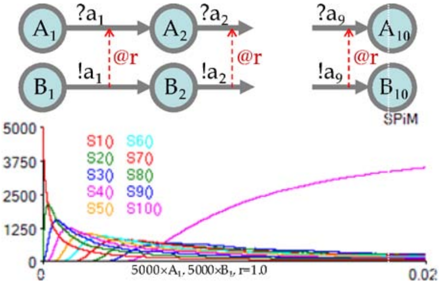


Fig. 13 Sequence of interactions



In order to compare the behavior of different automata collectives, we must in general go beyond the corresponding chemical reactions, and we must instead compute the corresponding ODEs (which can be obtained from the chemical reactions). For example, the automaton in Fig. 12 has a different pattern of interactions and rates, different chemical reactions, but the same ODEs as the one in Fig. 11. In both cases, $[A]^\bullet = -[B]^\bullet = t[B] + r\gamma[A][B] - s\gamma[A]^2$, but note that the b rate in Fig. 12 is set to $s/2$ in order to obtain the same rate of decrease in A population and rate of increase in B population as in Fig. 11, given the differences in the corresponding chemical reactions.

The time course of second-order reactions decreases linearly with the scaling up of the initial quantities. For example, if we start with 10 times as many automata as in Fig. 13, and we scale down the vertical axis by a factor of 10, and we scale up the time axis by a factor of 10, we obtain the same plot. Meaning that the "speed of the system" is 10 times faster than before, and since there are also 10 times more reactions, the "execution rate" is 100 times higher. Moreover, the system in Fig. 13 is about 500 times faster than the one in Fig. 9 in reaching 75% of input level in its final state, even though it has the same rates and number of automata. Second-order reactions “go faster.”

3.4 Zero-Order Reactions

As we have just seen, the two basic kinds of automata interactions, which correspond to the two basic kinds of chemical reactions, lead to collective dynamics characterized by various kinds of curves. But what if we wanted to build a population that spontaneously (from fixed initial conditions) increases or decreases at a constant rate, as in Fig. 14? We can consider this as a programming exercise in automata collectives: *can we make straight lines*? This question will lead us to implementing a basic analog component: a signal comparator.

First-order reactions have a law of the form $r[A]$, and second-order reactions a law of the form $r[A][B]$. Zero-order reactions are those with a law of the form r (a constant derivative), meaning that the “execution rate” is constant, and hence the “speed of the system” gets slower when more ingredients are added. Zero-order reactions are not built into chemistry (except as spontaneous creation reactions $0 \to^r A$), but can be approximated by chemical means. The main biochemical methods of obtaining zero-order reactions are a special case of enzyme kinetics when enzymes are saturated.

Real enzyme kinetics corresponds to a more sophisticated notion of automata: both are treated in Sect. 4. For now, we discuss a close analog of enzyme kinetics that exhibits zero-order behavior and can be represented within the automata framework described so far. We will make precise how this is a close analog of enzymes, and in fact, with a few assumptions, it can be used to model enzyme kinetics in a simplified way.

Consider the system of Fig. 14. Here, E is the (*pseudo*-) enzyme, S is the substrate being transformed with the help of E, and P is the product resulting from the transformation. The state *ES* represents an enzyme that is “temporarily unavailable” because it has just transformed some S into some P, and needs time to recover (*ES* does not represent an E molecule bound to an S molecule: it is just a different state of the E molecule alone).

This system exhibits zero-order kinetics (i.e., a constant slope), as can be seen from the plot. If we start with lots of S and a little E, the rate of production of P is constant, independently of the instantaneous quantity of S. That happens because E becomes maximally busy, and effectively processes S sequentially. Adding more

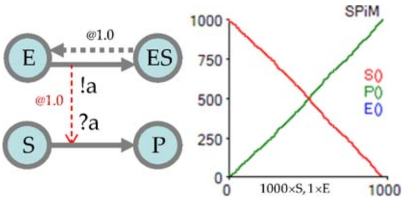


Fig. 14 Zero-order reactions

enzyme (up to a point) will just increase the ES population: to obtain the zero-order behavior it is not necessary to have a single E, just that most E be normally busy. All our rates are 1.0 as usual, but note that $E \rightarrow ES$ happens fast, proportionally to $[E][S]$, while $ES \rightarrow E$ happens slowly, proportionally to $[ES]$.

To explain the reason for the constant slope, and to make the connection to enzyme kinetics precise, we now mimic the standard derivation of Michaelis–Menten kinetics [10]. The reactions for the system in Fig. 14 are

$$\begin{aligned} &E + S \rightarrow^{r\gamma} ES + P, \\ &ES \rightarrow^{s} E. \end{aligned}$$

The corresponding ODEs are

$$\begin{aligned} &[E]^{\bullet} = s[ES] - r\gamma[E][S], \\ &[ES]^{\bullet} = r\gamma[E][S] - s[ES], \\ &[S]^{\bullet} = -r\gamma[E][S], \\ &[P]^{\bullet} = r\gamma[E][S]. \end{aligned}$$

We call $[E_0] = [E] + [ES]$ the total amount of enzyme, either free or busy; that is, the concentration of enzyme. We now assume that, in normal operation, the enzyme is at equilibrium, and in particular $[ES]^{\bullet} = 0$. This implies that $[ES] = r\gamma[E][S]/s$. Set:

$$\begin{aligned} &K_m = s/r\gamma, \\ &V_{\max} = s[E_0]. \end{aligned}$$

Hence, $[ES] = [E][S]/K_m$, and $[ES] = ([E_0] - [ES])[S]/K_m$, and from that we obtain $[ES] = [E_0]([S]/(K_m + [S]))$. From the $[ES]^{\bullet} = 0$ assumption, we also have $[P]^{\bullet} = s[ES]$, and substituting $[ES]$ yields

$$[P]^{\bullet} = V_{\max}[S]/\bigl(K_m + [S]\bigr),$$

which describes $[P]^{\bullet}$ just in terms of $[S]$ and two constants. Noticeably, if we have $K_m \ll [S]$, then $[P]^{\bullet} \approx V_{\max}$; that is, we are in the zero-order regime, with constant growth rate $V_{\max} = s[E_0]$. For the system of Fig. 14 at $\gamma = 1$, we have $K_m = 1$, $[S]_0 = 1000$, $V_{\max} = 1$, and hence $[P]^{\bullet} \approx 1$, as shown in the simulation.

The chemical reactions for Fig. 14 are significantly different from the standard enzymatic reactions, where P is produced after the breakup of ES, and not before as here. Still, the expressions for K_m, $V_{\max}$, and $[P]^{\bullet}$ turn out to be the same as in Michaelis–Menten kinetics (and not just for the zero-order case), whenever the dissociation rate of ES back to $E + S$ is negligible, that is, for good enzymes.

3.5 Ultrasensitivity

Zero-order kinetics can be used, rather paradoxically, to obtain sudden nonlinearity or switching behavior. Let us first compare the behavior of directly competing enzymes in zero-order and second-order kinetics. For conciseness, we now depict a pair of states E, ES (as in Fig. 14) as a single state E with a solid/dashed arrow representing the transition through the now hidden state ES (Fig. 15).

At the top of Fig. 15, in zero-order regime, a fixed quantity of F is competing against a linearly growing quantity of E. The circuit is essentially computing the subtraction $[F] - [E]$ (the rest being sequestered in the hidden "unavailable" states): the E quantity is neutralized until it can neutralize and then exceed the F quantity. At the bottom, we have almost the same system, except in second-order regime: the result of the competition is quite different because neither quantity can be sequestered.

On that basis, we now reproduce the peculiar phenomenon of ultrasensitivity in zero-order regime [14], confirming that our simplified kinetics, while not agreeing with enzyme kinetics at the microscopic level, still manages to reproduce some of its macroscopic effects: the core of the matter is the zero-order regime of operation. In an ultrasensitivity situation, a minor switch in relative enzyme quantities creates a much amplified and sudden switch in two other quantities. In Fig. 16, we start with a fixed amount (=100) of enzyme F, which is holding the S vs. P equilibrium in the S state (at $S = 1000$), and we let E grow from zero. As E grows, we do not initially observe much free E, but the level of free F decreases (as in Fig. 15 top), indicating that it is getting harder for F to maintain the S equilibrium against E.

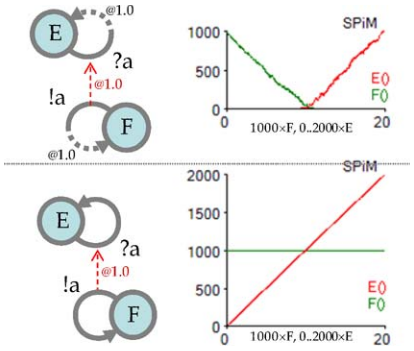


Fig. 15 Subtraction (top)

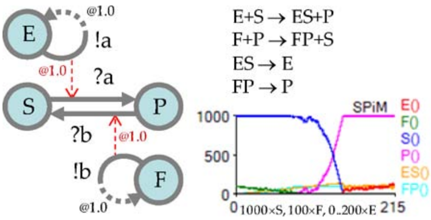


Fig. 16 Ultrasensitivity

Eventually, the level of free F drops to zero, at which point we see a sudden switch of the S vs. P equilibrium, and then we observe the level of free E growing. Hence, in this case, a switch in the levels of E vs. F controls a factor of 10 bigger switch in the levels of S vs. P. If P is itself an enzyme, it can then cause an even bigger and even more sudden switch of an even larger equilibrium.

Therefore, we have obtained a device that can compare the relative levels of two slowly varying weak signals (E, F), and produce a strong, quickly-switching output signal that means $E > F$.

3.6 Positive Feedback Transitions

We now come to another programming exercise in automata collectives. None of the curves that we have seen so far are symmetric; we can then ask: *can we make a symmetric bell shape* as in Fig. 18? This question will lead us to building another basic analog component: an oscillator.

A theorem of probability theory guarantees that we can in fact approximate any shape we want by combining exponential distributions, but the resulting automata would normally be huge. Here, we are looking for a compact programming solution. One way to obtain a sharp raising transition (the first half of a bell curve) is by positive feedback. In Fig. 17, the more B's there are, the faster the A's are transformed into B's, so the B's accumulate faster and faster, up to saturation. (Note that at least one B is needed to bootstrap the process.)

By linking two such transitions in series (Fig. 18), we obtain a symmetrical bell shape. We can see that after the B's start accumulating, they are being drained faster and faster by the accumulating C's. The fact that the B's are being drained in a symmetrical way can be explained by the kinetics of B: $[B]^\bullet = [B]([A] - [C])$. (Note that there is a very small chance that the B's will be drained by the C's before the wave can accumulate in B, therefore, stalling it.)

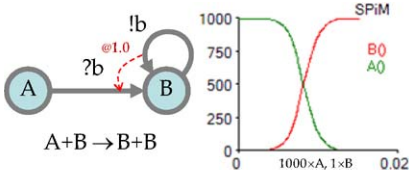


Fig. 17 Positive feedback transition

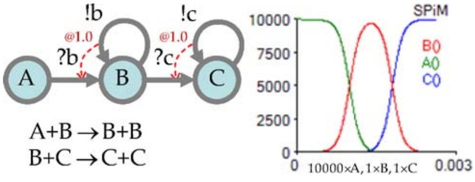


Fig. 18 Bell shape

Fig. 19 Oscillator

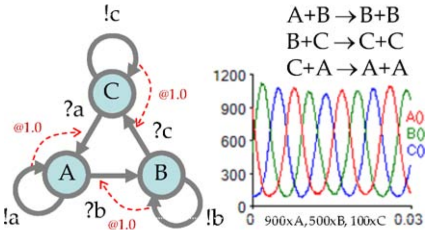


Fig. 20 Positive two-stage feedback

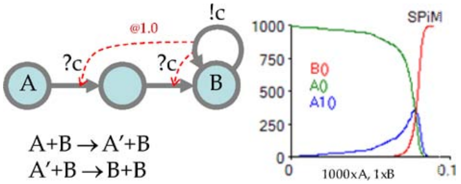
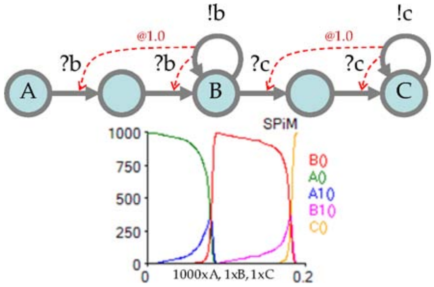


Fig. 21 Square shape



Linking several such transitions in series (not shown) produces a soliton-like wave that does not dissipate (a delay transition between adjacent states is needed to prevent stalling).

Linking three positive feedback transitions in a loop produces a stochastic oscillator (Fig. 19); moreover, the ODEs extracted from the chemical reactions describe a (never stopping) deterministic oscillator. A sustained stochastic oscillation can be obtained by starting with all states nonzero; the oscillation can then survive as long as no state A, B, C goes to zero; when that happens (usually after a long time) there is nothing to pull on the next wave, and the oscillation stops.

An interesting variation is a two-stage positive feedback loop (Fig. 20) where the drop of state A is delayed and the growth of state B is steeper. Joining two such transitions (Fig. 21) produces a shape that approximates a rectangular wave as we increase the cardinality of A. Linking three such transitions in a loop produces again an oscillator (Fig. 22). However, this time it is critical to add doping because each state regularly drops to zero and needs to be repopulated to start the next propagation. This oscillator is a 3-states version of the oscillators in Fig. 7, and is very robust.

Fig. 22 Hysteric 3-way groupies

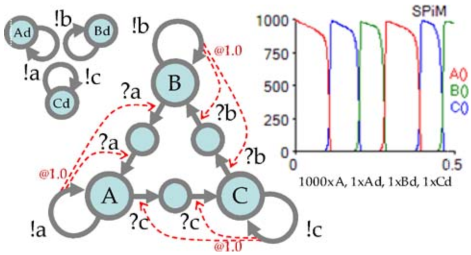
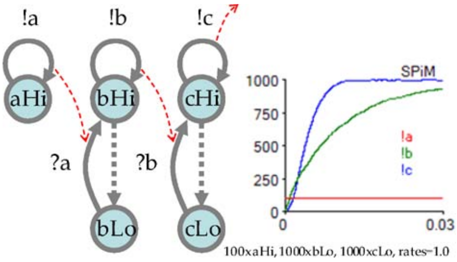


Fig. 23 Second-order cascade



3.7 Excitation Cascades

Beyond signal comparators and oscillators, other basic analog devices, we may want to build include signal amplifiers and dividers. We next imitate some amplifiers found in biological systems that are made of cascades of simpler stages. As a technical note, the modular nature of these staged amplifiers leads us to write compact, parametric simulation code that is instantiated at each stage. This compactness is not reflected in the figures, where we simply redraw each stage. But, as a consequence, it is more convenient in the simulation code to plot the counts of active outputs, e.g., $!a$, $!b$, $!c$, rather than the counts of the states that produce those outputs, e.g., aHi, bHi, cHi: this plotting style is adopted from now on.

We consider cascades where one enzyme activates another enzyme. A typical situation is shown in Fig. 23 (again, all rates are 1.0), where a low constant level of first-stage aHi results in a maximum level of third-stage cHi, and where, characteristically, the third stage raises with a sigmoidal shape, and faster than the second-stage level of bHi. This network can be considered as the skeleton of a MAPK cascade, which similarly functions as an amplifier with three stages of activation, but which is more complex in structure and detail [11].

The resulting amplification behavior, however, is nonobvious, as can be seen by comparison with the zero-order activation cascade in Fig. 24; the only difference there is in the zero-order kinetics of the enzymes obtained by introducing a delay of 1.0 after each output interaction. Within the same time scale as before, the level of cHi raises quickly to the (lower) level of aHi, until aHi is all bound. On a much longer time scale, cHi then grows linearly to maximum. Linear amplification in cascades has been attributed to negative feedback [22], but apparently can be obtained also by zero-order kinetics. Of course, the behavior in Fig. 23 is the limit of that in Fig. 24, as we decrease the zero-order delay.

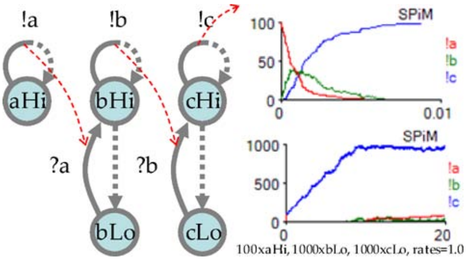


Fig. 24 Zero-order cascade

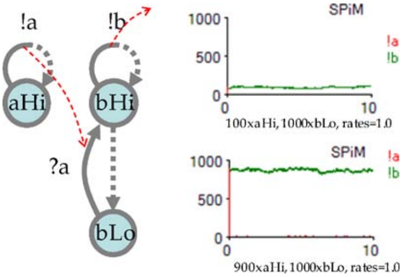


Fig. 25 Zero-order cascade-1 stage

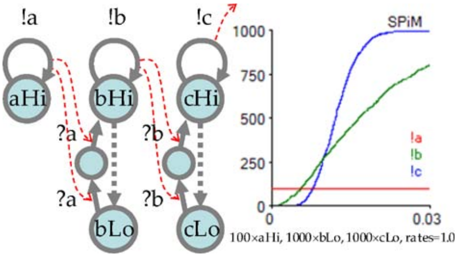


Fig. 26 Second-order double cascade

A single stage of a second-order cascade works like the bHi level shown in Fig. 23 (since no bHi is actually consumed by the next stage), that is, as an amplifier. Surprisingly, a single stage of the zero-order cascade, works quite differently, as a signal replicator. In Fig. 25, a given level of aHi (=100 or =900), induces an equal level of bHi, as long as it is lower than the reservoir of bLo (=1000). (If aHi exceeds bLo, then $bHi := bLo$ and $aHi := aHi - bLo$.) However, the two-stage cascade in Fig. 24 does not work like two signal replicators in series. This seems to happen because the bHi are not available for degradation to bLo while bound by interaction with the next stage, cLo, and hence can accumulate.

Real MAPK cascades are actually based on double activation, as shown in Fig. 26, where the sigmoid output is more pronounced and delayed than in Fig. 23. And once again, the zero-order regime brings surprises: the cascade in Fig. 27 works in reverse, as a signal attenuator, where even a very high amount of aHi produces a low stable level of cHi which is at most $1/3$ of maximum cHi. This is because one stage of such a cascade is actually a signal level divider, where if $bHi = 1000$ then $cHi = 333$, with the signal being distributed among the three states of the stage.

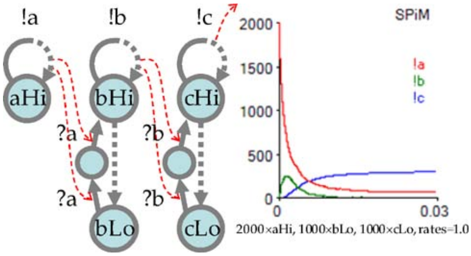


Fig. 27 Zero-order double cascade

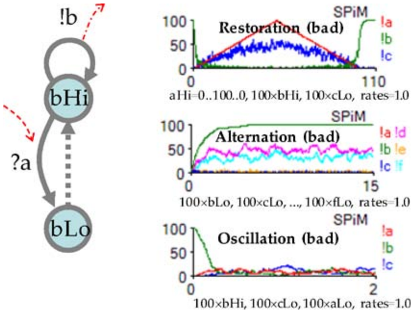


Fig. 28 Simple inverter

3.8 Boolean Inverters

Having seen how to obtain basic analog functions (comparators, oscillators, amplifiers, and dividers) it is now time to consider digital devices: inverters, and other Boolean gates. Automata with distinguished "low" and "high" states can be used to represent respectively Boolean *false* and *true*.

We begin by investigating automata collectives that implement Boolean inverters. The most obvious inverter, $\mathrm{Inv}(a, b)$ with input $?a$ and output $!b$, is shown in Fig. 28: its natural state is high because of the spontaneous decay from low to high. The high state sustains (by a self loop) an output signal (b) that can be used as input to further gates. A high input signal (a) pulls the high state down to low, therefore, inverting the input. (Steady state analysis of the ODEs shows that $[bHi] = \max/(1 + \gamma[aHi])$, where $\max = [bHi] + [bLo]$.)

We test the behavior of populations of inverters according to three quality measures, which are first applied to the inverter in Fig. 28:

(1) Restoration. With two inverter populations in series ($100 \times \mathrm{Inv}(a, b) + 100 \times \mathrm{Inv}(b, c)$), a triangularly-shaped input signal ($?a$) is provided that ramps up from 0 to 100 and then back down to 0.

(2) Alternation. We test a connected sequence $100 \times \mathrm{Inv}(a, b) + 100 \times \mathrm{Inv}(b, c) + \cdots + 100 \times \mathrm{Inv}(e, f)$.

(3) Oscillation. We test a cycle of three populations, $100 \times \mathrm{Inv}(a, b) + 100 \times \mathrm{Inv}(b, c) + 100 \times \mathrm{Inv}(c, a)$.

In Fig. 28, top plot, we see that the first stage inverter is very responsive, quickly switching to low $!b$ output and then quickly switching back to high $!b$ output when

Fig. 29 Feedback inverter

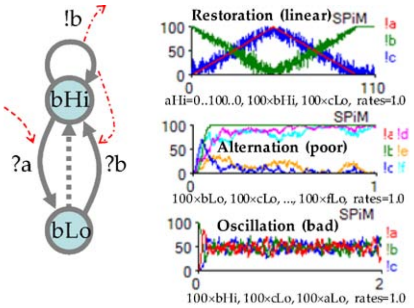
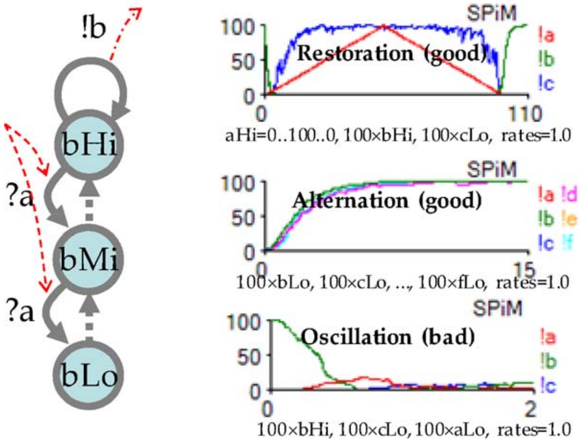


Fig. 30 Double-height inverter



the input is removed, but the second stage $!c$ output is neither a faithful reproduction nor a Boolean restoration of the $?a$ input. In the middle plot, we see that intermediate signals in the alternation sequence are neither high nor low: the Boolean character is lost. In the bottom plot, we see that three gates in a loop fail to sustain a Boolean oscillation. Therefore, we conclude that this inverter does not have good Boolean characteristics, possibly because it reacts too strongly to a very small input level, instead of switching on a substantial signal.

In an attempt to force a Boolean behavior, we add a positive feedback to the high state, so that (one might think) a higher input level would be required to force switching, hence improving the Boolean switching characteristics (Fig. 29). The result is, unexpectedly, a linear signal inverter. (We can deduce the linearity from the steady state analysis of the ODEs: $[bLo] = [aHi][bHi]/([bHi] + 1/\gamma) \approx [aHi]$ for $[bHi] \gg 1/\gamma$, hence $[bHi] = \max -[bLo] \approx \max -[aHi]$.) Such a linear inverter can be useful for inverting an analog signal, and also has decent Boolean alternation properties. But it does not oscillate.

A good Boolean inverter can be obtained instead by doubling the height of the simple inverter (Fig. 30). This double height inverter gives perfect alternation, and good restoration (transforming a triangular input, $!a$ into a nearly rectangular output, $!c$). However, it still fails to oscillate.

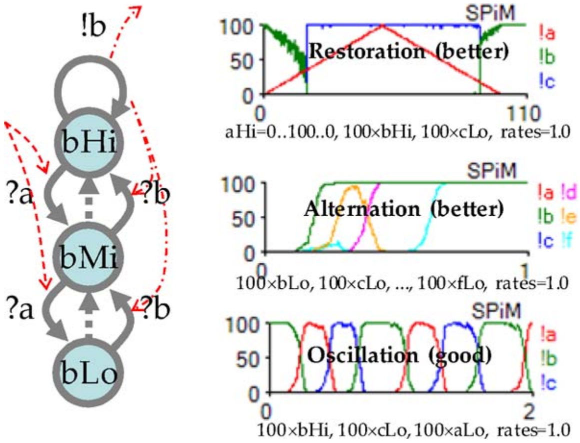


Fig. 31 Double-height feedback

Finally, we combine the two techniques in a double-height feedback inverter (Fig. 31). This has perfect restoration, transforming a triangular input into a sharp rectangle. It also has strong and quickly achieved alternation, and regular oscillation.

In conclusion, it is possible to build good Boolean inverters and signal restorers. This is important because it lessens the requirements on other circuits: if those circuits degrade signals, we can always restore a proper Boolean signal by two inverters in series. We examine some Boolean circuits next.

3.9 Boolean Circuits

It probably seems obvious that we can build Boolean circuits out of populations of automata, and hence support general computation. However, it is actually surprising, because finite populations of interacting automata are *not* capable of general computation [3], and only by using stochasticity one can approximate general computation up to an arbitrarily small error [23]. Those are recent results, and the relationship with the Boolean circuits shown here is not clear; likely one needs to use sufficiently large populations to reduce computation errors below a certain level.

We again consider automata with low states and high states to represent respectively Boolean *false* and *true*. In general, to implement Boolean functions, we also need to use intermediate states and multiple high and low states.

Figure 32 shows the Boolean gate automata for "$c = a$ Or b" And "$c = a$ and b". The high states spontaneously relax to low states, and the low states are driven up by other automata providing inputs to the gates (not shown). A self-loop on the high states provides the output. In the plots, two input signals that partially overlap in time are used to test all four input combination; their high level is just $1/10$ of max (where max is the number of gate automata).

The chemical reactions for the Or gate are $aHi + cLo \to^{\gamma} aHi + cHi$, $bHi + cLo \to^{\gamma} bHi + cHi$, $cHi \to cLo$. From their ODEs, by setting derivatives to zero, and with the automata constraint $[cHi] + [cLo] = \max$, we obtain $[cHi] =$

Fig. 32 Or and And

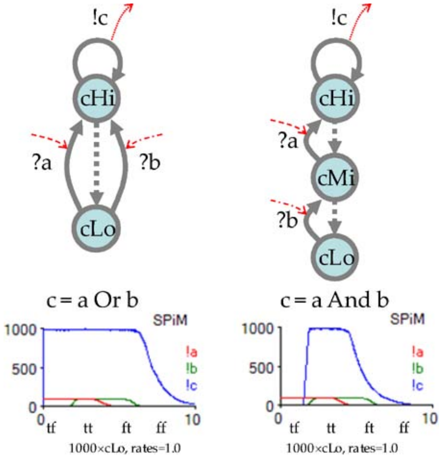
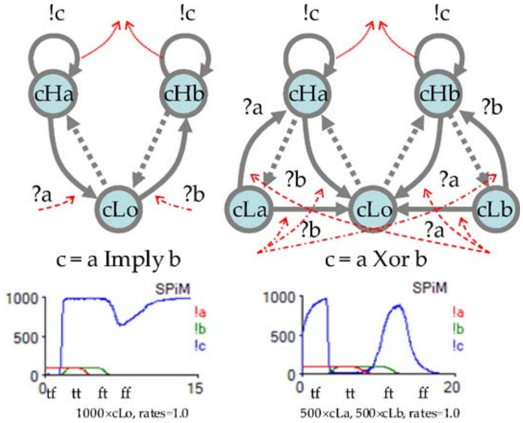


Fig. 33 Imply and Xor



$\max([aHi]+[bHi])/([aHi]+[bHi]+1/\gamma)$. That is, if the inputs are zero, then $[cHi]=0$. If, say, $[aHi]$ is nonzero, then $[cHi]=\max[aHi]/([aHi]+1/\gamma)$ so that for $[aHi]\gg 1/\gamma$ we have $[cHi]\approx \max$.

The And gate can be similarly analyzed. Note that And is not perfectly commutative because the decay back to cLo on a single input is slightly different depending on which input is provided. However, as usual, any analog implementation of a digital gate must be given enough time to stabilize.

Figure 33 shows automata for "$c = a$ Imply b" and "$c = a$ Xor b". In these automata, we use two high states (both producing the same output) to respond to different inputs. The dip in the plot for Imply arises when many automata decay from the high state cHb to the high state cHa, through cLo, in a transition from *false* Imply *true* to *false* Imply *false*.

The steady state behavior of Imply is: output $= [cHa] + [cHb] = \max - \max[aHi]/([aHi][bHi]+[aHi]+1/\gamma)$ where max is the size of the collective. If

$[aHi] = 0$, we have output = max; if $[aHi] \neq 0$ and $[bHi] = 0$ we have output ≈ 0; if $[aHi] \approx [bHi] \approx$ max, we have output $\approx$ max.

Although Xor can be constructed from a network of simpler gates, the Xor gate in Fig. 33 is implemented as a single uniform collective.

3.10 Bistable Circuits

We have now assembled a collection of basic analog and digital devices that can be used to perform signal processing and combinatorial computation. For completeness, we need to discuss also memory elements; these can be constructed either as bistable digital circuits (flip-flops), using the gates of Sects. 3.8 and 3.9, or as bistable analog devices.

We have already seen examples of bistable analog devices: the ultrasensitive comparator of Sect. 3.5, and the groupies of Sect. 2.2. The groupies, however, are not stable under perturbations: any perturbation that moves them away from one of the two stable states can easily cause them to wander into the other stable state. Hence, they would not be very good as stable memory elements.

In Fig. 34, we show a modified version of the groupies, obtained by adding an intermediate state shared by the two state transitions. This automaton has very good memory properties. The top-left and top-center plots show that it is, in fact, spontaneously bistable. The bottom-left plot shows that it is stable in presence of sustained 10% fluctuations produced by doping automata. The bottom-center plot shows that, although resistant to perturbations, it can be switched from one state to another by a signal of the same magnitude as the stability level: the switching time is comparable to the stabilization time. In addition, this circuit reaches stability 10 times faster than the original groupies: the top-right plot shows the convergence times of 30 runs each of the original groupies with 2 states, the current automaton with 3 states, and a similar automaton (not shown) with 4 states that has two middle states in series. The bottom-right plot is a detailed view of the same data, showing that the automaton

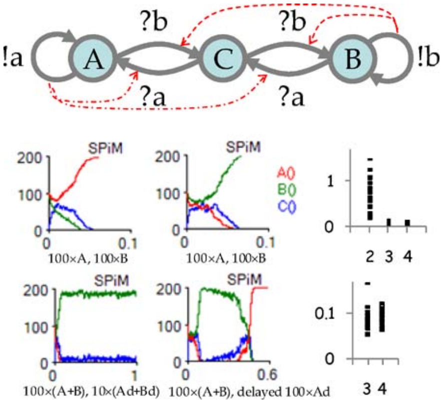


Fig. 34 Memory elements

with 4 states is not significantly faster than the one with 3 states. Therefore, we have a stable and fast memory element.

3.11 Discrete vs. Continuous Modeling

We finally get back to the oscillating behavior of the hysteric groupies of Fig. 7. In the previous sections, we have analyzed many systems both by stochastic simulation and by differential equations, implying that in most cases we have a match between the two approaches, and that we can use whatever analysis is most useful. However, continuous techniques are not always appropriate [2, 25], and there are well-known examples of that [26].

As an illustration of the general issue, in Fig. 35, we compare the stochastic simulation of groupie collectives against numerical solutions of their corresponding differential equations (found in the Fig. 35 scripts in Auxiliary Materials [1]). On the left column, we have the basic groupies from Fig. 5, and on the middle and right columns we have the groupies with one or two intermediate steps from Fig. 7; in all cases, we include a low number of doping automata, as in Fig. 7, to prevent deadlocks. The bottom row has plots of the stochastic simulations, all starting with 2000 automata in state A. (Comparable, noisier simulations with 200 automata are found in the mentioned figures.) The top row has instead the ODE simulations with the same initial conditions (including doping), and with the volume of the solution taken as $\gamma = 1.0$.

As we can see, in the right column, there is an excellent match between the stochastic and deterministic simulations; moreover, the match gets better when increasing the number of molecules, as expected. In the middle column, however, after a common initial transient, the deterministic simulation produces a dampened oscillation, implying that all 4 states are eventually equally occupied. Instead, the stochas-

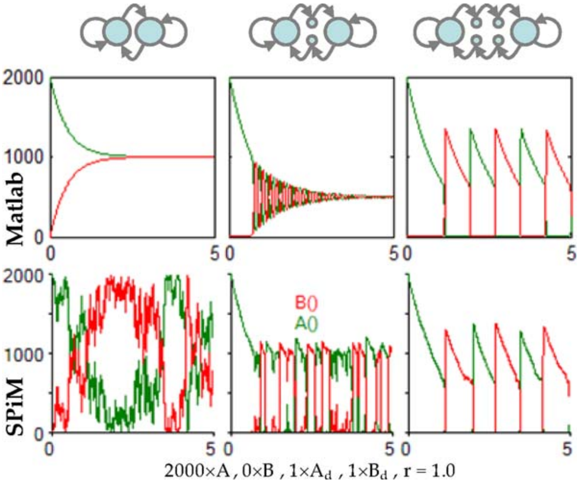


Fig. 35 Discrete vs. continuous modeling

tic simulation produces an irregular but not at all dampened oscillation, where the A and B states persistently peak at twice the continuous values (detailed plots reveal that A and its successor state peak together, and so do B and its successor). Even more strikingly, in the left column, the deterministic simulation produces an equilibrium level, which also happens to be stable to perturbations, while the stochastic simulation produces a completely unstable random walk.

The conclusions one should draw from this situation is that it is not always appropriate to use a deterministic approximation of a stochastic system (and it is likely impossible in general to tell when it is appropriate). The difference can be particularly troublesome when studying just one run of one copy of a stochastic system (e.g., the behavior of one cell), as opposed to the average of a number of runs, or the average of a number of copies. Unfortunately, to understand the behavior of, e.g., cells, one must really study them one at a time. Stochastic behavior can be characterized precisely by other kinds of differential equations (the chemical master equation), which are always consistent with the stochastic simulations, but those are much more difficult to analyze than the mass action ODEs [6].

4 The Biochemistry of Automata

4.1 Beyond Simple Automata

As we have seen, the simple automata of Sect. 2 can model typical chemical interactions. Biochemistry, however, is based on a richer set of molecular interactions, and in this section we explore a corresponding richer notion of interacting automata.

A characteristic feature of biochemistry, and of proteins in particular, is that biological molecules can stick to each other to form *complexes*. They can later break up into the original components, with each molecule preserving its identity. This behavior can be represented by chemical reactions, but only by considering a complex as a brand new chemical species, thus losing the notion of molecular identity. Moreover, *polymers* are formed by the iterated complexation of identical molecules (*monomers*): chemically this can be represented only by an unbounded number of chemical species, one for each length of a polymer, which is obviously cumbersome and technically infinite.

In order to model the complexation features of biochemistry directly, we introduce *polyautomata*, which are automata that can form reversible complexes, in addition to interacting as usual. *Association* (&) represents the event of joining two specific automata together out of a population, and *dissociation* (%) is the event that causes two specific associated automata to break free; both events result in state changes. Association does not prevent an automaton from performing normal interactions or other associations, but it prevents it from reassociating on the same interface, unless it first dissociates. Association and dissociation can be encoded in π-calculus [15] (as shown in Auxiliary Materials [1]: Fig. 36), by taking advantage of its most powerful features: fresh channels and scope extrusion. That encoding

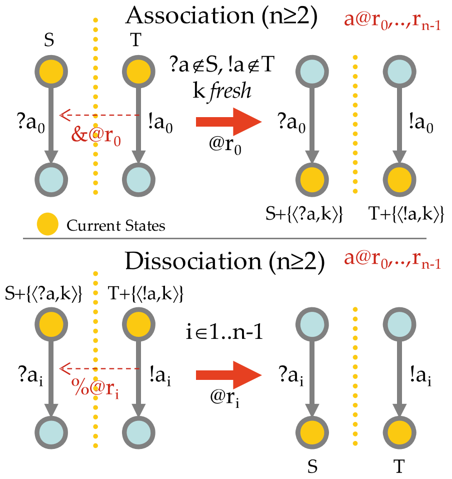


Fig. 36 Polyautomata reactions

results in flexible modeling of complexation [20], but does not enforce constraints on reassociation.

Here, we strive again to remain within the confines of an automata-like framework, including diagrammatic descriptions. A formal presentation of polyautomata is given in [3], where it is shown that they are Turing-complete in conjunction with an unbounded supply of a finite number of monomer species (and that instead, the automata of Sect. 2 can achieve Turing-completeness only with an unbounded supply of *different* species, which means infinite-size programs).

4.2 Polyautomata

Polyautomata are automata with an association history, and with additional kinds of interactions that modify such history. The main formal difference from the automata of Sect. 2 is that the current state now carries with it a set S of *current associations*.

An association is a pair $\langle \pi, k \rangle$ where π is an event label ($?a$ or $!a$ for the complementary sides of an association), and k is a *unique* integer identifying an association event between two automata. We assume that a *fresh* k can be produced from, e.g., a global counter during the evolution of a collective: only two automata should have the same k in their associations at any given time. This unique k is used to guarantee that the *same* two automata that associated in the past will dissociate in the future.

There can be multiple ways of disassociating two automata after a given association; the rates of association and of each possible disassociation can differ. Therefore, each channel will now be attributed with a list of one or more rates: this is written $a@r_0, \ldots, r_{n-1}$ for $n \geq 1$. We then say that $\text{arity}(a) = n$.

If $\text{arity}(a) = 1$, then r_0 is called the *interaction rate*, because it covers the old case of ordinary interactions: the old interaction rules from Fig. 2 apply with $r_0 \equiv r$, with the understanding that the association sets are unaffected. If $\text{arity}(a) \geq 2$, then

r_0 is the *association rate*, and $r_1, \ldots, r_{n-1}$ are the *dissociation rates*. The association rules from Fig. 36 then apply.

An association (Fig. 36 top) on a channel *cannot* happen if an automaton has a past association event on that channel, as recorded in the current state; that is, that particular "surface patch" of the automaton is currently occupied and cannot be reused. The preconditions $?a \notin S$ (short for $\langle ?a, k\rangle \notin S$ for any k) and $!a \notin T$, check for such conflicts, where S and T are the sets of associations. If a new association is possible, then a fresh integer k is chosen and stored in the association sets after the transition. The transition labels are $?a_0$ and $!a_0$, indicating an association at rate r_0 on channel a. In examples, we use instead the notation $\&?a$ and $\&!a$ for these labels, where & indicates association, omitting index 0.

Symmetrically, a dissociation (Fig. 36 bottom) on a channel happens only if the two automata have a current association on that channel, as identified by the same k in their current states. If a dissociation is possible, the corresponding associations are removed from the association sets after the transition (+ here is disjoint union), enabling further associations. The transition labels are $?a_i$ and $!a_i$ with $i \in 1, \ldots, n-1$, indicating a dissociation at rate r_i on channel a. In examples, we use the notation $\%?a_i$ and $\%!a_i$ for these labels, where % indicates dissociation; if $\text{arity}(a) = 2$, then we write simply $\%?a$ and $\%!a$, omitting index 1.

4.3 Complexation

As an example of the association/dissociation notation, in Fig. 37, we consider two automata that cyclically associate, moving to *bound* states A_b, B_b, and then dissociate, moving back to *free* states A_f, B_f. We also show the association sets under each state, although the number k can change at each iteration. The yellow cartoon shapes illustrates the mechanics of complexation, where complexation channels are depicted as complementary surfaces. The plot shows that for the chosen rates, the dynamic equilibrium is heavily biased toward the bound states. (In this section, a state A_f corresponds to the plot line $!A\_f$.)

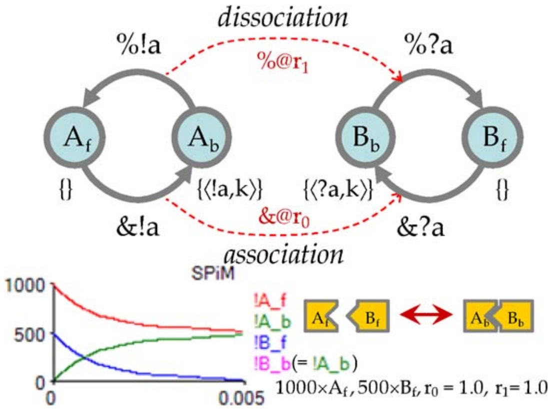


Fig. 37 Complexation/decomplexation

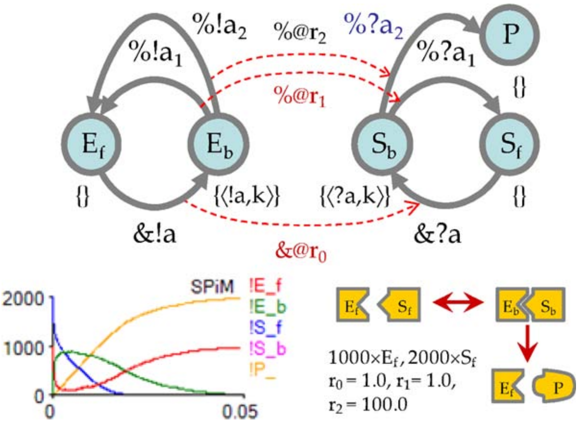


Fig. 38 Enzymatic reactions

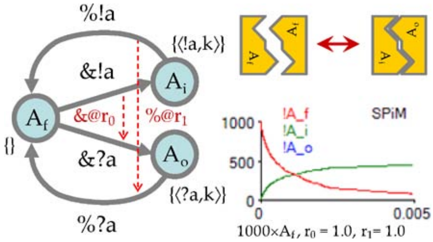


Fig. 39 Homodimerization

The use of multiple dissociation rates is exemplified by enzymatic reactions (Fig. 38): these are now the true enzymatic reactions, not the ones from Sect. 3.4 [20]. From the bound state of enzyme (E_b) and substrate (S_b), two dissociations are possible with the one at higher rate producing product (P).

More subtle forms of complexation exist. Homodimerization (Fig. 39) is symmetric complexation: a monomer offers both an input and an output complexation on the same channel, meaning that it offers two complementary surfaces, and can stick to a copy of itself. Note that a monomer here cannot bind to two other monomers over its two complementary surfaces. That situation leads to polymerization, as shown next.

4.4 Polymerization

A *polymer* is obtained by the unbounded combination of *monomers* out of a finite set of monomer shapes. There are many forms of polymerization; here, we consider just two basic linear ones.

In linear bidirectional polymerization, each monomer can join other monomers on one of two complementary surfaces, without further restrictions. Therefore, two polymers can also join in the same way, and a single polymer can form a loop

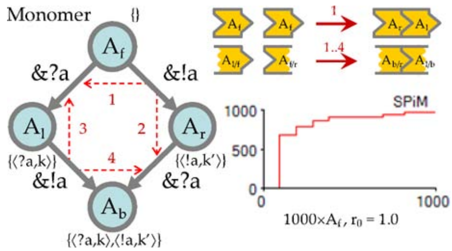


Fig. 40 Bidirectional polymerization

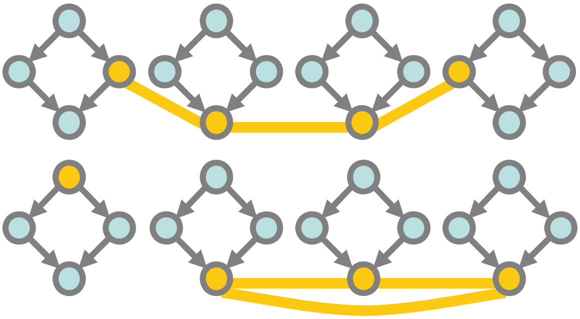


Fig. 41 Automata polymers

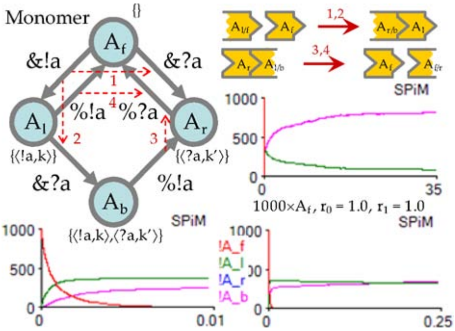
(although a single monomer cannot). For simplicity, we do not allow these polymers to break apart.

In Fig. 40, we show a monomer automaton that can be in one of four states: A_f (free), A_l (bound on the left), A_r (bound on the right), and A_b (bound on both sides, with two association events). The sequence of transitions is from free, to bound on either side, to bound on both sides.

There are four possible input/output associations between two monomers, indicated by the red dashed arrows in the figure. Number 1 is the association of two free monomers in state A_f: one becomes bound to the left (A_l) and the other bound to the right (A_r). Number 2 is the association of a free monomer with the leftmost monomer of a polymer (a monomer bound to the right): the free monomer becomes bound to the right and the leftmost monomer becomes bound on both sides (A_b). Number 3 is the symmetric situation of a free monomer binding to the right of a polymer. Number 4 is the leftmost monomer of a polymer binding to the rightmost monomer of another polymer (or possibly of the same polymer, forming a loop, as long as the two monomers are distinct). Figure 41 is a schematic representation of some possible configurations of monomers, with thick lines joining their current states and representing their current associations.

The plot in Fig. 40 shows the result of a fairly typical simulation run with 1000 monomers. When all the monomers are fully associated, we are left with a number of circular polymers: the plot is obtained by scanning the circular polymers after they stabilize. The horizontal axis is discrete and counts the number of such polymers (9 in this case). Each vertical step corresponds to the length of one of the circular polymers (polymers are picked at random for plotting: the vertical steps are not

Fig. 42 Actin-like polymerization



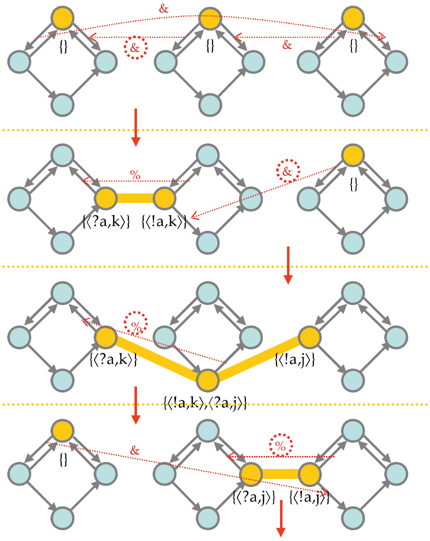
sorted by size). It is typical to find one very long polymer in the set (∼800 in this case), and a small number of total polymers (<10).

We now consider a different form of polymerization, inspired by the actin biopolymer, which can grow only at one end and shrink only at the other end. In Fig. 42, we have the same four monomer states, but the sequencing of transitions is different. There are four possible input/output associations between two monomers, indicated by the red dashed arrows in the figure. Number 1 is the association of two free monomers in states A_f: one becomes bound to the left (A_l) and the other bound to the right (A_r). Number 4 is the breakup of a polymer made of just two monomers, one that is bound to the left and one that is bound to the right; they both return free. Number 2 is the association of a free monomer with the rightmost monomer of a polymer (a monomer bound to the left): the free monomer becomes bound to the left and the rightmost monomer becomes bound on both sides (A_b). Number 3 is the dissociation of a monomer bound to the right, from the leftmost monomer to its right which is bound on both sides; one becomes a free monomer and the other remains bound to the right. Loops cannot form here, because if we have a monomer bound to the left and one bound to the right (which could be the two ends of the same polymer), then there is no interaction that can make them bound on both sides.

The plots in Fig. 42 show three views of the same simulation run with 1000 monomers, at times 0.01, 0.25, and 35; all rates are 1.0. During an initial quick transient the number of A_b and of $A_l = A_r$ temporarily stabilize, each approaching level 333 (with average polymer length 3). A_b crosses over at time 0.02 and then slowly grows until $A_l = A_r = 100$ around time 35; hence, the final number of polymers is ∼100 with average length ∼10.

Figure 43 shows in detail a typical sequence of interactions among three monomers, with two associations followed by two dissociations. At each step, we show the possible interactions by dashed red arrows connecting the enabled transitions. The thick lines indicate the current associations, which are actually encoded in the association sets shown under the current states, by the shared k and j. Note that in state A_b, the association set has the form $\{\langle !a, k\rangle, \langle ?a, j\rangle\}$. This illustrates the need to store $!a$ and $?a$ separately in the history: if we recorded only the channel, $\langle a, k\rangle$, then the second association for $\langle a, j\rangle$ would be prevented because the set would already contain the channel a. And if we modified the occurrence check to

Fig. 43 Typical monomer interactions



allow storing distinct pairs $\langle a, k\rangle$, $\langle a, j\rangle$, this would allow arbitrary reassociations on the same channel.

In conclusion, polyautomata provide a relatively simple graphical notation for representing combinatorial systems of interacting *and* complexing molecules. Systems that grow without bounds by complexation can be represented compactly and finitely: this would not be possible using the automata of Sect. 2 or, in fact, using chemical reactions.

5 Conclusions and Related Work

Despite ongoing conscious efforts, biochemistry is still lacking an adequate notation for describing large and complex biological models in a compositional, parameterizable, and scalable way [12]. *Formal notations* (such as programming languages and process algebras) are fundamental tools for achieving those goals: they enable engineering and analysis techniques that are orthogonal to the ones available by *mathematical models* (such as set theory, calculus, and Markov chains). In computing, adequate notation is key to the maintainability of large information processing systems consisting of millions of lines of code, whose complexity is dwarfed only by biological systems. Noticeably, information processing systems are not written using differential equations, nor set theory, because those are not useful tools in that domain.

Still, it is always important to relate formal notation to mathematical models. We have used automata notation for exploring simple, but intriguing biochemical systems, aiming to demonstrate how easy it is to "play with" the notation to get insights into a system. We have shown, by example, how to relate the interacting automata notation to stochastic behavior and to differential equations. For a full development, the process algebra foundations for this work are found in [2], which connects the stochastic π-calculus approach to modeling biochemistry [20], to stochastic and deterministic chemical kinetics [25]. Additionally, foundations for Sect. 4 are found in [3].

The use of compositional, graphical, formal notation has been long advocated [7], but most graphical notations in systems biology still lack such fundamental properties. Our automata are at least compositional, but are neither parameterizable nor scalable, unless they are embedded in the richer framework of process algebras, which has all such properties. Our diagrams are based on previous work on the graphical representation of the whole π-calculus [17]. Dealing with the full π-calculus, however, means having to graphically represent, in general, bound variables. The full π-calculus also seems excessive for use in biochemical models, both semantically and graphically. Therefore, while the diagrams in Sects. 1–3 are essentially a reduced version of the ones in [17], the diagrams in Sect. 4 opt to use operators instead of bound variables to deal with the important biochemical operation of complexation, and also enforce an invariant against reassociation of occupied sites.

The pragmatics of graphical and formal notation for the biochemical domain still requires investigation. It has taken decades to develop adequate notations and analysis techniques for large software and hardware systems; we are just at the beginning to do the same for biochemical systems, where the task will certainly be much harder.

Acknowledgements Much of this material was initially prepared for a doctorate course of the same title at the University of Trento, in May 2006, and improved by referees.

References

1. Auxiliary materials for this article. http://lucacardelli.name
2. Cardelli L (2008) On process rate semantics. Theor Comput Sci 391(3):190–215. http://dx.doi.org/10.1016/j.tcs.2007.11.012
3. Cardelli L, Zavattaro G (2008) On the computational power of biochemistry. In: Third international conference on algebraic biology, AB 2008, July 2008, Linz
4. Fontana W, Buss LW (1996) The barrier of objects, from dynamical systems to bounded organization. In: Casti J, Karlqvist A (eds) Boundaries and barriers. Addison–Wesley, Reading, pp 56–116
5. Gillespie DT (1977) Exact stochastic simulation of coupled chemical reactions. J Phys Chem 81:2340–2361
6. Jahnke T, Huisinga W (2007) Solving the chemical master equation for monomolecular reaction systems analytically. Math Biol 54(1):1–26
7. Harel D (1987) Statecharts: a visual formalism for complex systems. Sci Comput Program 8:231–274
8. Harel D, Efroni S, Cohen IR (2003) Reactive animation. In: Proceedings of FMCO 2002. LNCS, vol 2852. Springer, Berlin, pp 136–153
9. Hillston J (1996) A compositional approach to performance modelling. Cambridge University Press, Cambridge

10. House JE (2007) Principles of chemical kinetics. Academic Press, New York
11. Huang C-YF, Ferrell JE Jr (1996) Ultrasensitivity in the mitogen-activated protein cascade. Proc Natl Acad Sci USA 93:10078–10083
12. Kitano H (2003) A graphical notation for biochemical networks. BioSilico 1(5):169–176
13. Lerman K, Galstyan A (2004) Automatically modeling group behavior of simple agents. In: Agent modeling workshop, AAMAS-04, New York
14. Meinke MH, Bishops JS, Edstrom RD (1986) Zero-order ultrasensitivity in the regulation of glycogen phosphorylase. Proc Natl Acad Sci USA 83:2865–2868
15. Milner R (1999) Communicating and mobile systems: the π-calculus. Cambridge University Press, Cambridge
16. Phillips A, Cardelli L (2004) A correct abstract machine for the stochastic pi-calculus. In: Proceedings of BioConcur'04
17. Phillips A, Cardelli L, Castagna G (2006) A graphical representation for biological processes in the stochastic pi-calculus. Trans Comput Syst Biol VII:123–152
18. Priami C (1995) Stochastic π-calculus. Comput J 38:578–589
19. Priami C, Regev A, Shapiro E, Silverman W (2001) Application of a stochastic name-passing calculus to representation and simulation of molecular processes. Inf Process Lett 80:25–31
20. Regev A, Shapiro E (2004) The π-calculus as an abstraction for biomolecular systems. In: Ciobanu G, Rozenberg G (eds) Modelling in molecular biology. Springer, Berlin, pp 219–266
21. Regev A, Shapiro E (2002) Cellular abstractions: cells as computation. Nature 419:343
22. Sauroa HM, Kholodenko BN (2004) Quantitative analysis of signaling networks. Prog Biophys Mol Biol 86:5–43
23. Soloveichik D, Cook M, Winfree E, Bruck J (2007) Computation with finite stochastic chemical reaction networks. http://www.dna.caltech.edu/DNAresearch_publications.html
24. Tumer K, Wolpert D (2004) A survey of collectives. In: Collectives and the design of complex systems. Springer, Berlin, pp 1–42
25. Wolkenhauer O, Ullah M, Kolch W, Cho K (2004) Modelling and simulation of intracellular dynamics: choosing an appropriate framework. IEEE Trans Nanobiosci 3:200–207
26. Vilar JMG, Kueh HY, Barkai N, Leibler S (2002) Mechanisms of noise-resistance in genetic oscillators. Proc Natl Acad Sci 99(9):5988–5992

Process Calculi Abstractions for Biology

**Maria Luisa Guerriero, Davide Prandi,
Corrado Priami, and Paola Quaglia**

Abstract Several approaches have been proposed to model biological systems by means of the formal techniques and tools available in computer science. To mention just a few of them, some representations are inspired by Petri nets theory and others by stochastic processes.

A most recent approach consists in interpreting living entities as terms of process calculi, by composition of a few behavioural abstractions. This paper comparatively surveys the state of the art of the process calculi approach to biological modelling.

The modelling features of a set of calculi are tested against a simple biological scenario, and available extensions and tools are briefly commented upon.

1 Introduction

The recent progress of biology is rapidly producing a huge number of experimental results and it is becoming impossible to coherently organise them using only human capacity. Abstract models to reason about biological systems are becoming an indispensable conceptual and computational tool for biologists, so calling for computer science.

Research at the convergence of biology and computer science started by observing that biological molecules in real systems participate in very complex networks, like regulatory networks for gene expression, intracellular metabolic networks, and intra/inter-cellular communication networks. Due to the (relatively) recent studies in molecular biology and in the omics disciplines, there is an accurate description of the fundamental components of living systems, especially of proteins and cells. There is not, however, a complete knowledge on how these individual components are related and interact to form complex systems. Various computational approaches have been developed and used to cope with the complexity of these systems. Some of them are:

- *biochemical kinetic models* (see, e.g. [2, 75, 81]);
- *generalised models of regulation* (see, e.g. [1, 30, 78]);

This work has been partially sponsored by the PRIN 2006 Project BISCA—*Sistemi e calcoli di ispirazione biologica e loro applicazioni* and by the FIRB project RBPR0523C3.

M.L. Guerriero (✉)
Laboratory for Foundations of Computer Science, The University of Edinburgh,
Informatics Forum, 10 Crichton Street, EH8 9AB, Edinburgh, UK
e-mail: mguerrie@inf.ed.ac.uk

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_23, © Springer-Verlag Berlin Heidelberg 2009

- *functional object-oriented databases* (see, e.g. [3, 43, 80]);
- *integrated frameworks with GUI* (see, e.g. [38, 74]);
- *exchange languages* (see, e.g. [44]);
- *formal methods from theoretical computer science* (see, e.g. [31, 41, 59, 60]).

The above proposals aim at constructing models of biological systems to help life scientists in their research, but they do not directly investigate what biology is. A landmark paper by Regev and Shapiro [70] calls for a paradigm change, postulating the design of a *language for the cell*, i.e. of a language that allows the representation of biological theories. A forerunner work by Fontana et al. [32] exploits the λ-calculus for describing natural systems. Later on, process calculi [40, 54, 55, 73], which are languages introduced as foundations for interacting and distributed systems, have emerged as suitable formalisms for reasoning about biology.

The similarity between communicating distributed systems and networks of interacting biological entities is intriguing, and gave rise to a range of biologically inspired calculi. A comprehensive picture of the state of the art, however, is still missing, and this paper tries to overcome this limitation. The task is carried out by presenting a simple scenario coming from the biology of the immune system (Sect. 2), and then showing how this setting can be represented in the considered calculi (Sect. 3). We conclude with some final remarks (Sect. 4).

2 A Simple Biological Scenario

This section introduces the running example used to compare the various process calculi in the biology applicative domain. Since *interaction* is a central notion in process calculi, before describing the biological scenario, we will briefly overview the basics of biochemical interactions.

2.1 Biochemical Interactions

Living entities are constantly crossed by a flow of matter and energy. In this continuous random flow, reactions take place whenever there is a sufficient kinetic energy [77]. For instance, a reaction between molecules A and B in Fig. 1 may occur

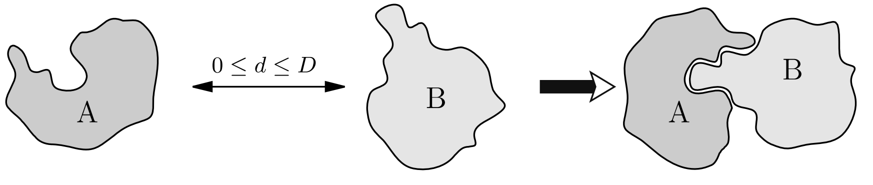


Fig. 1 Molecular interaction

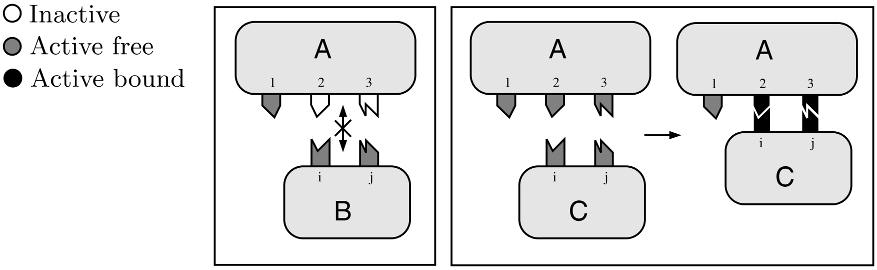


Fig. 2 Interfaces, sites, and states

if A and B are close enough and correctly oriented. Normally, the frequency of reactions is quite low. By need, enzymes may orient molecules in the right way favouring and speeding up reactions [16]. Referring to the example in Fig. 1, we observe that two molecules can bind if they possess complementary zones, called *domains*, and they have the right orientation (or, alternatively, the complementary domains are visible or available to each other).

These conditions, however, are not enough. A domain of a molecule can be either *active* or *inactive*. An inactive domain cannot bind, not even when a complementary domain with the right orientation is close to it. In order to activate a domain, a molecule needs to be involved in some specific reaction, e.g. in a phosphorylation (binding of a phosphate group to the protein).

Concluding, domains can be classified depending on three possible states: active bound, active free, and inactive. Figure 2 shows a schematic representation. Biological entities (named A, B, and C in the picture) possess an *interface* (the rounded box with coloured hooks). Each interface has $n > 0$ sites (the hooks sticking out the rounded box), and each of them can be in one of the three mentioned states (the colour of the hook). A site is an indivisible structure that can only join to a complementary site. In the scenario drawn in Fig. 2, A cannot bind to B. In fact, sites 2 and i, as well as 3 and j, are complementary, but 2 and 3 are both inactive. On the contrary, A and C can bind together: sites i and 2, as well as j and 3, are pairwise complementary, and all of them are active free.

An interesting point is relative to the possibility to dynamically change the number and/or the state of the sites available on a given interface. For instance, as shown below, it might be necessary to be able to add sites.

2.2 The Running Example

We introduce here the running example that will be used to present and compare the considered calculi in the biology applicative domain. The example comes from the biology of the immune system, and it is relative to the activation of the *lymphocyte T helper*. The scenario has two main properties: (i) it is sufficiently complex to be

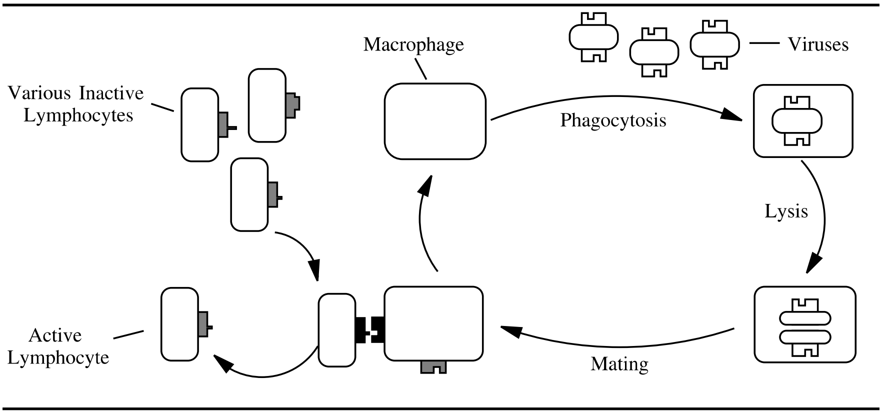


Fig. 3 Lymphocyte T helper activation

an interesting case study for modelling issues; (ii) it is abstract enough to allow us to omit a number of biological details.

Lymphocytes T helper (or helper T cells) are eukaryote cells belonging to our immune system. They play a central role by activating and, so, controlling many specific defence strategies. Lymphocytes are normally inactive, and they start their activity only after being triggered by special events. Here, we will focus on

- the sequence of *phagocytosis—digestion—presentation* phases, and
- the *activation* of lymphocytes T helper performed by macrophages.

Macrophages are cells that engulf a virus (phagocytosis). When this happens, the virus is degraded into fragments (digestion or lysis), and a molecule, the so-called antigen, is displayed on the surface of the macrophage (presentation or mating). The antigen may be recognised by a specific lymphocyte T helper, and this in turn activates the mechanisms of immune reply, a response specific to the recognised virus.

Figure 3 gives an abstract representation of the described phenomenon. Viruses are modelled as entities with inactive sites which represent the viral antigens. The process starts with the phagocytosis of the virus by the macrophage. The virus is then decomposed, and eventually viral antigens are moved to the surface of the macrophage. So, the macrophage acquires some active sites from the virus, and it can wait for a lymphocyte with a complementary site. When the appropriate lymphocyte T helper binds to the macrophage, it becomes active and starts playing its role in the immune reply. Observe that lymphocytes have active sites even before binding to a macrophage.

3 Calculi for Biology

In this section, we survey the main calculi for biology which have been proposed in the literature. We begin by pointing out the general features of process calculi for the description of interacting distributed systems. Then each of the considered calculi for biology is presented and used to model the simple scenario described in Sect. 2. Available simulation and analysis tools are also mentioned and briefly commented upon.

3.1 Process Calculi: The Approach

Starting from the forerunner *Calculus of Communicating Systems* (CCS) [54], process calculi have been defined with the primary goal of providing formal specifications of concurrent processes, namely of computational entities executing their tasks in parallel and able to synchronise over certain kinds of activities. The model of a system is given as a term that defines the possible behaviours of the various components of the system. Calculi are equipped with an *operational semantics* [65], which consists in a set of syntax-driven rules, that allow users to automatically infer the possible future of the system. Usually, the fact that process P evolves into process Q is written as $P \rightarrow Q$.

The basic entities of process calculi are *actions*. In the most basic view (e.g. in CCS), actions are input/output operations on channels. Actions can be composed sequentially, meaning that a process can perform an action after the other. Processes can also be composed in *parallel*: in $P_1 \mid P_2$, where '$|$' is the infix parallel composition operator, processes P_1 and P_2 can either interleave their basic actions or interact. Moreover, most of the calculi are equipped with some sort of *restriction* operator which fixes the *scope* of actions. For instance, assuming a to be an action, and using 'ν' to denote restrictions, $P_1 \mid \nu a P_2$ means that action a is private to process P_2.

The interaction policy assumed by each calculus is probably its main distinguishing feature, and it drives the design of its primitive operators. Here is a(n incomplete) list of possibilities.

- *Two-way synchronisation* (*see, e.g.* [54]). Each action has a complementary action, typically called co-action. Actions and co-actions can synchronise with each other. So, for instance, assuming a and $\overline{a}$ to be complementary actions, the following interaction would be possible: $a.P_1 \mid \overline{a}.P_2 \rightarrow P_1 \mid P_2$ where '.' is the action-prefix operator.
- *Multi-way synchronisation* (*see, e.g.* [39]). The parallel composition operator is parametric w.r.t. a set of actions (sometimes called *cooperation set*) on which all of the parallel processes are obliged to synchronise. There is no need to resort to complementary actions in this case. For example, $a.P_1 \mid_{\{a\}} a.P_2 \mid_{\{a\}} a.P_3 \rightarrow P_1 \mid_{\{a\}} P_2 \mid_{\{a\}} P_3$. On the other hand, assuming b to be distinct from a, no interaction is possible for $b.P_1 \mid_{\{a\}} a.P_2 \mid_{\{a\}} a.P_3$, because the leftmost component cannot contribute to the synchronisation with the other processes.

Table 1 Process calculi abstraction for systems biology, adapted from [70]

Biology	Process calculi
Entity	Process
Interaction capability	Action name
Interaction	Synchronisation/Communication
Modification/Evolution	State change

- *Name-passing* (*see*, *e.g*. [55]). In this case, actions have either the form $\overline{a}\langle b\rangle$ or the form $a(c)$. The first one stands for "output the channel name b along the channel named a", and the second one stands for "input any name from channel a and then use it instead of the parameter name c". The name-passing interaction policy is a specific instance of the two-way communication paradigm: the actions $\overline{a}\langle b\rangle$ and $a(c)$ are complementary to each other, and they can be involved in an interaction. In this case, more than simple synchronisation occurs: channel names flow from senders to receivers. For instance, $\overline{a}\langle b\rangle.P_1 \mid a(c).P_2 \to P_1 \mid P_2\{b/c\}$, where $\{b/c\}$ denotes the substitution of the free occurrences of c by the actually received name b.

Since names can be transmitted in interactions, the restriction operator plays a special role in name-passing calculi. Restricted names cannot be used as transmission media. They can, however, be used as transmitted data and, once transmitted, they become private resources shared by the sender and the receiver. For example, $(\nu b\overline{a}\langle b\rangle.P_1) \mid a(c).P_2 \to \nu b(P_1 \mid P_2\{b/c\})$, which is an instance of the so-called *scope extrusion*. The peculiarity of this kind of communication has been extensively used in modelling biological behaviours. Since P_1 and $P_2\{b/c\}$ can privately interact over b in $\nu b(P_1 \mid P_2\{b/c\})$, if P_1 and P_2 represent molecules, then the process $\nu b(P_1 \mid P_2\{b/c\})$ can be seen as the complex of these two molecules.

The mentioned operators are those common to various process calculi; in addition to these, each calculus adopts a few specific operators. The complete set of process calculi operators is very small and yet it contains all the ingredients for the description of concurrent systems in terms of their *behaviour*, i.e. of their evolution. Indeed, the operational semantics of process calculi allows for the formal interpretation of the behaviour of a process as a directed graph, called *transition system*. The nodes of the graph represent the processes, and an edge connects P to Q only if P evolves into Q.

Two main properties of process calculi are worth mentioning. First, the behaviour of a complex system is expressed in terms of the behaviour of its components. A model can be designed following a bottom-up approach: one defines the basic operations that each sub-component can perform, and then the full system is obtained by composition of these basic building blocks. This property is called *compositionality*. Second, the rules defining the operational semantics of processes allow for both the automatic generation of the transition system and the simulation of runs of the represented system. So, process calculi are specification languages that can be directly implemented and executed.

Table 1 gives a concise picture of the mapping between biology and process calculi. In the process calculi interpretation, a biological entity (e.g. a protein) is seen as a computation unit (i.e. a process) with interaction capabilities abstracted as action names. Similarly to biological entities, which interact/react through complementary capabilities, processes synchronise/communicate on complementary actions. The modifications/evolutions of molecules after reactions are represented by state changes following communications. The abstraction in Table 1 has four main properties [70]: (i) it captures an essential part of the phenomenon; (ii) it is computable (or, better, it is executable), allowing for computer aided analysis; (iii) it offers a formal framework to reason; and (iv) it can be extended.

In the rest of this section, we will survey the main process calculi proposed for representing biological systems, and we will comment on the way the various languages focus on particular extensions of the abstraction principles in Table 1.

3.2 Biochemical *π*-Calculus

The *biochemical π-calculus* [71] is a name-passing process calculus which extends the π-calculus [55, 73]. Molecules are modelled as processes, and molecular complexes are rendered by parallel compositions of processes sharing private names. Movements between complexes and formations of new complexes are represented as transmissions of private names. Once a complex is formed, its components interact by communicating on complementary sites.

Figure 4 reports a code fragment that specifies the antigen presentation phase. The global system SYS is given by the parallel composition of four processes: VIRUS, MACROPHAGE, TCELL1, and TCELL2. Figure 4 only presents the specifications of the first two elements. Here, we just sketch the intuition of the behaviour of the sub-system given by MACROPHAGE | VIRUS. The restriction on top of each component stands for its enclosing membrane. The macrophage phagocytizes the virus by means of a communication on the public channel Tlr. Operationally, this communication involves the output action $\overline{\mathsf{Tlr}}\langle\mathsf{MemM}\rangle$ and its complementary input action Tlr(y). Its effect is twofold: (i) the restricted name MemM undergoes a scope

System specification

$\mathsf{SYS::= MACROPHAGE \mid VIRUS \mid TCELL1 \mid TCELL2}$

$\mathsf{MACROPHAGE::=}(\nu\mathsf{MemM})(\overline{\mathsf{Tlr}}\langle\mathsf{MemM}\rangle.\ \mathsf{MemM(a)}.\ !\,\overline{\mathsf{a}}\langle\mathsf{str}\rangle)$

$\mathsf{VIRUS::=}(\nu\mathsf{MemV})(\mathsf{Tlr(y)}.\overline{\mathsf{y}}\langle\mathsf{Ant1}\rangle)$

System evolution

$\mathsf{MACROPHAGE \mid VIRUS} \rightarrow$

$(\nu\mathsf{MemM})\ (\ \mathsf{MemM(a)}.!\,\overline{\mathsf{a}}\langle\mathsf{str}\rangle \mid (\nu\mathsf{MemV})(\overline{\mathsf{MemM}}\langle\mathsf{Ant1}\rangle))\rightarrow$

$(\nu\mathrm{MemM})(\ !\,\overline{\mathsf{Ant1}}\langle\mathsf{str}\rangle)$

Fig. 4 Phagocytosis-digestion-presentation in π-calculus

System specification

$\mathsf{SYS::=MACROPHAGE' \mid TCELL1 \mid TCELL2}$

$\mathsf{MACROPHAGE'::=(\nu MemM)\ (!\,\overline{Ant1}\langle str\rangle)}$

$\mathsf{TCELL1::=(\nu MemT1)(Ant1(x).ACTIVITIES)}$

$\mathsf{TCELL2::=(\nu MemT2)(Ant2(x).ACTIVITIES)}$

System evolution

$\mathsf{SYS} \rightarrow$

$\mathsf{MACROPHAGE' \mid (\nu MemT1)(ACTIVITIES) \mid (\nu MemT2)(Ant2(x).ACTIVITIES)}$

Fig. 5 TCELL activation in π-calculus

extrusion and becomes a private resource of both MACROPHAGE and VIRUS (thus, modelling the engulfment of the virus); (ii) the name y in VIRUS is renamed into MemM (modelling the adaptation of the internal machinery of the macrophage to start the lysis). The subsequent communication over the channel MemM is such that Ant1 is transmitted to MACROPHAGE, which in turn can make Ant1 available to the lymphocytes T (either TCELL1 or TCELL2) by means of the last action $!\overline{\mathsf{Ant1}}\langle\mathsf{str}\rangle$. The *bang* operator, '!', allows us to model infinite behaviours. In particular, $!\,\overline{\mathsf{Ant1}}\langle\mathsf{str}\rangle$ behaves as $\overline{\mathsf{Ant1}}\langle\mathsf{str}\rangle(!\overline{\mathsf{Ant1}}\langle\mathsf{str}\rangle)$ and, therefore, MACROPHAGE can activate many TCells expressing Ant1.

Figure 5 shows the implementation of the activation of the appropriate lymphocyte T helper. We assume that the antigen presentation phase has already occurred, and hence that the macrophage is ready to communicate on channel Ant1 with whichever lymphocyte can execute a complementary action on the same channel. In the evolution drawn in Fig. 5, this lymphocyte is TCELL1 which, after the synchronisation on Ant1, can start its activities. Notice that the active form of the macrophage, MACROPHAGE', can activate another TCell because of the bang operator.

The *biochemical stochastic π-calculus* [62, 68] is a stochastic extension of the biochemical π-calculus, which allows modellers to quantitatively describe biological systems. Several variants of the biochemical stochastic π-calculus have been recently proposed (e.g. SPICO [46] and Sp@ [79]), while a graphical representation for the biochemical stochastic π-calculus is described in [63]. There exist implementations of the biochemical stochastic π-calculus that make real in silico experiments possible. Two examples of simulation tools for the biochemical stochastic π-calculus are BioSPI [7] and SPiM [76], both based on Gillespie's stochastic simulation algorithm [33]. In addition to simulation, various methods to analyse π-calculus models have been proposed (e.g. causality and concurrency analysis [19]).

Several complex models of real biochemical systems have been implemented and simulated using these tools. Notably, the simulation of extra-vasation in multiple sclerosis reported in [51] showed to have a sort of predictive flavour: an unexpected behaviour of leukocytes has been guessed by the results of in silico simulations, and a posteriori proved in wet experiments. Other complex systems described in the biochemical stochastic π-calculus include gene regulatory networks (e.g. the control of transcription initiation at the lambda switch [47]), cell cycle control [50], and signalling pathways (e.g. the Rho GTP-binding protein cycle [15] and the FGF signalling pathway [37]). In [17], a model of a full cell with a hypothetical minimal gene set is presented and analysed.

3.3 BioAmbients

BioAmbients [69] is an extension of the biochemical π-calculus enriched with an explicit notion of compartments. Similarly to the biochemical π-calculus, BioAmbients models biochemical interactions as communications on channels, but it focuses on compartments: the location of molecules within specific compartments is considered a key issue for regulatory mechanisms in biological systems. Biomolecular systems are organised in a hierarchical and modular way, and molecules can perform their task only if they are in the right compartment. In BioAmbients, systems are represented as a hierarchy of nested ambients, which represent the boundaries of compartments containing communicating π-calculus like processes.

Three kinds of communication are defined in BioAmbients, depending on the relative position of the communicating processes:

- local, i.e. between two processes in the same ambient,
- s2s, i.e. between two processes located in sibling ambients,
- p2c / c2p, i.e. between processes located in ambients with a parent-child / child-parent relation.

Additional pairs of primitives are provided to represent the movement of ambients:

- enter n / accept n, for entering into an ambient and accepting the entrance, respectively,
- exit n / expel n, for exiting from a containing ambient and expelling a contained ambient, respectively,
- merge+ n / merge- n, for merging two ambients together.

Figure 6 shows a possible specification of the digestion of the virus by the macrophage. The two processes Infect and Digest abstract the infection capability of the virus and the digestion capability of the macrophage, respectively. Virus and macrophage synchronise on channel tlr, and the virus enters the macrophage by an enter / accept pair. Then the macrophage obtains the antigen with a p2c communication on channel tlr, and eventually, it makes the antigen available to lymphocytes T helper.

BioAmbients uses communication channels to implement interfaces of biological entities. Figure 7 reports the BioAmbients implementation of the activation of the lymphocyte T helper. Each lymphocyte reacts to a specific antigen, and it begins

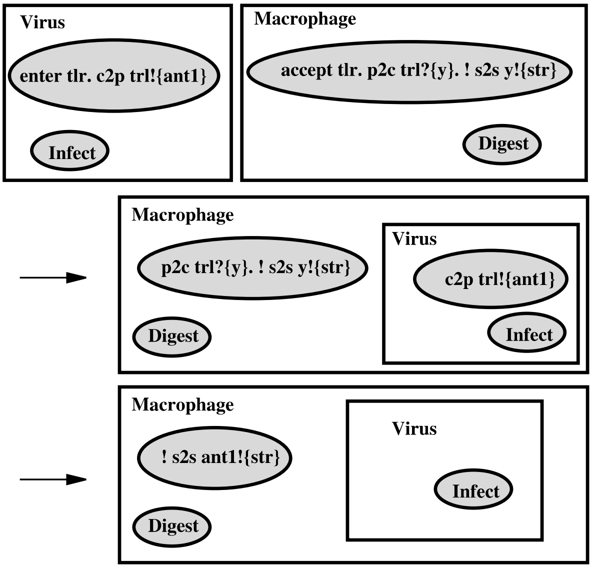


Fig. 6 Phagocytosis-digestion-presentation in BioAmbients

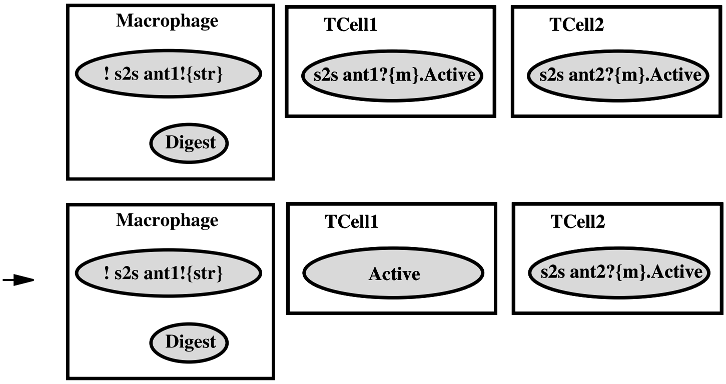


Fig. 7 TCELL activation in BioAmbients

its task by means of a communication on a dedicated channel. After the right lymphocyte has been activated by an s2s communication, the macrophage can activate other TCells (notice the use of the bang operator).

The third version of the BioSpi simulator [7] can handle BioAmbients stochastic simulation. Static analysis techniques that allow users to analyse descriptions of systems to discover dynamic properties have been adapted to BioAmbients (e.g.

control flow analysis [57] and spatial analysis [58]). A symbolic approach which allows modellers to deal with partially unspecified biological systems is presented in [5].

A few biological case studies have been simulated and analysed by means of the BioAmbients calculus (e.g. the FGF endocytotic pathway [4] and the LDL cholesterol degradation pathway [64]).

3.4 Brane Calculi

Brane calculi [14] builds on the observation that membranes are not just containers, but also active entities that take care of coordinating specific activities. This means that membranes are considered as active elements and the whole computation happens *on* membranes. In Brane, calculi membranes can move, merge, split, enter, or exit into/from another membrane. Some constraints, inspired by the real behaviour of biological membranes, need to be satisfied when applying these operations. The most important is continuity of transformations. For instance, a membrane cannot simply pass across another; it rather has to gradually buckle and create a bubble that later on may detach. Another important constraint is the preservation of the orientation of membranes, so merging of membranes cannot occur arbitrarily. For instance, this constraint prevents membranes with a different orientation to merge.

A system is represented as a set of nested membranes, and a membrane as a set of actions. Actions carry out membrane transformations. The primitives related to movement to/from membranes are classified in two main groups.

- Phagocytosis (phago) for engulfing one external membrane, pinocytosis (pino) for engulfing zero external membranes, and exocytosis (exo) for expelling an internal membrane.
- Budding (bud) for splitting off one internal membrane, dripping (drip) for splitting off zero internal membranes, and mating (mate) for the controlled merging of two membranes.

Communication can be *on-membrane* or *cross-membrane*, and it is associated with distinct pairs of primitives.

- *On-Membrane*: the primitives $\mathsf{p2p}_n$ / $\mathsf{p2p}_n^\perp$ are for on-membrane communications only; they follow the π-calculus communication style.
- *Cross-Membrane*: the primitives $\mathsf{s2s}_n$ / $\mathsf{s2s}_n^\perp$, $\mathsf{p2c}_n$ / $\mathsf{p2c}_n^\perp$, and $\mathsf{c2p}_n$ / $\mathsf{c2p}_n^\perp$ are for communications between processes in distinct membranes; they follow the BioAmbients communication style.

Figure 8 reports a specification of the running example in Brane calculi. The operator $\circ$ stands for parallel composition. The rounded parentheses $(\!|\ |\!)$ enclose the content of the membrane. The actions that a membrane can perform are represented at the left of its enclosing rounded parentheses. In the first step, MACROPHAGE engulfs VIRUS by a phago on *trl*. Notice that the INFECT part of VIRUS is now

System specification

SYS::= VIRUS $\circ$ MACROPHAGE $\circ$ TCELL1 $\circ$ TCELL2

VIRUS::= $\mathsf{phago}_{trl}.\ \mathsf{c2p}_n$(ant1).INFECT $\llparenthesis$CAPSID$\rrparenthesis$

MACROPHAGE::= $\mathsf{phago}^{\perp}_{trl}$(DIGEST).$\mathsf{c2p}^{\perp}_n$(a).$\mathsf{s2s}_a$(str) $\llparenthesis$CYTOSOL$\rrparenthesis$

DIGEST::= $\mathsf{c2p}^{\perp}_n$(a).$\mathsf{c2p}_n$(a).DIGEST

System evolution

MACROPHAGE $\circ$ VIRUS $\rightarrow$

$\mathsf{c2p}^{\perp}_n$(a).$\mathsf{s2s}_a$(str) $\llparenthesis$DIGEST $\llparenthesis\mathsf{c2p}_n$(ant1).INFECT $\llparenthesis$CAPSID$\rrparenthesis\rrparenthesis$ $\circ$ CYTOSOL$\rrparenthesis$ $\rightarrow$

$\mathsf{c2p}^{\perp}_n$(a).$\mathsf{s2s}_a$(str) $\llparenthesis\mathsf{c2p}_n$(ant1).DIGEST $\llparenthesis$INFECT $\llparenthesis$CAPSID$\rrparenthesis\rrparenthesis$ $\circ$ CYTOSOL$\rrparenthesis$ $\rightarrow$

$\mathsf{s2s}_{ant1}$(str) $\llparenthesis$DIGEST $\llparenthesis$INFECT $\llparenthesis$CAPSID$\rrparenthesis$ $\rrparenthesis$ $\circ$ CYTOSOL$\rrparenthesis$

Fig. 8 Phagocytosis-digestion-presentation in Brane calculi

System specification

SYS := MACROPHAGE' $\circ$ TCELL1 $\circ$ TCELL2

MACROPHAGE'::= $\mathsf{s2s}_{ant1}$(str).PHAGO $\llparenthesis$CYTOSOL$\rrparenthesis$

TCELL1::= $\mathsf{s2s}^{\perp}_{ant1}$(x).T ACTIVITIES$\llparenthesis$CYTOSOL$\rrparenthesis$

TCELL2::= $\mathsf{s2s}^{\perp}_{ant2}$(x).T ACTIVITIES$\llparenthesis$CYTOSOL$\rrparenthesis$

System evolution

SYS $\rightarrow$ PHAGO$\llparenthesis$CYTOSOL$\rrparenthesis$ $\circ$ T ACTIVITIES$\llparenthesis$CYTOSOL$\rrparenthesis$ $\circ$ TCELL2

Fig. 9 TCELL activation in Brane calculi

inside the DIGEST part of MACROPHAGE, mimicking the real biological behaviour. Then by means of a c2p communication, the macrophage makes *ant1* available to T cells. Availability is shown by the pending s2s communication on *ant1*.

Figure 9 shows the Brane calculi code for the activation of the appropriate lymphocyte. The approach is analogous to the one adopted in the biochemical π-calculus and in BioAmbients: The T cell that knows the right name (in this case *ant1*) can synchronise with MACROPHAGE' and proceed with its activities T_ACTIVITIES.

The projective Brane calculus [27] is a refinement of Brane calculi. Following the observation that biological membranes reactions are directed, Brane calculi actions are replaced by directed actions, so that interaction capabilities are specified as facing inward or outward. This refinement results in an abstraction which is closer to biological settings than the one provided by the original language. In [8], a method for analysing causality in Brane calculi is proposed. The *Brane Logics* [53] allows temporal properties to be expressed and checked.

As for case studies, the Brane calculi description of the LDL cholesterol degradation pathway appeared in [9].

System specification

$$\mathsf{SYS}::=[\mathsf{VIRUS}]_1 \mid [\mathsf{MACROPHAGE}]_2 \mid [\mathsf{TCELL1}]_3 \mid [\mathsf{TCELL2}]_4$$

$$\mathsf{VIRUS}::=\mathsf{Tlr}.\underline{\overline{\mathsf{Ant1}}}.\mathsf{INACT} + \mathsf{INFECT}$$

$$\mathsf{MACROPHAGE}::= \overline{\mathsf{Tlr}}.\mathsf{DIGEST}$$

System evolution

$$\mathsf{SYS} \leftrightarrow {\scriptstyle \langle 2,\mathsf{Tlr},\mathsf{INFECT}\rangle .}\ [\underline{\overline{\mathsf{Ant1}}}.\mathsf{INACT}]_1 \mid {\scriptstyle \langle 1,\mathsf{Tlr},\mathsf{nil}\rangle .}\ [\mathsf{DIGEST}]_2 \mid [\mathsf{TCELL1}]_3 \mid [\mathsf{TCELL2}]_4$$

Fig. 10 Phagocytosis-digestion-presentation in CCS-R

3.5 CCS-R

CCS-R [25] is an extension of CCS that taking into account the fact that most biochemical reactions can be reversed, allows reversibility to be explicitly expressed.

The calculus adopts a two-way synchronisation mechanism. No name is passed in communications, and hence it is not possible to render complex formations by means of transmissions of private names. For this reason, the information flow from the virus to the macrophage in the Phagocytosis-Digestion-Presentation phase cannot be faithfully rendered in CCS-R. It is still possible though to specify this phase as a pathway activation. Such coding is used in the specification of the running example shown in Fig. 10, where the binary *choice* operator '$+$' is used to mean non-determinism between the two operands. Each parallel component of SYS is associated with a numerical identifier, used to allow for consistent backtracking from synchronisations. Let us consider the case when $[\mathsf{VIRUS}]_1$ non-deterministically synchronises with $[\mathsf{MACROPHAGE}]_2$ on channel Tlr. After the transition, a memory is added to the residuals of both interacting processes. On one side, $\langle 2, \mathsf{Tlr}, \mathsf{INFECT}\rangle$ records that a synchronisation with process number 2 occurred, and that the sub-process INFECT was non-deterministically discharged. On the other side, the memory $\langle 1, \mathsf{Tlr}, \mathsf{nil}\rangle$ records that this is the result of a synchronisation with process number 1 and no relevant process was discharged (written nil). The data provided by these memories allow the system to backtrack to the initial process SYS. This behaviour mimics the fact that the bond between a macrophage and a virus is weak and can be broken, so reversing the reaction. On the contrary, the bond between a T cell and the corresponding antigen is strong and cannot be broken. So, the activation of T Cells (Fig. 11) can be modelled by using irreversible interactions that in CCS-R, are those involving underlined partners.

Reversible CCS [24] is a refinement of CCS-R which allows processes to backtrack from reactions only if this is in agreement with a true-concurrency notion of causal equivalence.

System specification

SYS::=VIRUS' | MACROPHAGE' | TCELL1 | TCELL2

MACROPHAGE'::= DIGEST VIRUS'::=$\overline{\text{Ant1}}$.INACT

TCELL1::=$\underline{\text{Ant1}}$.ACTIVITIES TCELL2::=$\underline{\text{Ant2}}$.ACTIVITIES

System evolution

SYS → INACT | DIGEST | ACTIVITIES | Ant2.ACTIVITIES

Fig. 11 TCELL activation in CCS-R

3.6 PEPA

PEPA [39] is a formal language for describing Markov processes. It was introduced as a tool for performance analysis of large computer and communication systems, to examine both quantitative properties (e.g. throughput, utilisation, and response time) and qualitative properties (e.g. deadlock freeness). PEPA explicitly supports stochastic information. With the advent of the systems biology era, the abstraction facilities of PEPA have been exploited in biochemical signalling pathways analysis and simulation [12].

The calculus adopts a multi-way synchronisation mechanism. The parallel composition operator is ‘ $\underset{\mathcal{L}}{\bowtie}$ ’, with $\mathcal{L}$ being the cooperation set. A PEPA specification taken from [12] follows:

$$\begin{aligned} \mathit{Prot1}_H &::= (r1, k1).\mathit{Prot1}_L \\ \mathit{Prot2}_H &::= (r1, k1).\mathit{Prot2}_L \\ \mathit{Prot3}_L &::= (r1, \top).\mathit{Prot3}_H \\ \mathit{Sys} &::= \mathit{Prot1}_H \underset{\{r1\}}{\bowtie} \mathit{Prot2}_H \underset{\{r1\}}{\bowtie} \mathit{Prot3}_L \end{aligned}$$

The system is composed of two proteins which can interact on r_1 with stochastic rate k_1 (written $(r1, k1)$), and of a third protein that can execute r_1 with indefinite rate (written $(r1, \top)$). The subscripts H and L stand for high and low protein level. The multi-way synchronisation mechanism allows the three processes to advance in a single step. So, *Sys* is transformed into $\mathit{Prot1}_L \underset{\{r1\}}{\bowtie} \mathit{Prot2}_L \underset{\{r1\}}{\bowtie} \mathit{Prot3}_H$, a system with a high level of *Prot3*, and with low levels of *Prot1* and *Prot2*.

PEPA cannot represent compartments neither directly nor indirectly by means of name passing and scope extrusion. So, the information flow from the virus to the macrophage in our example cannot be rendered in a completely faithful way; like in CCS-R, however, it can be rendered as a chain of biochemical interactions. Figure 12 shows the PEPA code. First, MACROP synchronises with VIRUS on activity Tlr with rate k. Then activity Ant1 is visible and ready to activate the proper T cells.

The specification of TCell activation is reported in Fig. 13. The virus (rather than the macrophage) synchronises with the TCell that possesses the right antigen. Notice that *Ant2* is in the cooperation set $\{\mathit{Ant1}, \mathit{Ant2}\}$. Otherwise, due to the PEPA semantics, TCELL2 could proceed without recognising the right antigen.

System specification

SYS::=VIRUS $\underset{\{Tlr\}}{\bowtie}$ MACROP

VIRUS::= (Tlr,k) . (Ant1,k_{A1}) . INACT

MACROP::= (Tlr,k) . DIGEST

System evolution

SYS $\longrightarrow$ (Ant1,k_{A1}) . INACT $\underset{\{Tlr\}}{\bowtie}$ DIGEST

Fig. 12 Phagocytosis-digestion-presentation in PEPA

System specification

SYS::= VIRUS' $\underset{\{Ant1,Ant2\}}{\bowtie}$ MACROP' $\underset{\{Ant1,Ant2\}}{\bowtie}$ TCELL1 $\underset{\{Ant1,Ant2\}}{\bowtie}$ TCELL2

MACROP'::= DIGEST

VIRUS'::=(Ant1,k_{A1}) . INACT

TCELL1::=(Ant1, k_{A1}) . ACTIVITIES1

TCELL2::=(Ant2, k_{A2}) . ACTIVITIES2

System evolution

SYS $\longrightarrow$ INACT $\underset{\{Ant1,Ant2\}}{\bowtie}$ MACROP' $\underset{\{Ant1,Ant2\}}{\bowtie}$ ACTIVITIES1 $\underset{\{Ant1,Ant2\}}{\bowtie}$ TCELL2

Fig. 13 TCELL activation in PEPA

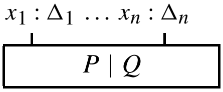
Bio-PEPA [18] is an extension of PEPA specifically designed for the representation of biochemical networks. Its main features are that it enables modellers to explicitly define the stoichiometry of reactions and to use general kinetic laws different from the basic mass-action.

Several tools for simulation and analysis support PEPA models. The PEPA Workbench [34] is the first analysis tool developed for PEPA: it enables several kinds of analysis (e.g. freedom from deadlocks and Markovian analysis of both transient and steady-state properties). The PEPA Eclipse Plug-in [61], a contribution to the Eclipse integrated development environment, includes a PEPA editor and performance analysers which use Markov chain, ODE methods, or simulation. PEPA is also supported by the multi-paradigm modelling tool Möbius [56]. ODEs can be automatically obtained from PEPA descriptions as described in [10], and models suitable for analysis in the PRISM model checker [48] can be automatically derived from PEPA models, making it possible to perform model checking of temporal properties expressed in the Continuous Stochastic Logic (CSL).

A few biological systems have been modelled and analysed using PEPA. A relevant example is the ERK signalling pathway [11].

3.7 Beta-binders

Beta-binders [67] is a bio-inspired process calculus that interprets biological entities as boxes enclosing internal processing engines. Each box is provided with typed interfaces for interaction with other boxes. The internal unit is given by parallel processes interacting by a π-calculus name-passing paradigm. Graphically, a box looks like the following:



In each pair $x_i : \Delta_i$, x_i is the name of the interface, and Δ_i is its associated type. Communication between boxes departs from plain name-passing, and is inspired by enzyme theory [45]. In particular, it is based on a user-definable notion of *compatibility* over types (see [66] for an example). Interaction between two boxes can occur only if the corresponding internal units are ready to perform complementary (input/output) actions over interface names with compatible types.

Beta-binders offers only a limited support to the dynamic modification of boxes. A few primitives drive possible changes of interaction sites: *hide* and *unhide* actions make invisible (resp. visible) a visible (resp. invisible) site, so accounting for phosphorylation and dephosphorylation, and the *expose* action allows for the addition of a new typed interface to the box. The *join* and *split* primitives allow two boxes to join together, and a box to split in two, respectively.

Figure 14 reports the Beta-binders fragment that encodes the antigen presentation phase. The global system SYS is given by the parallel composition of four boxes representing a macrophage, a virus, and two T cells, respectively. The macrophage phagocytizes the virus, resulting in a box whose interaction capabilities are inherited from the macrophage, and whose internal unit is the parallel composition of the

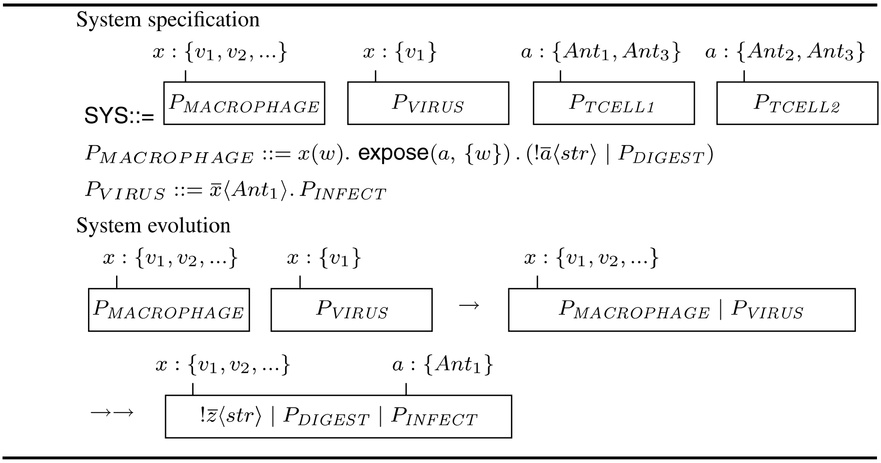

System specification
$x : \{v_1, v_2, \ldots\}$
$x : \{v_1\}$
$a : \{Ant_1, Ant_3\}$
$a : \{Ant_2, Ant_3\}$
SYS::=
$P_{MACROPHAGE}$
P_{VIRUS}
P_{TCELL1}
P_{TCELL2}
$P_{MACROPHAGE} ::= x(w).\ \mathsf{expose}(a, \{w\}).(!\bar{a}\langle str\rangle \mid P_{DIGEST})$
$P_{VIRUS} ::= \bar{x}\langle Ant_1\rangle . P_{INFECT}$
System evolution
$x : \{v_1, v_2, \ldots\}$
$x : \{v_1\}$
$P_{MACROPHAGE}$
P_{VIRUS}
$\rightarrow$
$x : \{v_1, v_2, \ldots\}$
$P_{MACROPHAGE} \mid P_{VIRUS}$
$\rightarrow\rightarrow$
$x : \{v_1, v_2, \ldots\}$
$a : \{Ant_1\}$
$!\bar{z}\langle str\rangle \mid P_{DIGEST} \mid P_{INFECT}$

Fig. 14 Phagocytosis-digestion-presentation in Beta-binders

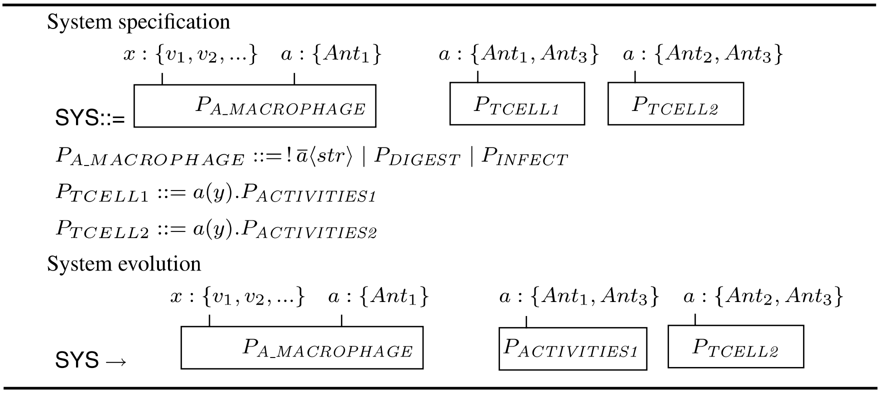


Fig. 15 TCELL activation in Beta-binders

internal bodies of the joined boxes. This represents the fact that the genetic material of the virus is absorbed by the macrophage. Hence, the virus antigen Ant_1 is passed to the macrophage and then a new site typed by $\{Ant_1\}$ is exposed.

Figure 15 shows the implementation of the T cell activation phase. Assuming that $\{Ant_1\}$ is compatible with Δ only if $Ant_1 \in \Delta$, a communication between the two leftmost boxes activates the expected T cell. Note that in Fig. 15 the boxes containing P_{TCELL1} and P_{TCELL2} have non-disjoint types $\{Ant_1, Ant_3\}$ and $\{Ant_2, Ant_3\}$, respectively. This allows us to model a typical scenario of immune systems: the two T cells have a similar shape (they can be both activated by Ant_3), and yet they are not identical (e.g. only one of them can be activated by Ant_1).

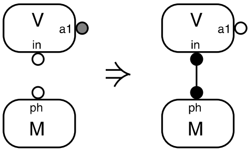
A stochastic extension of Beta-binders is presented in [28], where the notion of compatibility is used to derive quantitative parameters. In another extension [35] of the formalism, an explicit notion of compartments has been introduced, making the representation of static hierarchies of boxes easier.

The Beta Workbench [6, 72] is a collection of tools for modelling, simulating, and analysing Beta-binders systems. The simulator is based on a variant of Gillespie's stochastic simulation algorithm [33]. In [52], a parallel simulator for Beta-binders is presented.

The authors of [29] propose a formal method for measuring the effect of drugs on systems modelled in Beta-binders, using the NO-cGMP pathway as a case study. Techniques for analysing spatial [35] and temporal [36] properties of Beta-binders systems have been described and applied to the cAMP signalling pathway in OSNs, and to the ERK pathway, respectively.

3.8 κ-Calculus

The κ-calculus [26] is a formal calculus of protein interactions. It was conceived to represent complexation and decomplexation of proteins, using the concept of



(a) Graphical representation

System specification

$\mathbf{r}_1$: $\mathsf{M(ph),V(in,\overline{a1})} \rightarrow \mathsf{(x)(M(ph^{x}),V(in^{x},a1))}$

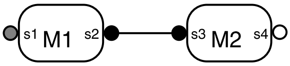
System evolution

$\mathsf{M(ph),V(in,\overline{a1})} \rightarrow \mathbf{r}_1$ $\mathsf{(x)(M(ph^{x}),V(in^{x},a1))}$

(b) Textual representation

Fig. 16 Phagocytosis-digestion-presentation in the κ-calculus

shared names to represent bonds. The units of κ-calculus are proteins, and operators are meant to represent creation and division of protein complexes. The κ-calculus comes equipped with a very clear visual notation: proteins are drawn as boxes with sites on their boundaries, which can be either visible, hidden, or bound. For instance,



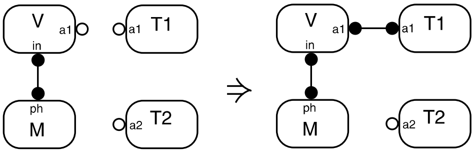
represents two molecules M1 and M2 bound at sites s2 and s3, respectively. Moreover, the site s1 of M1 is hidden and the site s4 is visible.

Besides the graphical representation, the κ-calculus provides a language in the style of process calculi. Expressions and boxes are given a semantics by a set of basic reactions. Once the initial system has been specified and the basic reductions have been fixed, the behaviour of the system is obtained by rewriting it after the reduction rules. This kind of reduction resembles pathway activation.

The calculus does not offer a natural support for managing the evolution of compartments. Like in CCS-R and PEPA, it is possible to represent the Phagocytosis-Digestion-Presentation example as an activation pathway (see Fig. 16(a)). The virus is rendered by the box V, which has a visible site in, used to enter a cell, and a hidden site a1, which represents the antigen. The macrophage is represented by the box M, which has a visible site ph, used to phagocytize a molecule. Figure 16(b) shows the single reaction relevant to our running example. In this reaction rule, the superscript x in ph^x and in^x means that the sites in and ph are linked by the channel named x.

Figure 17(a) shows both the graphical and the textual representation of the activation of a lymphocyte T helper. After phagocytosis, the virus has a visible site a1, which represents its antigen: only the lymphocyte with the right site can bind it. The graphical notation does not clearly represent the selection of the right lymphocyte. This gap is filled by the formal model via the definition of the basic reactions. In particular, the system described in Fig. 16(b) may be extended with the rules defined in Fig. 17(b). By these rules, it is possible to infer the reduction shown in Fig. 17(b).

The bioκ-calculus [49] is an extension of the κ-calculus with the addition of operations on membranes in the style of Brane calculi. Therefore, the bioκ-calculus



(a) Graphical representation

System specification

$\mathbf{r}_2$: $\mathsf{M(ph^{\mathit{x}}),V(in^{\mathit{x}},a1),T1(a1)} \rightarrow \mathsf{M(ph^{\mathit{x}}),V(in^{\mathit{x}},a1^{\mathit{y}}),T1(a1^{\mathit{y}})}$

$\mathbf{r}_3$: $\mathsf{M(ph^{\mathit{x}}),V(in^{\mathit{x}},a2),T2(a2)} \rightarrow \mathsf{M(ph^{\mathit{x}}),V(in^{\mathit{x}},a2^{\mathit{y}}),T2(a2^{\mathit{y}})}$

System evolution

$\mathsf{(x)(M(ph^{\mathit{x}}),V(in^{\mathit{x}},a1),T1(a1))} \quad \rightarrow \mathbf{r}_2$
$\mathsf{(xy)(M(ph^{\mathit{x}}),V(in^{\mathit{x}},a1^{\mathit{y}}),T1(a1^{\mathit{y}}))}$

(b) Textual representation

Fig. 17 TCELL activation in κ-calculus

can easily represent both protein interactions (complexation and decomplexation) and cell interactions (membrane fusions and protein translocations).

A stochastic κ-calculus simulator is described in [21]. The Kappa Factory [42] is a graphical platform for the design, analysis, and simulation of bio-molecular systems. Different kinds of analysis methods are supported, e.g. static dependencies on rules and analysis of traces. In [22] and [23], techniques for reachability and causality analysis are described.

The EGFR/ERK pathway has been extensively modelled and analysed using the κ-calculus [20, 22, 23].

4 Concluding Remarks

A survey of the process calculi that have been adopted to describe the behaviour of living entities has been presented. The common feature of all the surveyed languages is the interpretation of biological interactions in terms of synchronisation/communication of concurrent processes. As for the rest, the calculi differ in many respects. Some of them take compartments as primitive, while others adopt name-passing to simulate complex formation. Explicit compartments, when present, can be either flat or dynamically organised in a hierarchy.

None of the presented languages seems to emerge as the perfect general-purpose formalism for modelling biology. Rather, some of the calculi have been proved to be well suited for modelling interactions at the level of cellular systems, while others for rendering signalling pathways. Sometimes the main features of two calculi are merged together to get a more expressive formalism; an example is the bioκ-calculus, which combines Brane calculi with the κ-calculus.

Table 2 Summary on tools and case studies

	Simulation	Analysis	Case studies
Biochemical π-calculus	BioSPI [7], SPiM [76]	causality [19]	[15, 17, 47, 50, 51]
BioAmbients	BioSpi [7]	CFA [57],	
		spatial [58]	[4, 64]
Brane Calculi	–	causality [8],	
		modal logic [53]	[9]
CCS-R	–	causality [24]	–
PEPA	PEPA Workbench [34],	Markovian [61],	
	Eclipse Plug-in [61]	model checking [13]	[11]
Beta-binders	Beta Workbench [6]	reachability [29],	
		locality [35]	[29, 35]
κ-calculus	Kappa Factory [42]	reachability [22]	
		causality [23]	[20]

Table 2 schematically summarises some of the considered issues for each of the calculi we have dealt with: the availability of simulation platforms; the adaptation or development of analysis techniques; the investigation of complex case studies.

The largest number of case studies has been modelled using the biochemical π-calculus. BioAmbients first, and Brane calculi later on, has introduced an explicit notion of compartments. Relevant analysis tools and interesting case studies have been developed for both of them. Beta-binders and κ-calculus focus on the representation of biochemical interactions. The first one adopts a communication paradigm depending on the compatibility of the interacting sites, while the latter provides an accurate description of complexation and decomplexation. No simulation environment has been developed for CCS-R, but still this is the only language offering primitive support for the description of reversibility, a key biological mechanism. Finally, PEPA is surely the language exhibiting the largest number of well-established simulation and analysis tools. Moreover, its multi-way synchronisation makes the calculus close to chemical equations and hence more appealing to biologists.

Deepening the understanding of the process calculi approach to the description of living entities poses new challenges for computer scientists. First of all, process calculi may seem obscure to non-experts, and user-friendly software platforms need to be developed. Another crucial point is the relation between in-silico activity (modelling, simulation, analysis) and wetlab experiments. Models are meaningless if not derived from real biological observations, and in-silico experiments and analysis have to be validated to prove their predictive flavour. However, techniques that allow the direct quantitative measure of pathway activities or protein levels, considering a single cell, are not common practise and, so, laboratory protocols have to be adapted or renewed. Process calculi for biology are in their pioneering phase and, although they are powerful and promising, a closer collaboration between life and computer scientists is required to bring appreciable results.

References

1. Akutsu T, Miyano S, Kuhara S (2000) Algorithms for identifying Boolean networks and related biological networks based on matrix multiplication and fingerprint function. J Comput Biol 7(3):331–343
2. Alves R, Savageau MA (2000) Extending the method of mathematically controlled comparison to include numerical comparisons. Bioinformatics 16(9):786–798
3. Bader GD, Donaldson I, Wolting C, Ouellette BF, Pawson T, Hogue CW (2001) BIND—the biomolecular interaction network database. Nucleic Acids Res 29(1):242–245
4. van Bakel S, Kahn I, Vigliotti M, Heath J (2007) Modelling intercellular fate of FGF receptors with BioAmbients. In: ENTCS
5. Baldan P, Bracciali A, Brodo L, Bruni R (2007) Deducing interactions in partially unspecified biological systems. In: Proceedings of the second international conference on algebraic biology (AB 2007). Lecture notes in computer science, vol 4545. Springer, Berlin, pp 262–276
6. The Beta workbench home page. http://www.cosbi.eu/Rpty_Soft_BetaWB.php
7. BioSpi home page. http://www.wisdom.weizmann.ac.il/~biospi
8. Busi N (2007) Towards a causal semantics for brane calculi. In: Proceedings of the fifth brainstorming week on membrane computing, pp 97–111
9. Busi N, Zandron C (2006) Modeling and analysis of biological processes by mem(brane) calculi and systems. In: Proceedings of the winter simulation conference (WSC 2006), Monterey, CA, USA, December 3–6, 2006. WSC, Monterey, pp 1646–1655
10. Calder M, Gilmore S, Hillston J (2005) Automatically deriving ODEs from process algebra models of signalling pathways. In: Plotkin G (ed) Proceedings of computational methods in systems biology (CMSB 2005), Edinburgh, Scotland, pp 204–215
11. Calder M, Gilmore S, Hillston J (2006) Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA. Trans Comput Syst Biol VII(4230):1–23; also appeared in: Proc BioCONCUR'04
12. Calder M, Gilmore S, Hillston J, Vyshemirsky V (2009) Formal methods for biochemical signalling pathways. In: Formal methods: state of art and new directions, BCS FACS. Springer, Berlin (in press)
13. Calder M, Vyshemirsky V, Orton R, Gilbert D (2005) Analysis of signalling pathways using the PRISM model checker. In: Plotkin G (ed) Third international workshop on computational methods in systems biology (CMSB'05)
14. Cardelli L (2005) Brane calculi—interactions of biological membranes. In: Computational methods in systems biology, international conference CMSB 2004, revised selected papers, Paris, France, May 26–28, 2004. Lecture notes in computer science, vol 3082. Springer, Berlin, pp 257–278
15. Cardelli L, Gardner P, Kahramanoğulları O (2008) A process model of rho GTP-binding proteins in the context of phagocytosis. In: Proc. of FBTC'07. Electron Notes Theor Comput Sci 194(3):87–102
16. Chang R (2005) Physical chemistry for the biosciences. University Science
17. Chiarugi D, Degano P, Marangoni R (2007) A computational approach to the functional screening of genomes. PLoS Comput Biol 3(9):1801–1806
18. Ciocchetta F, Hillston J (2007) Bio-PEPA: an extension of the process algebra PEPA for biochemical networks. In: From biology to concurrency and back (FBTC 07), ENTCS
19. Curti M, Degano P, Priami C, Baldari C (2004) Modelling biochemical pathways through enhanced π-calculus. Theor Comput Sci 325(1):111–140
20. Danos V, Feret J, Fontana W, Harmer R, Krivine J (2007) Rule-based modelling of cellular signalling. In: Proceedings CONCUR'07. Lecture notes in computer science. Springer, Berlin
21. Danos V, Feret J, Fontana W, Krivine J (2007) Scalable simulation of cellular signaling networks. In: Proceedings APLAS'07
22. Danos V, Feret J, Fontana W, Krivine J (2008) Abstract interpretation of reachable complexes in biological signalling networks. In: Proceedings VMCAI'08. Lecture notes in computer science. Springer, Berlin

23. Danos V, Fontana W, Harmer R, Krivine J (2007) Biological signalling and causality
24. Danos V, Krivine J (2004) Reversible communicating systems. In: Proceedings CONCUR'04. Lecture notes in computer science, vol 3170. Springer, Berlin, pp 292–307
25. Danos V, Krivine J (2007) Formal molecular biology done in CCS-R. Electron Notes Theor Comput Sci 180(3):31–49
26. Danos V, Laneve C (2004) Formal molecular biology. Theor Comput Sci 325(1)
27. Danos V, Pradalier S (2005) Projective brane calculus. In: Computational methods in systems biology, international conference CMSB 2004, revised selected papers, Paris, France, May 26–28, 2004. Lecture notes in computer science, vol 3082. Springer, Berlin, pp 134–148
28. Degano P, Prandi D, Priami C, Quaglia P (2006) Beta-binders for biological quantitative experiments. Electron Notes Theor Comput Sci 164(3):101–117
29. Dematte L, Prandi D, Priami C, Romanel A (2007) Effective index: a formal measure of drug effects. In: Proceedings FOSBE 2007, pp 485–490
30. Dhaeseleer P, Liang S, Somogyi R (2000) Genetic network inference: from co-expression clustering to reverse engineering. Bioinformatics 16(8):707–726
31. Efroni S, Harel D, Cohen IR (2003) Towards rigorous comprehension of biological complexity: modeling, execution, and visualization of thymic T-cell maturation Genome Res 13(11):2485–2497
32. Fontana W, Buss L (1996) The barrier of objects: from dynamical systems to bounded organizations. In: Boundaries and barriers: on the limits to scientific knowledge. Addison–Wesley, Reading
33. Gillespie D (1977) Exact stochastic simulation of coupled chemical reactions. J Phys Chem 81(25):2340–2361
34. Gilmore S, Hillston J (1994) The PEPA workbench: a tool to support a process algebra-based approach to performance modelling. In: Proceedings of the seventh international conference on modelling techniques and tools for computer performance evaluation. Lecture notes in computer science, vol 794. Springer, Vienna, pp 353–368
35. Guerriero M, Priami C, Romanel A (2007) Beta-binders with static compartments. In: Proceedings of the second international conference on algebraic biology (AB07). Lecture notes in computer science. Springer, Berlin
36. Guerriero ML, Priami C (2006) Causality and concurrency in beta-binders. Cosbi Technical Report TR-01-2006. Available at http://www.cosbi.eu/templates/cosbi/php/get_paper.php?id=1
37. Heath J, Kwiatkowska M, Norman G, Parker D, Tymchyshyn O (2008) Probabilistic model checking of complex biological pathways. Theor Comput Sci 319:239–257
38. van Helden J, Naim A, Mancuso R, Eldridge M, Wernisch L, D DG, Wodak S (2000) Representing and analysing molecular and cellular function using the computer. Biol Chem 381(9–10):921–935
39. Hillston J (1996) A compositional approach to performance modelling. Cambridge University Press, Cambridge
40. Hoare CAR (1985) Communicating sequential processes. Prentice–Hall, New York
41. Kam N, Harel D, Kugler H, Marelly R, Pnueli A, Hubbard EJA, Stern MJ (2003) Formal modeling of C. elegans development: a scenario-based approach. In: Computational methods in systems biology, proceedings of the first international workshop, CMSB 2003, Roverto, Italy, February 24–26, 2003. Lecture notes in computer science, vol 2602. Springer, Berlin
42. The kappa factory. http://www.lix.polytechnique.fr/~krivine/kappaFactory.html
43. Karp PD, Riley M, Saier M, Paulsen I, Collado-Vides J, Paley S, Pellegrini-Toole A, Bonavides C, Gama-Castro S (2002) The EcoCyc database. Nucleic Acids Res 30(1):56–58
44. Kazic T (2000) Semiotes: a semantics for sharing. Bioinformatics 16(12):1129–1144
45. Koshland D (1958) Application of a theory of enzyme specificity to protein synthesis. Proc Natl Acad Sci USA 44(2):98–104
46. Kuttler C, Lhoussaine C, Niehren C (2007) A stochastic pi calculus for concurrent objects. In: Algebraic biology, second international conference, AB 2007, Castle of Hagenberg, Austria, July 2–4, 2007. Lecture notes in computer science, vol 4545. Springer, Berlin, pp 232–246

47. Kuttler C, Niehren J (2006) Gene regulation in the pi calculus: simulating cooperativity at the lambda switch. Trans Comput Syst Biol VII(4230):24–55
48. Kwiatkowska M, Norman G, Parker D (2002) PRISM: probabilistic symbolic model checker. In: Proceedings TOOLS 2002. Lecture notes in computer science, vol 2324. Springer, Berlin, pp 200–204
49. Laneve C, Tarissan F (2007) A simple calculus for proteins and cells. Electron Notes Theor Comput Sci 171:139–154
50. Lecca P, Priami C (2007) Cell cycle control in eukaryotes: a biospi model. Electron Notes Theor Comput Sci 180(3):51–63
51. Lecca P, Priami C, Quaglia P, Rossi B, Laudanna C, Costantin G (2004) A stochastic process algebra approach to simulation of autoreactive lymphocyte recruitment. SIMULATION: Trans Soc Mod Simul Int 80(6):273–288
52. Leye S, Priami C, Uhrmacher A (2007) A parallel beta-binders simulator. Tech Rep TR-17-2007, The Microsoft Research, University of Trento CoSBi
53. Miculan M, Bacci G (2006) Modal logics for brane calculus. In: Proceedings of the computational methods in systems biology, international conference (CMSB 2006). Lecture notes in computer science, vol 4210. Springer, Berlin, pp 1–16
54. Milner R (1989) Communication and concurrency. Prentice–Hall, New York
55. Milner R (1999) Communicating and mobile systems: the π-calculus. Cambridge University Press, Cambridge
56. Möbius home page. http://www.mobius.uiuc.edu/
57. Nielson F, Nielson H, Priami C, Rosa D (2007) Control flow analysis for BioAmbients. Electron Notes Theor Comput Sci 180(3):65–79
58. Nielson HR, Nielson F, Pilegaard H (2004) Spatial analysis of bioambients. In: Giacobazzi R (ed) Proceedings of the static analysis, 11th international symposium (SAS'04). Lecture notes in computer science, vol 3148. Springer, Berlin, pp 69–83
59. Păun G (2002) Membrane computing. An introduction. Springer, Berlin
60. Peleg M, Yeh I, Altman R (2002) Modeling biological processes using workflow and Petri net models. Bioinformatics 18:825–837
61. The PEPA plug-in project. http://www.dcs.ed.ac.uk/pepa/tools/plugin/
62. Phillips A, Cardelli L (2004) A correct abstract machine for the stochastic pi-calculus. In: BioConcur '04, workshop on concurrent models in molecular biology
63. Phillips A, Cardelli L, Castagna G (2006) A graphical representation for biological processes in the stochastic pi-calculus. Trans Comput Syst Biol 4230:123–152
64. Pilegaard H, Nielson F, Nielson HR (2005) Static analysis of a model of the LDL degradation pathway. In: Plotkin G (ed) Third international workshop on computational methods in systems biology (CMSB'05)
65. Plotkin GD (2004) A structural approach to operational semantics. J Log Algebr Program 60–61:17–139
66. Prandi D, Priami C, Quaglia P (2006) Shape spaces in formal interactions. ComPlexUS 2(3–4):128–139
67. Priami C, Quaglia P (2005) Beta binders for biological interactions. In: Danos V, Schächter V (eds) Proceedings of the 2nd international workshop on computational methods in systems biology (CMSB '04). Lecture notes in bioinformatics, vol 3082. Springer, Berlin, pp 21–34
68. Priami C, Regev A, Silverman W, Shapiro E (2001) Application of a stochastic name-passing calculus to representation and simulation of molecular processes. Inf Process Lett 80(1):25–31
69. Regev A, Panina EM, Silverman W, Cardelli L, Shapiro EY (2004) BioAmbients: an abstraction for biological compartments. Theor Comput Sci 325(1):141–167
70. Regev A, Shapiro E (2002) Cells as computation. Nature 419:343
71. Regev A, Silverman W, Shapiro E (2001) Representation and simulation of biochemical processes using the π-calculus process algebra. In: Proceedings of pacific symposium on biocomputing (PSB'01), vol 6, pp 459–470
72. Romanel A, Dematté L, Priami C (2007) The beta workbench. Tech rep TR-03-2007, The Microsoft Research, University of Trento CoSBi

73. Sangiorgi D, Walker D (2001) The π-calculus: a theory of mobile processes. Cambridge University Press, Cambridge
74. Sauro HM, Hucka M, Finney A, Wellock C, Bolouri H, Doyle J, Kitano H (2003) Next generation simulation tools: the systems biology workbench and biospice integration. OMICS: J Integr Biol 7(4):355–372
75. Schuster S, Fell DA, Dandekar T (2000) A general definition of metabolic pathways useful for systematic organization and analysis of complex metabolic networks. Nat Biotechnol 18(3):326–332
76. SPiM home page. http://research.microsoft.com/en-us/projects/spim/
77. Stenesh J (1998) Biochemistry. Springer, Berlin
78. Szallasi Z (1999) Genetic network analysis in light of massively parallel biological data. In: Altman R, Dunker A, Hunter L, Klein T (eds) Pacific symposium on biocomputing, vol 4. World Scientific, Singapore, pp 5–16
79. Versari C, Busi N (2007) Efficient stochastic simulation of biological systems with multiple variable volumes. In: Proceedings FBTC 07
80. Wingender E, Chen X, Fricke E, Geffers R, Hehl R, Liebich I, M MK, Matys V, Michael H, Ohnhauser R, Pruss M, Schacherer F, Thiele S, Urbach S (2001) The TRANSFAC system on gene expression regulation. Nucleic Acids Res 29(1):281–283
81. Yi TM, Huang Y, Simon MI, Doyle J (2000) Robust perfect adaptation in bacterial chemotaxis through integral feedback control. Proc Natl Acad Sci USA 97(9):4649–4653

Deriving Differential Equations from Process Algebra Models in Reagent-Centric Style

Jane Hillston and Adam Duguid

Abstract The *reagent-centric* style of modeling allows stochastic process algebra models of biochemical signaling pathways to be developed in an intuitive way. Furthermore, once constructed, the models are amenable to analysis by a number of different mathematical approaches including both stochastic simulation and coupled ordinary differential equations. In this chapter, we give a tutorial introduction to the reagent-centric style, in PEPA and Bio-PEPA, and the way in which such models can be used to generate systems of ordinary differential equations.

1 Introduction

Process algebras are a well-established modeling approach for representing concurrent systems to facilitate both qualitative and quantitative analysis [6, 20, 27]. Within the last decade, they have also been proposed as the basis for several modeling techniques applied to biological problems, particularly intracellular signaling pathways, e.g. [2, 8, 9, 16, 21, 22, 30, 32, 33]. The original motivation for this proposal was the observation of the clear mapping that can be made between *molecules*, within a biochemical pathway, and *processes*, within concurrent systems [34]. The mapping is then elaborated with reactions between molecules represented by communication between processes, etc.

This mapping has been extremely influential with much subsequent work following its lead. It takes an inherently *individuals*-based view of a pathway or cell. This is unsurprising because process algebras have traditionally been used in this manner with each entity within the system captured explicitly. However, there are drawbacks to such an individuals-based view, principally the problem of *state-space explosion*. When each individual within a system is represented explicitly and all transitions within or between individuals are captured as discrete events, the number of states becomes prohibitively high. This problem prohibits the use of techniques which rely on access to the state space in its entirety, such as model checking or numerical solution of a Markov chain. However, it also impacts on other techniques such as stochastic simulation because although they explore the state space iteratively, the size of the state space still has a deleterious impact on the likely computational cost of obtaining statistically significant results.

J. Hillston (✉)
School of Informatics, The University of Edinburgh, Edinburgh, Scotland, UK
e-mail: Jane.Hillston@ed.ac.uk

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_24, © Springer-Verlag Berlin Heidelberg 2009

Biologists have recognized and empirically solved this problem for decades with the use of systems of ordinary differential equations (ODEs) to represent intracellular systems. Although rarely presented in that way, these mathematical models are continuous or *fluid* approximations of the true model of the system. The reason that the approximation aspect of the ODE models is often ignored is because in many circumstances it is an extremely good approximation, certainly adequate for the analyses and reasoning which need to be performed. The idea underlying the approximation is that when the number of molecules become high, the discrete changes that take place as counts are increased or decreased by one or two molecules (according to stoichiometry), may be regarded as continuous changes. In the biological setting, where the exact number of molecules is often difficult to obtain but is known to be extremely large, this more abstract view is both intellectually and computationally appealing. The continuous state space models, in the form of systems of ODEs, are much more efficiently solved than their discrete state space counterparts.

We have been investigating the role of process algebra modeling in this *population*-based modeling approach. This requires a somewhat different abstraction than the *molecules-as-processes* one proposed by Regev et al. in order to cleanly derive the fluid approximation. We term this style of modeling *reagent-centric* modeling and we exemplify it below with the stochastic process algebra PEPA and the recently defined biologically-inspired modification Bio-PEPA.

2 Reagent-Centric Modeling

A process algebra model captures the behavior of a system as the actions and interactions between a number of entities, usually termed *agents* or *components*. In classical process algebras such as CCS [27] and the π-calculus [28], the actions are atomic and interactions are strictly binary, consisting of complementary input and output actions. In CSP [26], the form of interaction is generalized to be synchronization on shared names, allowing multi-way communications to take place. In stochastic process algebras, such as PEPA [25] or the stochastic π-calculus [31], a random variable representing average duration is associated with each action. In the stochastic π-calculus, as in the π-calculus, interactions are strictly binary whereas in PEPA the more general, multi-way synchronization is supported.

We use the term *reagent* to mean an entity which engages in a reaction. In the basic case, this will be a biochemical species such as a protein, receptor molecule or an enzyme. However, it may be more abstract, capturing the behavior of a group of entities or a whole subsidiary pathway. Although the participants in a reaction will be particular species, in some circumstances, the modeler may choose not to represent them explicitly. There may be a variety of reasons for this choice. For example, detailed knowledge of the pathway may be unknown or may be only tangentially relevant to the pathway which is under study. Equally as we add a new species/entity to the model we want to specify that it interacts with the existing components, the existing reagent, building the behavior of the model incrementally.

This addresses one of the shortcomings of the molecules-as-processes abstraction. In a process algebra compositionality is a key feature. Thus, a model, consisting of a number of interacting agents, is itself an agent. However, the same cannot be said for molecules. A number of molecules who interact do not, in general, result in a molecule. However, with our abstract notion of a reagent, it is the case that interacting regents do result in a reagent.

The reagent-centric style of modeling also makes a subtle but essential shift from the molecules-as-processes abstraction for modeling biochemical pathways in process algebra in terms of how models are constructed. In the molecules-as-processes abstraction each molecule in a cell is modeled as an agent and the actions of the agent record the changes which occur for that molecule. Consider a reaction *react* in which a molecule of species A combines with a molecule of species B to form a compound C. In the molecules-as-processes style, this would be represented as:[1]

$$A \stackrel{\text{def}}{=} \mathit{react}.C, \qquad B \stackrel{\text{def}}{=} \overline{\mathit{react}}.\mathbf{0}, \qquad A \mid B.$$

In contrast in the reagent-style each *species* is represented as a distinct agent:

$$\begin{aligned} A(l) &\stackrel{\text{def}}{=} \mathit{react}.A(l-1),\\ B(l) &\stackrel{\text{def}}{=} \mathit{react}.B(l-1),\\ C(l) &\stackrel{\text{def}}{=} \mathit{react}.C(l+1). \end{aligned}$$

Here, the formal parameter l used within a model records the capacity of the component to participate in the reaction. When the model is to be interpreted as a set of ODEs, then this denotes the concentration. The key thing is that model description records the impact of the reaction on this capacity. The system is then the composition of all these species, interacting on the action *react*:

$$\big(A(l_{A_0}) \underset{\text{react}}{\bowtie} B(l_{B_0})\big) \underset{\text{react}}{\bowtie} C(l_{C_0}).$$

Note that all species are represented in this *model equation* even though some may have initial concentration zero (say $l_{C_0} = 0$)

Thus, in the reagent-centric style of modeling the atomic components of the model are generally the species of the biological system. Each component records the reactions which the species may be involved in together with the role that the species has in that reaction, i.e. whether the species is a reactant or a product or an enzyme. In PEPA models, this was done informally, but in the recently developed process algebra Bio-PEPA, the syntax is designed to record such details.

To some extent, the different styles here result from the forms of interaction which are supported in the process algebras. In the π-calculus, where complementary actions are used, it would not be possible to model in the reagent-centric style,

[1] We disregard rates for the initial discussion to be able to focus on the style of representation.

whereas this is completely natural in a language such as PEPA which supports multiway synchronization.

In our discussions so far, we have ignored the specification of the rate at which a reaction takes place, and not used any specific process algebra. In the subsequent discussions, we will use the stochastic process algebra PEPA which has the following syntax:

$$S ::= (\alpha, r).S \mid S + S \mid C_S,$$
$$P ::= P \underset{L}{\bowtie} P \mid P/L \mid C$$

where S denotes a *sequential component* and P denotes a *model component* which executes in parallel. C stands for a constant which denotes either a sequential component or a model component as introduced by a definition. C_S stands for constants which denote sequential components. The effect of this syntactic separation between these types of constants is to constrain legal PEPA components to be cooperations of sequential processes. Thus, sequential components may have a designated first action, or offer alternative actions. Constants are used to name the behavior of components, and mutually recursive definitions allow the possibility of infinite behavior. Model components are *cooperations* of model components, controlled by a cooperation set L which specifies which actions must be synchronized between the components. Actions outside the set L may be undertaken independently and concurrently. For more details, see [25].

Let us consider a simple Michaelis–Menten reaction.

$$S + E \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} C \xrightarrow{k_2} E + P.$$

Here, S is a substrate, P is a product, and E is an enzyme. Substrate and enzyme may bind to form a complex, C. This reaction has a rate constant k_1. The complex may dissociate into its original components with rate k_{-1} or into the enzyme and product with rate k_2.

Modeling this in PEPA in the reagent-centric style, assuming just two abstract levels (a justification for this will be discussed in the next section) for each species, we distinguish only *high*, denoted by a subscript H, and *low*, denoted by a subscript L:

$$\begin{aligned}
S_H &\overset{\text{def}}{=} (bind, k_1).S_L,\\
S_L &\overset{\text{def}}{=} (dissoc, \top).S_H,\\
E_H &\overset{\text{def}}{=} (bind, \top).E_L,\\
E_L &\overset{\text{def}}{=} (dissoc, \top).E_H + (prod, \top).E_H,\\
C_H &\overset{\text{def}}{=} (dissoc, k_{-1}).C_L + (prod, k_2).C_L,\\
C_L &\overset{\text{def}}{=} (bind, \top).C_H,
\end{aligned}$$

$$P_L \overset{\text{def}}{=} (prod, \top).P_H.$$

The model equation is

$$\big((S_H \underset{L_1}{\bowtie} E_H) \underset{L_2}{\bowtie} C_L\big) \underset{L_2}{\bowtie} P_L$$

where $L_1 = \{bind, dissoc\}$ and $L_2 = \{bind, dissoc, prod\}$.

In the following, we term *components* of the model the sequential components corresponding to the species in the signaling pathways (S, E, C, and P in the example above) and *derivatives*, the internal states of the components, e.g. S_H and S_L.

3 Deriving ODEs

Even at the coarsest level of abstraction, distinguishing only high and low levels, the reagent-centric process algebra model provides sufficient information for deriving an ODE representation of the same system [7]. This is because it is sufficient to know which reactions increase concentration (low-to-high) and which decrease it (high-to-low), and this is recorded quite explicitly in the reagent-centric style. Thus, for any reagent-centric PEPA model, with derivatives designated high and low, it is straightforward to construct an *activity graph* which records this information.

Definition 1 (Activity Graph) An *activity graph* is a bipartite graph (N, A). The nodes N are partitioned into N_r, the *reactions*, and N_a, the *reagents*. $A \subseteq (N_r \times N_a) \cup (N_a \times N_r)$, where $a = (n_r, n_a) \in A$ if n_r is a reaction in which the concentration of reagent n_a is increased, and $a = (n_a, n_r) \in A$ if n_r is a reaction in which the concentration of reagent n_a is decreased.

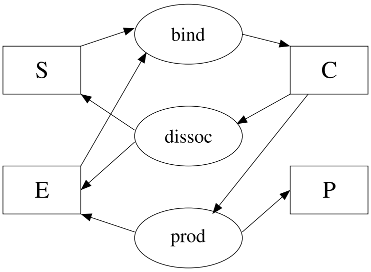
The same information can be represented in a matrix, termed the *activity matrix*.

Definition 2 (Activity Matrix) For a pathway which has R reactions and S reagents, the *activity matrix* M_a is an $S \times R$ matrix, and the entries are defined as follows:

$$(s_i, r_j) = \begin{cases} +1 & \text{if } (r_j, s_i) \in A, \\ -1 & \text{if } (s_i, r_j) \in A, \\ 0 & \text{if } (s_i, r_j) \notin A \cup (r_j, s_i) \notin A. \end{cases}$$

The activity graph and activity matrix corresponding to the simple Michaelis–Menten reaction modeled in the previous section are presented in Fig. 1.

In the activity matrix, each row corresponds to a single reagent and is essentially the stoichiometry matrix. In the representation of the pathway as a system of ODEs, there is one equation for each reagent, detailing the impact of the rest of the system on the concentration of that reagent. This can derived automatically from the activity matrix, when we associate a concentration variable m_i with each row of the matrix. The entries in the row indicate which reactions have an impact on this reagent, the



$$\begin{array}{c} \\ S \\ E \\ C \\ P \end{array} \begin{pmatrix} \text{bind} & \text{dissoc} & \text{prod} \\ -1 & +1 & 0 \\ -1 & +1 & +1 \\ +1 & -1 & -1 \\ 0 & 0 & +1 \end{pmatrix}$$

Fig. 1 Activity graph and activity matrix corresponding to the Michaelis–Menten reaction

```
**for** $i \leftarrow 1 \ldots N_A$ **do** {One differential equation per reagent}
  $\frac{dN_i}{dt} \leftarrow 0$
  **for** $j \leftarrow 1 \ldots N_R$ **do**
    **if** $M_a(i, j) \neq 0$ **then** {Activity $j$ influences reagent $i$}
      $S \leftarrow \emptyset$ {Create the set of reactants for activity $j$}
      **for** $k \leftarrow 1 \ldots N_A$ **do**
        **if** $M_a(k, j) = -1$ **then**
          $S \leftarrow S \cup \{k\}$
      **if** $M_a(i, j) = +1$ **then** {Product from activity $j$}
        $\frac{dN_i}{dt} \leftarrow \frac{dN_i}{dt} + r_j \times \prod_{k \in S} n_k(t)$
      **else** {Reactant from activity $j$}
        $\frac{dN_i}{dt} \leftarrow \frac{dN_i}{dt} - r_j \times \prod_{k \in S} n_k(t)$
```

Fig. 2 Pseudo-code for the algorithm deriving a system of ODEs from an activity matrix

sign of the entry showing whether the effect is to increase or decrease concentration. Thus, the number of terms in the ODE will be equal to the number of non-zero entries in the corresponding row, each term being based on the rate constant for the reaction associated with that row. Assuming that reactions are governed by mass action, the actual rate of change caused by each reaction will be the rate constant multiplied by the current concentration of those reagents consumed in the reaction. The identity of these reagents can be found in the column corresponding to the reaction, a negative entry indicating that a reagent is consumed. The pseudo-code for this algorithm is presented in Fig. 2.

4 Example: Tumor Necrosis Factor α

In this section, we demonstrate the use of reagent-centric modeling, the derivation of an ODE model and the results which can be obtained, for a more biologically interesting system.

Tumor Necrosis Factor α (TNFα) is a potent proinflammatory *cytokine* (a protein used for cell signaling). TNFα plays an important role in a wide spectrum of biological processes including cellular proliferation, differentiation, inflammation, lipid metabolism, coagulation, and programmed cell death (*apoptosis*). It has been implicated in a variety of diseases, including autoimmune diseases, insulin resistance, arthritis, and cancer. A thorough mathematical treatment of the dynamics of the TNFα pathway may help to throw light on these important concerns in human health.

The role of TNFα in regulating these processes depends on complicated signaling cascades and networks involving binding of TNFα to receptors including TNFR1 and recruitment of the signal transducers TRADD, RIP, and TRAF2 which activate effectors such as IKK and NF-κB. The TNF pathway is shown in Fig. 3.

In the figure, the circles depict the reagents and the rectangles depict the reactions. Note that reversible reactions are shown as two uni-directional reactions, which is also how they are represented in the PEPA model. The shaded circles indicate those reagents (TNFα, TNFR1, TRADD, RIP, TRAF2, IKK, and NF-κB/IκB) which are known to have a non-zero initial concentration. All other initial concentrations are assumed to be zero.

A system-theoretic approach to the analysis and quantitative modeling of the TNFα-mediated NF-κB-signaling pathway appears in [17]. The authors develop an ODE model and use it to validate their model of the pathway against experimental results. Here, we develop a PEPA model in reagent-centric style for the TNFα-mediated NF-κB-signaling pathway. The definitions of the components used appear in Fig. 4 and the corresponding model equation is presented in Fig. 5. In the PEPA model, the initial state of each reagent is recorded in the model equation, where those reagents which have a non-zero initial concentration start in the "high" state. In this representation, such terms are followed by a parameter in square brackets indicating the initial value. Note also that in the model equation the cooperation sets are not explicitly represented. Instead, a wildcard notation "*" is used to indicate that synchronization takes place over all shared names, i.e. whenever two components have an activity in common they must cooperate on that activity.

From this PEPA model, we are able to generate a system of coupled ODEs as explained above. These are presented in Fig. 6. Our system of ODEs agrees with the system which was developed directly in [17]. This provides convincing evidence that our PEPA model is describing the same system. Through numerical integration of our ODEs, we can see the processes of binding, recruitment, and activation at work in the graphs in Fig. 7. Reading the graphs from left to right, top to bottom, we see the cascade of reactions corresponding to the reactions in the top center, and then right-hand side of Fig. 3. These time series results agree with those presented in [17].

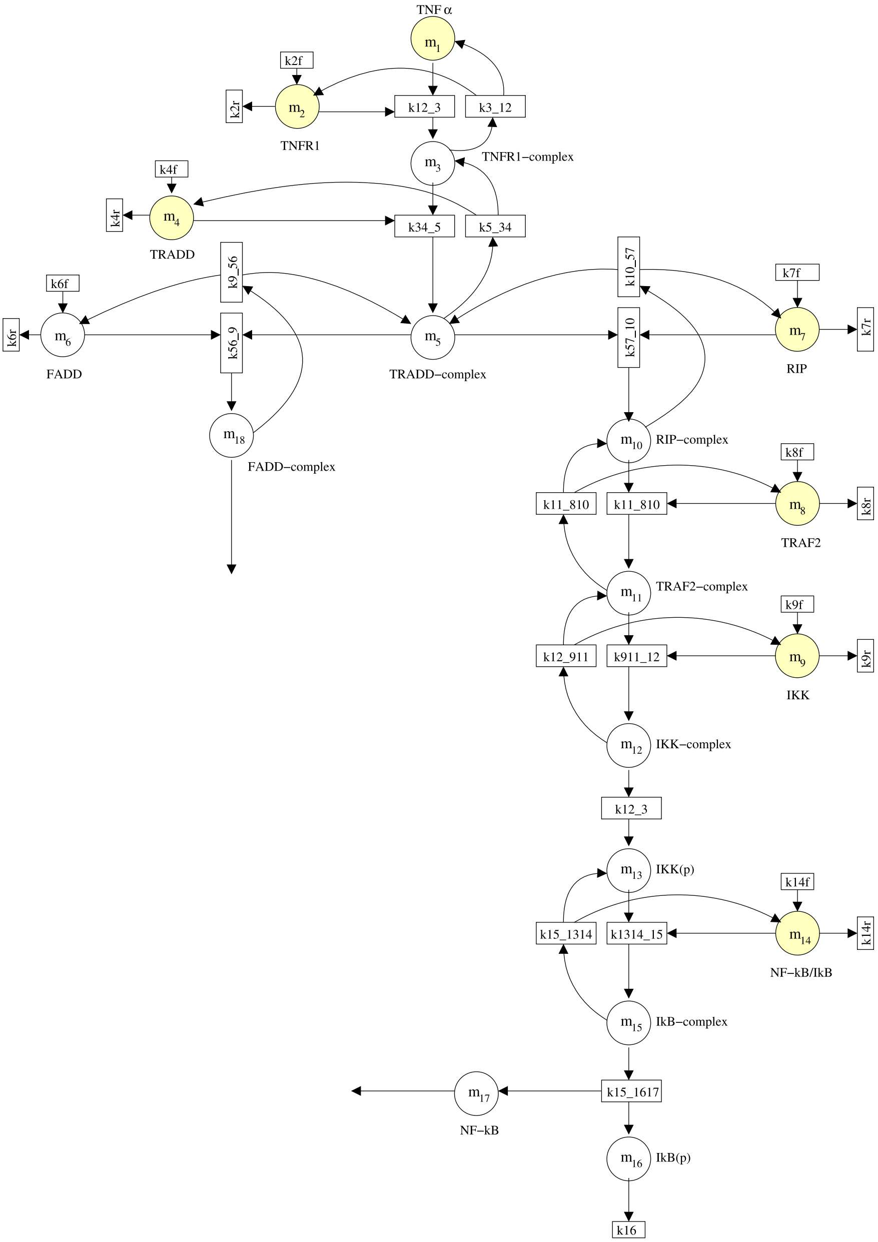


Fig. 3 Schematic diagram showing the reactions of the TNF signaling pathway

The graphs presented in Fig. 8 show the same results from a different perspective, making the timing behavior of the cascade explicit.

$$
\begin{aligned}
\mathrm{TNF}\alpha_H &\overset{\mathrm{def}}{=} (r_{12_3}, k_{12_3}).\mathrm{TNF}\alpha_L,\\
\mathrm{TNF}\alpha_L &\overset{\mathrm{def}}{=} (r_{3_12}, k_{3_12}).\mathrm{TNF}\alpha_H,\\
\mathrm{TNFR1}_H &\overset{\mathrm{def}}{=} (r_{12_3}, k_{12_3}).\mathrm{TNFR1}_L + (r_{2r}, k_{2r}).\mathrm{TNFR1}_L,\\
\mathrm{TNFR1}_L &\overset{\mathrm{def}}{=} (r_{3_12}, k_{3_12}).\mathrm{TNFR1}_H + (r_{2f}, k_{2f}).\mathrm{TNFR1}_H,\\
\mathrm{TNFR1}^C_H &\overset{\mathrm{def}}{=} (r_{3_12}, k_{3_12}).\mathrm{TNFR1}^C_L + (r_{34_5}, k_{34_5}).\mathrm{TNFR1}^C_L,\\
\mathrm{TNFR1}^C_L &\overset{\mathrm{def}}{=} (r_{12_3}, k_{12_3}).\mathrm{TNFR1}^C_H + (r_{5_34}, k_{5_34}).\mathrm{TNFR1}^C_H,\\
\mathrm{TRADD}_H &\overset{\mathrm{def}}{=} (r_{34_5}, k_{34_5}).\mathrm{TRADD}_L + (r_{4r}, k_{4r}).\mathrm{TRADD}_L,\\
\mathrm{TRADD}_L &\overset{\mathrm{def}}{=} (r_{5_34}, k_{5_34}).\mathrm{TRADD}_H + (r_{4f}, k_{4f}).\mathrm{TRADD}_H,\\
\mathrm{TRADD}^C_H &\overset{\mathrm{def}}{=} (r_{5_34}, k_{5_34}).\mathrm{TRADD}^C_L + (r_{57_10}, k_{57_10}).\mathrm{TRADD}^C_L\\
&\quad + (r_{56_9}, k_{56_9}).\mathrm{TRADD}^C_L,\\
\mathrm{TRADD}^C_L &\overset{\mathrm{def}}{=} (r_{34_5}, k_{34_5}).\mathrm{TRADD}^C_H + (r_{10_57}, k_{10_57}).\mathrm{TRADD}^C_H\\
&\quad + (r_{9_56}, k_{9_56}).\mathrm{TRADD}^C_H,\\
\mathrm{FADD}_H &\overset{\mathrm{def}}{=} (r_{6r}, k_{6r}).\mathrm{FADD}_L + (r_{56_9}, k_{56_9}).\mathrm{FADD}_L,\\
\mathrm{FADD}_L &\overset{\mathrm{def}}{=} (r_{6f}, k_{6f}).\mathrm{FADD}_H + (r_{9_56}, k_{9_56}).\mathrm{FADD}_H,\\
\mathrm{RIP}_H &\overset{\mathrm{def}}{=} (r_{57_10}, k_{57_10}).\mathrm{RIP}_L + (r_{7r}, k_{7r}).\mathrm{RIP}_L,\\
\mathrm{RIP}_L &\overset{\mathrm{def}}{=} (r_{10_57}, k_{10_57}).\mathrm{RIP}_H + (r_{7f}, k_{7f}).\mathrm{RIP}_H,\\
\mathrm{TRAF2}_H &\overset{\mathrm{def}}{=} (r_{810_11}, k_{810_11}).\mathrm{TRAF2}_L + (r_{8r}, k_{8r}).\mathrm{TRAF2}_L,\\
\mathrm{TRAF2}_L &\overset{\mathrm{def}}{=} (r_{11_810}, k_{11_810}).\mathrm{TRAF2}_H + (r_{8f}, k_{8f}).\mathrm{TRAF2}_H,\\
\mathrm{IKK}_H &\overset{\mathrm{def}}{=} (r_{911_12}, k_{911_12}).\mathrm{IKK}_L + (r_{9r}, k_{9r}).\mathrm{IKK}_L,\\
\mathrm{IKK}_L &\overset{\mathrm{def}}{=} (r_{12_911}, k_{12_911}).\mathrm{IKK}_H + (r_{9f}, k_{9f}).\mathrm{IKK}_H,\\
\mathrm{RIP}^C_H &\overset{\mathrm{def}}{=} (r_{10_57}, k_{10_57}).\mathrm{RIP}^C_L + (r_{810_11}, k_{810_11}).\mathrm{RIP}^C_L,\\
\mathrm{RIP}^C_L &\overset{\mathrm{def}}{=} (r_{57_10}, k_{57_10}).\mathrm{RIP}^C_H + (r_{11_810}, k_{11_810}).\mathrm{RIP}^C_H,\\
\mathrm{TRAF2}^C_H &\overset{\mathrm{def}}{=} (r_{11_810}, k_{11_810}).\mathrm{TRAF2}^C_L + (r_{911_12}, k_{911_12}).\mathrm{TRAF2}^C_L,\\
\mathrm{TRAF2}^C_L &\overset{\mathrm{def}}{=} (r_{810_11}, k_{810_11}).\mathrm{TRAF2}^C_H + (r_{12_911}, k_{12_911}).\mathrm{TRAF2}^C_H,\\
\mathrm{IKK}^C_H &\overset{\mathrm{def}}{=} (r_{12_911}, k_{12_911}).\mathrm{IKK}^C_L + (r_{12_13}, k_{12_13}).\mathrm{IKK}^C_L,\\
\mathrm{IKK}^C_L &\overset{\mathrm{def}}{=} (r_{911_12}, k_{911_12}).\mathrm{IKK}^C_H,\\
\mathrm{IKK}^P_H &\overset{\mathrm{def}}{=} (r_{1314_15}, k_{1314_15}).\mathrm{IKK}^P_L,\\
\mathrm{IKK}^P_L &\overset{\mathrm{def}}{=} (r_{12_13}, k_{12_13}).\mathrm{IKK}^P_H + (r_{15_1314}, k_{15_1314}).\mathrm{IKK}^P_H,\\
\mathrm{NF}\text{-}\kappa\mathrm{B}/\mathrm{I}\kappa\mathrm{B}_H &\overset{\mathrm{def}}{=} (r_{1314_15}, k_{1314_15}).\mathrm{NF}\text{-}\kappa\mathrm{B}/\mathrm{I}\kappa\mathrm{B}_L + (r_{14r}, k_{14r}).\mathrm{NF}\text{-}\kappa\mathrm{B}/\mathrm{I}\kappa\mathrm{B}_L,\\
\mathrm{NF}\text{-}\kappa\mathrm{B}/\mathrm{I}\kappa\mathrm{B}_L &\overset{\mathrm{def}}{=} (r_{15_1314}, k_{15_1314}).\mathrm{NF}\text{-}\kappa\mathrm{B}/\mathrm{I}\kappa\mathrm{B}_H + (r_{14f}, k_{14f}).\mathrm{NF}\text{-}\kappa\mathrm{B}/\mathrm{I}\kappa\mathrm{B}_H,\\
\mathrm{I}\kappa\mathrm{B}^C_H &\overset{\mathrm{def}}{=} (r_{15_1314}, k_{15_1314}).\mathrm{I}\kappa\mathrm{B}^C_L + (r_{15_1617}, k_{15_1617}).\mathrm{I}\kappa\mathrm{B}^C_L,\\
\mathrm{I}\kappa\mathrm{B}^C_L &\overset{\mathrm{def}}{=} (r_{1314_15}, k_{1314_15}).\mathrm{I}\kappa\mathrm{B}^C_H,\\
\mathrm{I}\kappa\mathrm{B}_H &\overset{\mathrm{def}}{=} (r_{16}, k_{16}).\mathrm{I}\kappa\mathrm{B}_L,\\
\mathrm{I}\kappa\mathrm{B}_L &\overset{\mathrm{def}}{=} (r_{15_1617}, k_{15_1617}).\mathrm{I}\kappa\mathrm{B}_H,\\
\mathrm{NF}\text{-}\kappa\mathrm{B}_L &\overset{\mathrm{def}}{=} (r_{15_1617}, k_{15_1617}).\mathrm{NF}\text{-}\kappa\mathrm{B}_H,\\
\mathrm{NF}\text{-}\kappa\mathrm{B}_H &\overset{\mathrm{def}}{=} (\mathit{proliferation}, p).\mathrm{NF}\text{-}\kappa\mathrm{B}_H,\\
\mathrm{I}\kappa\mathrm{B}^P_H &\overset{\mathrm{def}}{=} (r_{9_56}, k_{9_56}).\mathrm{I}\kappa\mathrm{B}^P_L,\\
\mathrm{I}\kappa\mathrm{B}^P_L &\overset{\mathrm{def}}{=} (r_{56_9}, k_{56_9}).\mathrm{I}\kappa\mathrm{B}^P_H.
\end{aligned}
$$

Fig. 4 Component definitions for the PEPA model of the TNF pathway in the reagent-centric style

$$\begin{aligned}&\mathrm{TNF}\alpha_H[20] \underset{\{*\}}{\bowtie} \mathrm{TNFR1}_H[25] \underset{\{*\}}{\bowtie} \mathrm{TNFR1}_L^C \underset{\{*\}}{\bowtie} \mathrm{TRADD}_H[25] \underset{\{*\}}{\bowtie}\\ &\quad \mathrm{TRADD}_L^C \underset{\{*\}}{\bowtie} \mathrm{FADD}_L \underset{\{*\}}{\bowtie} \mathrm{RIP}_H[25] \underset{\{*\}}{\bowtie} \mathrm{RIP}_L^C \underset{\{*\}}{\bowtie}\\ &\qquad \mathrm{TRAF2}_H[25] \underset{\{*\}}{\bowtie} \mathrm{TRAF2}_L^C \underset{\{*\}}{\bowtie} \mathrm{IKK}_H[25] \underset{\{*\}}{\bowtie} \mathrm{IKK}_L^C \underset{\{*\}}{\bowtie}\\ &\qquad\quad \mathrm{IKK}_L^P \underset{\{*\}}{\bowtie} \mathrm{NF}\text{-}\kappa\mathrm{B}/\mathrm{I}\kappa\mathrm{B}_H[25] \underset{\{*\}}{\bowtie} \mathrm{I}\kappa\mathrm{B}_L^C \underset{\{*\}}{\bowtie} \mathrm{I}\kappa\mathrm{B}_L \underset{\{*\}}{\bowtie}\\ &\qquad\qquad \mathrm{NF}\text{-}\kappa\mathrm{B}_L \underset{\{*\}}{\bowtie} \mathrm{I}\kappa\mathrm{B}_L^P\end{aligned}$$

Fig. 5 Model equation for the PEPA model of the TNF pathway in reagent-centric style

5 More Powerful Reagent-Centric Modeling: Bio-PEPA

We viewed our work on reagent-centric modeling in PEPA as an experiment to investigate the usefulness of the approach and process algebra in the systems biology context. Whilst the reagent-centric style appears to offer distinct advantages, there were limitations in what could be expressed in PEPA. In particular, the main difficulties related to the definition of *stoichiometric coefficients* (i.e. the coefficients used to show the quantitative relationships of the reactants and products in a biochemical reaction) and the representation of *kinetic laws*.

The difficulty in representing stoichiometry is not limited to PEPA: other process algebras, where molecules are represented as processes and interactions are limited to two processes (as in the π-calculus), also have problems capturing reactions with stoichiometry greater than one.

In terms of kinetic laws, PEPA and other process algebras consider elementary reactions with constant rates (*mass-action* kinetic laws). The problem of extending to the domain of kinetic laws beyond basic mass-action (*general kinetic laws*) is particularly relevant, as these kinds of reactions are frequently found in the literature as abstractions of complex situations whose details are unknown. Reducing all reactions to the elementary steps is complex and often impractical. In the case of process algebras, such as the π-calculus, the assumption of elementary reactions is motivated by the fact that they rely on Gillespie's stochastic simulation for analysis. Some recent works have extended the approach of Gillespie to deal with complex reactions [1, 10], but these extensions are yet to be reflected in the work using process algebras.

In this section, we give a brief overview of Bio-PEPA, a new language for the modeling and analysis of biochemical networks. Bio-PEPA is based on the reagent-centric view in PEPA, modified in order to represent explicitly some features of biochemical models, such as stoichiometry and the role of the different species in a given reaction. A major feature of Bio-PEPA is the introduction of functional rates to express general kinetic laws. Each action type represents a reaction in the model and is associated with a functional rate.

Bio-PEPA is based on a high level of abstraction similar to the one proposed in formalisms such as *SBML* [4], which have been widely adopted by biologists. Following the reagent-centric view, models are based not on individual molecules, but on discrete levels of concentration within a species: each component represents

$$
\begin{aligned}
\frac{\mathrm{d}m_1(t)}{\mathrm{d}t} &= k_{3\_12} \cdot m_3(t) - k_{12\_3} \cdot m_1(t) \cdot m_2(t),\\
\frac{\mathrm{d}m_2(t)}{\mathrm{d}t} &= k_{2f} + k_{3\_12} \cdot m_3(t) - \big(k_{2r} + k_{12\_3} \cdot m_1(t)\big) \cdot m_2(t),\\
\frac{\mathrm{d}m_3(t)}{\mathrm{d}t} &= k_{12\_3} \cdot m_1(t) \cdot m_2(t) + k_{5\_34} \cdot m_5(t) - \big(k_{3\_12} + k_{34\_5} \cdot m_4(t)\big) \cdot m_3(t),\\
\frac{\mathrm{d}m_4(t)}{\mathrm{d}t} &= k_{4f} + k_{5\_34} \cdot m_5(t) - \big(k_{4r} + k_{34\_5} \cdot m_3(t)\big) \cdot m_4(t),\\
\frac{\mathrm{d}m_5(t)}{\mathrm{d}t} &= k_{34\_5} \cdot m_3(t) \cdot m_4(t) + k_{18\_56} \cdot m_{18}(t) + k_{10\_57} \cdot m_{10}(t)\\
&\quad - (k_{5\_34} + k_{56\_18} \cdot m_6(t) + k_{57\_10} \cdot m_7(t)) \cdot m_5(t),\\
\frac{\mathrm{d}m_6(t)}{\mathrm{d}t} &= k_{6f} + k_{18\_56} \cdot m_{18}(t) - \big(k_{6r} + k_{56\_18} \cdot m_5(t)\big) \cdot m_6(t),\\
\frac{\mathrm{d}m_7(t)}{\mathrm{d}t} &= k_{7f} + k_{10\_57} \cdot m_{10}(t) - \big(k_{7r} + k_{57\_10} \cdot m_5(t)\big) \cdot m_7(t),\\
\frac{\mathrm{d}m_8(t)}{\mathrm{d}t} &= k_{8f} + k_{11\_810} \cdot m_{11}(t) - \big(k_{8r} + k_{810\_11} \cdot m_{10}(t)\big) \cdot m_8(t),\\
\frac{\mathrm{d}m_9(t)}{\mathrm{d}t} &= k_{9f} + k_{12\_911} \cdot m_{12}(t) - \big(k_{9r} + k_{911\_12} \cdot m_{11}(t)\big) \cdot m_9(t),\\
\frac{\mathrm{d}m_{10}(t)}{\mathrm{d}t} &= k_{57\_10} \cdot m_5(t) \cdot m_7(t) + k_{11\_810} \cdot m_{11}(t)\\
&\quad - \big(k_{10\_57} + k_{810\_11} \cdot m_8(t)\big) \cdot m_{10}(t),\\
\frac{\mathrm{d}m_{11}(t)}{\mathrm{d}t} &= k_{810\_11} \cdot m_8(t) \cdot m_{10}(t) + k_{12\_911} \cdot m_{12}(t)\\
&\quad - \big(k_{11\_810} + k_{911\_12} \cdot m_9(t)\big) \cdot m_{11}(t),\\
\frac{\mathrm{d}m_{12}(t)}{\mathrm{d}t} &= k_{911\_12} \cdot m_9(t) \cdot m_{11}(t) - (k_{12\_911} + k_{12\_13}) \cdot m_{12}(t),\\
\frac{\mathrm{d}m_{13}(t)}{\mathrm{d}t} &= k_{12\_13} \cdot m_{12}(t) + k_{15\_1314} \cdot m_{15}(t) - k_{1314\_15} \cdot m_{13}(t) \cdot m_{14}(t),\\
\frac{\mathrm{d}m_{14}(t)}{\mathrm{d}t} &= k_{14f} + k_{15\_1314} \cdot m_{15}(t) - \big(k_{14r} + k_{1314\_15} \cdot m_{13}(t)\big) \cdot m_{14}(t),\\
\frac{\mathrm{d}m_{15}(t)}{\mathrm{d}t} &= k_{1314\_15} \cdot m_{13}(t) \cdot m_{14}(t) - (k_{15\_1314} + k_{15\_1617}) \cdot m_{15}(t),\\
\frac{\mathrm{d}m_{16}(t)}{\mathrm{d}t} &= k_{15\_1617} \cdot m_{15}(t) - k_{16} \cdot m_{16}(t),\\
\frac{\mathrm{d}m_{17}(t)}{\mathrm{d}t} &= k_{15\_1617} \cdot m_{15}(t),\\
\frac{\mathrm{d}m_{18}(t)}{\mathrm{d}t} &= k_{56\_18} \cdot m_5(t) \cdot m_6(t) - k_{18\_56} \cdot m_{18}(t).
\end{aligned}
$$

Fig. 6 Differential equations generated from the PEPA model of the TNF pathway in reagent-centric style

a species and it is parametric in terms of concentration level. Some advantages of this view are:

- It allows us to deal with uncertainty/incomplete information in the exact number of elements (semi-quantitative data);
- In a discrete state space representation, the focus is on the concentration levels rather than the number of elements. This means that the state space is reduced as there are fewer states for each component.

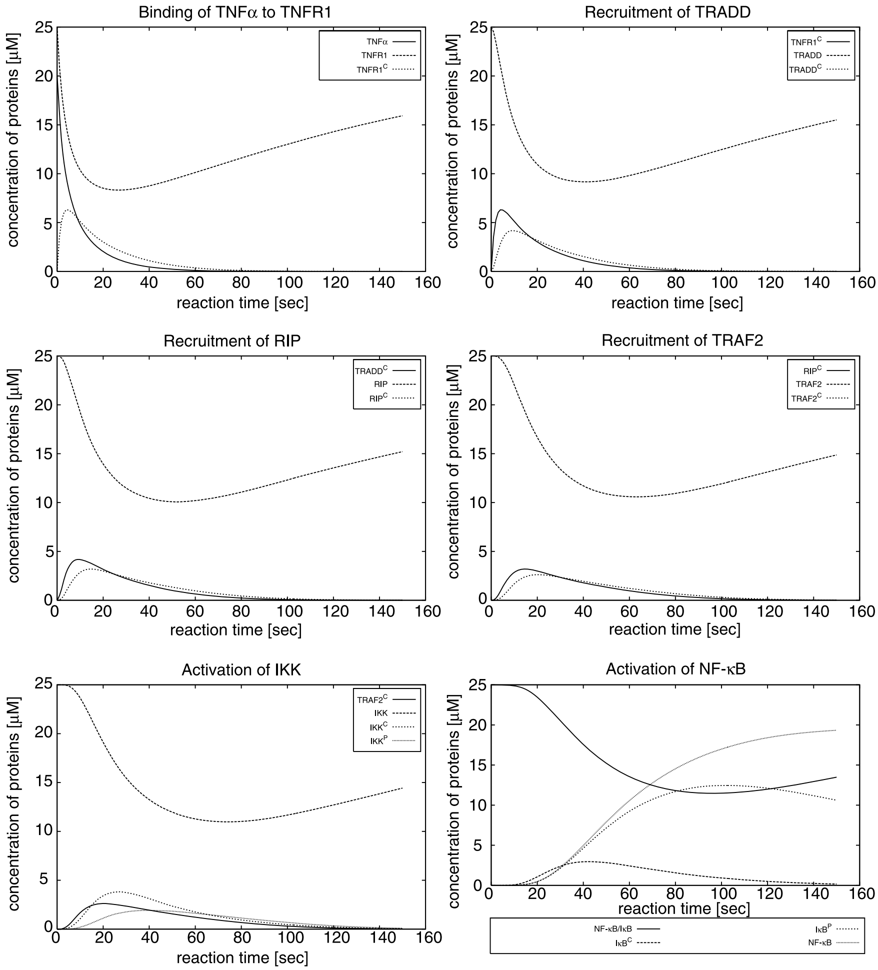


Fig. 7 Time-series plots obtained by numerically integrating the ODEs derived from the PEPA model. These plots show the processes of recruitment and activation at work

- The population level view, in terms of continuously changing concentrations, and the individual level view, counting molecules, are both easily recovered from this abstract view.

In Bio-PEPA the *granularity* of the system is defined in terms of the step size h of the concentration intervals; the same step size h is used for all the species. This is motivated by the fact that, following the *law of conservation of mass*, there must be a "balance" between the concentrations consumed (reactants) and the ones created (products). When the stoichiometry is greater than one, we need to consider concentration quantities proportional to stoichiometric coefficients. Given a species i, we can assume that it has a maximum finite concentration M_i. The number of levels

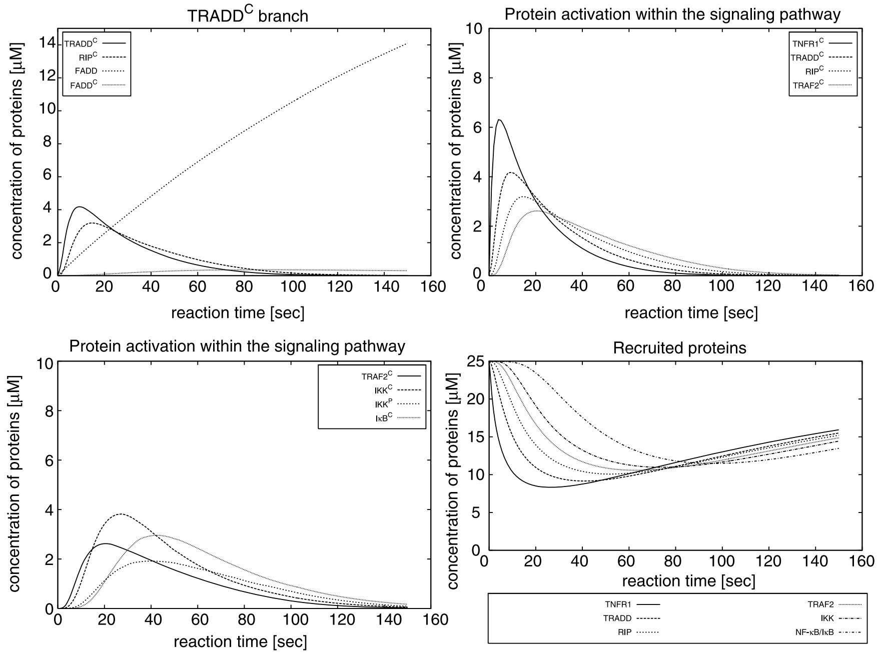


Fig. 8 Additional time-series plots obtained by numerically integrating the ODEs derived from the PEPA model. These plots show the cascade of protein activation at work in the signaling pathway and the temporal dependency in the recruitment of proteins

for the species i is given by $N_i + 1$ where $N_i = \lceil \frac{M_i}{h} \rceil$ (the integer value greater than or equal to $\frac{M_i}{h}$). Each species can assume the discrete concentration levels from 0 (concentration null) to N_i (maximum concentration).

If l_i is the concentration level for the species i, the concentration is taken to be $x_i = l_i \times h$.

The syntax of Bio-PEPA is similar to that of PEPA but with some important differences. As in PEPA, a model is made up of a number of sequential components; here there is one sequential component for each species. The syntax is designed in order to collect the biological information that we need. For example, instead of a single prefix combinator there are a number of different operators which capture the role that the species plays with respect to this reaction.

$$S ::= (\alpha, \kappa) \text{ op } S \mid S + S \mid C, \qquad P ::= P \underset{\mathcal{L}}{\bowtie} P \mid S(l)$$

where $\text{op} = \downarrow \mid \uparrow \mid \oplus \mid \ominus \mid \odot$.

The component S is called a *sequential component* (or *species component*) and represents the species. The component P, called a *model component*, describes the system and the interactions among components. We suppose a countable set of sequential components $\mathcal{C}$ and a countable set of action types $\mathcal{A}$. The parameter $l \in \mathbb{N}$ represents the discrete level of concentration. The prefix term, (α, κ) op S, contains information about the role of the species in the reaction associated with the action type α:

- (α, κ) is the *activity* or *reaction*, where $\alpha \in \mathcal{A}$ is the *action type* and κ is the *stoichiometric coefficient* of the species in that reaction; information about the rate of the reaction is defined elsewhere (in contrast to PEPA);
- the *prefix combinator* "op" represents the role of the element in the reaction. Specifically, $\downarrow$ indicates a *reactant*, $\uparrow$ a *product*, $\oplus$ an *activator*, $\ominus$ an *inhibitor* and $\odot$ a generic *modifier*.

The choice operator, cooperation, and definition of constant are unchanged.

To fully describe a biochemical network in Bio-PEPA, we need to define structures that collect information about the compartments, the maximum concentrations, number of levels for all the species, the constant parameters, and the functional rates which specify the rates of reactions. A preliminary version of the language was proposed in [18], with full details presented in [19].

In order to specify the dynamics of the system, we associate a functional rate f_{α_j} with each action α_j. This function represents the kinetic law of the associated reaction. For the definition of functional rates, we consider mathematical expressions with simple operations and operators involving constant parameters and components. In addition, Bio-PEPA has some predefined functions to express the most commonly used kinetic laws: mass-action, Michaelis–Menten, and Hill kinetics.

The functional rates are defined externally to the components and are evaluated when the system is derived. They are used to derive the transition rates of the system. In the functional rates, some parameter constants can be used. These must be defined in the model by means of a set of parameter definitions.

6 Related Work

As mentioned earlier, several other process algebras have been considered in the context of biological systems. Initial work focused upon the π*-calculus* and its biochemical stochastic extension [32], but later work has included several process algebra developed specifically for modeling biological processes, such as Beta-binders [30], Brane Calculi [11], BioAmbients [12], and the Calculus of Looping Sequences [3]. In most cases, any quantitative analysis is carried out using stochastic simulation, based on Gillespie's algorithm [24], treating each molecule individually.

In his work on the stochastic π-calculus, Cardelli has also considered the derivation of systems of ODEs and the relationship between the ODE models and the stochastic simulation [13, 14]. However, as the stochastic π-calculus models are based on the molecules-as-processes abstraction, the mapping to ODEs is less straightforward than is the case for the reagent-centric models presented here.

We have discussed how Bio-PEPA allows models which capture general kinetic laws and stoichiometry, features which are central in approaches designed by biologists, such as SBML [4], but are not readily represented in most process algebras. An exception is the stochastic extension of *Concurrent Constraint Programming* (sCCP) [5]. In [5], Bortolussi and Policriti show how to apply this process algebra to biological systems. In their work, each species is represented by a variable and the reactions are expressed by constraints on these variables. The domain of application is extended to any kind of reactions and the rate can be expressed by a generic function. The analysis is limited to stochastic simulation using Gillespie's algorithm.

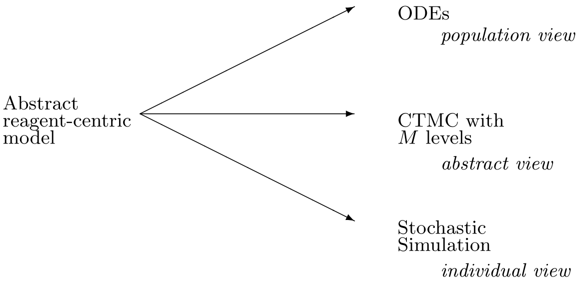
Also somewhat related is the work on BIOCHAM [15], a programming environment for modeling biochemical systems, making simulations and querying the model in temporal logic. In its current version, BIOCHAM is based on a rule-based language for modeling biochemical systems, in which species are expressed by objects and reactions by reaction rules. The rates are expressed using functions whose definitions are similar to those proposed in Bio-PEPA. This language allows the evaluation of CTL queries using the *NuSMV* model checker [29]. In addition to simulation models, ODE-based models may also be derived.

7 Conclusions

The reagent-centric style of modeling offers the modeler a great deal of flexibility in how models can be analysed. In particular, it offers an easy route to the population-based models, based on systems of ordinary differential equations, which are widely used by biologists. For many models, this style of mathematical analysis offers a fast and efficient route to obtaining quantitative insights into the behavior of the system. However, when a microscopic view of the system is more appropriate, for example, when certain macromolecules are present in extremely low numbers, and continuous approximation cannot be justified, discrete state space models are just as readily extracted from the reagent-centric description. This has the advantage that the modeler does not need to commit to a particular form of analysis at the instigation of a modeling study.

These two approaches can seem to be in opposition. Stochastic simulation uses a discrete state space, stochastic model based on the individual elements in the system, molecules. Models can be computationally expensive to solve and many trajectories are needed in order to get statistically significant results. In contrast, ODE approaches use a continuous state space, deterministic model based on the population view of the system, distinguishing populations of molecules in terms of concentrations. Many efficient solvers for ODEs are available and only one solution is needed due to the deterministic nature of the model. It is known that when populations are large, i.e. there are large number of molecules, stochastic simulation tends to the same results as the ODE models. So, these approaches should be viewed as alternatives, each having a valuable role to play in appropriate circumstances [35]. One of the motivations for introducing a high-level modeling formalism is that it gives the possibility of the same model being mapped to the alternative mathematical representations and solved in either way, rather than the modeler having to commit to just one approach at the inception of a modeling study. We have already demonstrated the benefits that this can bring for model verification. In [9], we give an account

Fig. 9 Alternative modeling approaches: a single reagent-centric description of a system may be used to derive alternative mathematical representations offering different analysis possibilities



of how an error in the time course data obtained from a hand-crafted ODE model in [23] was uncovered by comparing the ODE and stochastic simulation models readily derived from a PEPA model of the same pathway.

Furthermore, the reagent-centric style of modeling offers a third approach to modeling the system, which can be seen as a compromise between the population view of the ODEs and the individuals view of the stochastic simulation (see Fig. 9). The *abstract* view aggregates individual molecules/discretizes the population view and gives an intermediate which is discrete and stochastic, but without the prohibitively large state spaces encountered when each molecule is modeled as an individual. This gives us access to Markovian analysis techniques based on numerical linear algebra such as passage time analysis and stochastic model checking.

Acknowledgements The authors thank Stephen Gilmore for his helpful comments in the preparation of this paper. Jane Hillston is supported by the EPSRC ARF "Process Algebra Approaches for Collective Dynamics" (EP/c543696/01) and the EPSRC project "SIGNAL: Stochastic process algebra for biochemical signaling pathway analysis" (EP/E031439/1). Adam Duguid is supported by the EPSRC DTG EP/P501407/1.

References

1. Arkin AP, Rao CV (2003) Stochastic chemical kinetics and the quasi-steady-state assumption: application to the Gillespie algorithm. J Chem Phys 1:4999–5010
2. van Bakel S, Kahn I, Vigliotti M, Heath J (2007) Modeling intracellular fate of FGF receptors with bio-ambients. Electron Notes Comput Sci
3. Barbuti R, Maggiolo-Schettini A, Milazzo P, Troina A (2005) A calculus of looping sequences for modeling microbiological systems. In: Proceedings of CS&P'05, 14th international workshop on concurrency specification and programming. Warsaw University, Warsaw, pp 29–40
4. Bornstein BJ, Doyle JC, Finney A, Funahashi A, Hucka M, Keating SM, Kitano H, Kovitz BL, Matthews J, Shapiro BE, Schilstra MJ (2004) Evolving a lingua franca and associated software infrastructure for computational systems biology: the systems biology markup language (SBML) project. Syst Biol 1:41–53
5. Bortolussi L, Policriti A (2006) Modeling biological systems in stochastic concurrent constraint programming. In: Proceedings of WCB 2006
6. Bruns G (1997) Distributed systems analysis with CCS. Prentice–Hall, New York
7. Calder M, Gilmore S, Hillston J (2005) Automatically deriving ODEs from process algebra models of signaling pathways. In: Proceedings of CMSB'05, pp 204–215

8. Calder M, Gilmore S, Hillston J (2006) Modeling the influence of RKIP on the ERK signaling pathway using the stochastic process algebra PEPA. In: Transactions on computational systems biology VII. Lecture notes in computer science, vol 4230. Springer, Berlin, pp 1–23
9. Calder M, Duguid A, Gilmore S, Hillston J (2006) Stronger computational modeling of signaling pathways using both continuous and discrete-space methods. In: Proceedings of CMSB'06. Lecture notes in computer science, vol 4210. Springer, Berlin, pp 63–77
10. Cao Y, Gillespie DT, Petzold L (2005) Accelerated stochastic simulation of the stiff enzyme-substrate reaction. J Chem Phys 123(14):144917–144929
11. Cardelli L (2005) Brane calculi—interactions of biological membranes. In: Proceedings of workshop on computational methods in systems biology (CMSB'04). Lecture notes in computer science, vol 3082. Springer, Berlin, pp 257–278
12. Cardelli L, Panina EM, Regev A, Shapiro E, Silverman W (2004) BioAmbients: an abstraction for biological compartments. Theor Comput Sci 325(1):141–167
13. Cardelli L (2006) Artificial biochemistry. Technical Report TR-08-2006, The Microsoft Research, University of Trento Centre for Computational and Systems Biology
14. Cardelli L (2008) On process rate semantics. Theor Comput Sci 391(3):190–215
15. Chabrier-Rivier N, Fages F, Soliman S (2004) Modeling and querying interaction networks in the biochemical abstract machine BIOCHAM. J Biol Phys Chem 4:64–73
16. Chiarugi D, Degano P, Marangoni R (2007) A computational approach to the functional screening of genomes. PLOS Comput Biol 9:1801–1806
17. Cho K-H, Shin S-Y, Lee H-W, Wolkenhauer O (2003) Investigations into the analysis and modeling of the TNFα-mediated NF-κB-signaling pathway. Genome Res 13:2413–2422
18. Ciocchetta F, Hillston J (2008) Bio-PEPA: an extension of the process algebra PEPA for biochemical networks. Electron Notes Theor Comput Sci 194(3):103–117
19. Ciocchetta F, Hillston J (2008) Bio-PEPA: a framework for the modeling and analysis of biological systems. Technical Report of the School of Informatics, University of Edinburgh, EDI-INF-RR-1231
20. Clark A, Gilmore S, Hillston J, Tribastone M (2007) Stochastic process algebras. In: Formal methods in performance modeling, Bertinoro, June 2007. Lecture notes in computer science, vol 4486. Springer, Berlin, pp 132–179
21. Danos V, Krivine J (2003) Formal molecular biology done in CCS-R. In: Proceedings of workshop on concurrent models in molecular biology (BioConcur'03)
22. Danos V, Feret J, Fontana W, Harmer R, Krivine J (2007) Ruled-based modeling of cellular signaling. In: Proceedings of CONCUR'07. Lecture notes in computer science, vol 4703. Springer, Berlin
23. Eichler-Jonsson C, Gilles ED, Muller G, Schoeberl B (2002) Computational modeling of the dynamics of the MAP kinase cascade activated by surface and internalized EGF receptors. Nat Biotechnol 20:370–375
24. Gillespie DT (1977) Exact stochastic simulation of coupled chemical reactions. J Phys Chem 81:2340–2361
25. Hillston J (1996) A Compositional approach to performance modeling. Cambridge University Press, Cambridge
26. Hoare CAR (1985) Communicating sequential processes. Prentice–Hall, New York
27. Milner R (1989) Communication and concurrency. International series in computer science, Prentice–Hall, New York
28. Milner R (1999) Communicating and mobile systems: the π-calculus. Cambridge University Press, Cambridge
29. NuMSV model checker. http://nusmv.irst.itc.it
30. Priami C, Quaglia P (2005) Beta-binders for biological interactions. In: Proceedings of CMSB'04. Lecture notes in computer science, vol 3082. Springer, Berlin, pp 20–33
31. Priami C (1995) Stochastic π-calculus. Comput J 38(6):578–589
32. Priami C, Regev A, Silverman W, Shapiro E (2001) Application of a stochastic name-passing calculus to representation and simulation of molecular processes. Inf Process Lett 80:25–31

33. Regev A (2001) Representation and simulation of molecular pathways in the stochastic π-calculus. In: Proceedings of the 2nd workshop on computation of biochemical pathways and genetic networks
34. Regev A, Shapiro E (2002) Cells as computation. Nature 419:343
35. Wolkenhauer O, Ullah M, Kolch W, Cho KH (2004) Modeling and simulation of intracellular dynamics: choosing an appropriate framework. IEEE Trans Nanobiosci 3:200–207

Programmable DNA-Based Finite Automata

Tamar Ratner and Ehud Keinan

Abstract Computation using DNA has many advantages, including the potential for massive parallelism that allows for large number of operations per second, the direct interface between the computation process and a biological output, and the miniaturization of the computing devices to a molecular scale. In 2001, we reported on the first DNA-based, programmable finite automaton (2-symbol-2-state) capable of computing autonomously with all its hardware, software, input, and output being soluble biomolecules mixed in solution. Later, using similar principles, we developed advanced 3-symbol-3-state automata. We have also shown that real-time detection of the output signal, as well as real-time monitoring of all the computation intermediates, can be achieved by the use of surface plasmon resonance (SPR) technology. More recently, we have shown that it is possible to achieve a biologically relevant output, such as specific gene expression, by using a reporter-gene as an output-readout. We cloned the input into circular plasmids, and thereby achieved control over gene expression by a programmable sequence of computation events. Further efforts are currently directed to immobilization of the input molecules onto a solid chip to enable parallel computation, where the location of the input on the chip represents specific tagging.

1 Introduction

Bio-Molecular Computing (BMC) is rapidly evolving as an independent field of science at the interface of computer science, mathematics, chemistry, and biology [1–3]. Living organisms carry out complex physical processes dictated by molecular information. Biochemical reactions, and ultimately the entire organism's operations are ruled by instructions stored in its genome, encoded in sequences of nucleic acids. Comparison between the Turing machine and the intracellular processing of DNA and RNA show remarkable similarities. Both systems process information stored in a string of symbols built upon a fixed alphabet, and both operate by moving step-by-step along those strings, modifying or adding symbols according to a given set of

E. Keinan (✉)
Department of Chemistry, Technion—Israel Institute of Technology, Technion City,
Haifa 32000, Israel
e-mail: keinan@tx.technion.ac.il

E. Keinan
Department of Molecular Biology and The Skaggs Institute for Chemical Biology,
The Scripps Research Institute, 10550 North Torrey Pines Road, La Jolla, CA 92037,
USA

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_25, © Springer-Verlag Berlin Heidelberg 2009

rules. These parallels have inspired the idea that biological molecules could become the raw material of a new computer species. Such biological computers would not necessarily offer greater power or performance in traditional computing tasks. Obviously, the speed of natural molecular machines, which depend on the catalytic rates of enzymes, is significantly slower than that of electronic devices, which perform billions of gate-switching operations per second. However, bio-molecules possess the unique ability of speaking the language of living cells [4].

DNA is an obvious choice for making the basic building blocks for bio-computers, mainly due to the high density of information stored in DNA strands, the ease of constructing many copies of them, and the ready availability and high fidelity of various DNA enzymes. The base-pairing rules can serve as a programming language, and the comprehensive knowledge of DNA handling, editing, amplification, and detection enables designing and predicting computations.

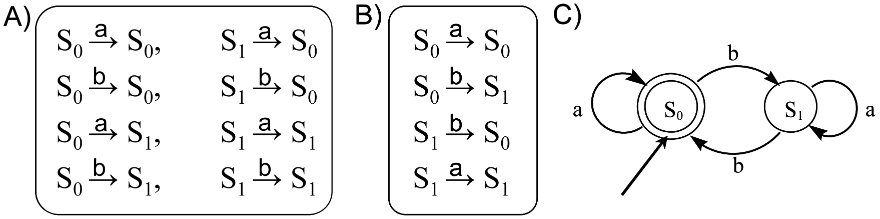
The growing interest in the design of new computing systems is attributed to the notion that although the silicon-based computing provides outstanding speed, it cannot meet some of the challenges posed by the developing world of bio-technology. New abilities, such as direct interface between computation processes and biological environment are necessary. In addition, the challenges of parallelism [5] and miniaturization are still driving forces for developing innovative computing technologies. Moreover, the growth in speed for silicon-based computers, as described by Moore’s law, may be nearing its limit [6]. The design of new computing architectures involves two main challenges: reduction of computation time and solving intractable problems. Most of the celebrated computationally intractable problems can be solved with electronic computers by an exhaustive search through all possible solutions. However, an insurmountable difficulty lies in the fact that such a search is too vast to be carried out using the currently available technology.

In his visionary talk in 1959, Feynman suggested to use atomic and molecular scale components for building machinery [7]. This idea has stimulated several research studies, but it was not until 1994 that an active use of molecules was presented in the form of computation [8]. Numerous architectures for autonomous molecular computing have been developed over the years on the basis of molecular biology opportunities [9–17]. Some of these have been explored experimentally and proved to be feasible [9, 18–21]. This account focuses on the realization of prorammable finite-state automata in our labs that can compute autonomously upon mixing all their components in solution.

2 Bio-molecular Finite Automata

2-Symbols-2-States Soluble DNA-Based Finite Automata

Our design of bio-molecular finite automata (Scheme 1) [18] was based on the basic principles of the Turing machine [22]. The hardware comprised of FokI, a type-II endonuclease restriction enzyme, T4 DNA ligase, and ATP. The software comprised


A)
$S_0 \xrightarrow{a} S_0$,
$S_0 \xrightarrow{b} S_0$,
$S_0 \xrightarrow{a} S_1$,
$S_0 \xrightarrow{b} S_1$,
$S_1 \xrightarrow{a} S_0$
$S_1 \xrightarrow{b} S_0$
$S_1 \xrightarrow{a} S_1$
$S_1 \xrightarrow{b} S_1$
B)
$S_0 \xrightarrow{a} S_0$
$S_0 \xrightarrow{b} S_1$
$S_1 \xrightarrow{b} S_0$
$S_1 \xrightarrow{a} S_1$
C)
b
a
S_0
S_1
a
b

Scheme 1 Two-symbol two-state automaton. (**A**) The 8 possible transition rules of a finite automaton having two internal states (S_0 and S_1) and two input symbols (a and b). (**B**) A selected subset of the 8 rules that represents 4 transition rules of a finite automaton. (**C**) A graphic description of this automaton that includes 2 states: S_0 and S_1 (indicated by *circles*), 2 symbols: a and b, a start state S_0 (indicated by a *straight arrow*) and 4 transition rules (indicated by *curved arrows*). This automaton answers the question whether the number of b's in a given input is even. A positive answer to this question will result in the accepting state S_0 (indicated by a *double circle*). On the other hand, an even number of b's in the input will result in a final state S_1

of transition molecules in the form of short double stranded (ds) DNA molecules, which encoded for the automaton transition rules. A dsDNA molecule encoded the initial state of the automaton and the input, with each input symbol being coded by a six base-pair (bp) dsDNA sequence. The system also contained 'peripherals', two output-detection molecules of different lengths. Each of these could interact selectively with a different output molecule to form an output-reporting molecule that indicated a final state and could be readily detected by gel electrophoresis (Fig. 1).

The two different internal states, either S_0 or S_1, were represented by two distinguishable restriction modes of any 6-bp symbol, either at the beginning of the symbol domain or 2-bp deeper into that domain, respectively (Fig. 2). The different cleavage site was achieved by using a type-II, 4-cutter endonuclease, FokI, which cuts 9 and 13 bases away from its recognition sites. In this system, there were 6 unique 4-nucleotide 5′-prime sticky end sequences that could be obtained by two restriction modes of 2 symbols and a terminator.

The automaton processed the input as shown in Scheme 2. First, the input was cleaved by FokI, thereby exposing a 4-nucleotide sticky end that encoded the initial state and the first input symbol. The computation proceeded via a cascade of transition cycles. In each cycle, the sticky end of an applicable transition molecule ligated to the sticky end of the input molecule, detecting the current state and the current symbol. The product was cleaved by FokI inside the next symbol, exposing a new four-nucleotide sticky end. The design of the transition molecules ensured that the 6-bp-long encodings of the input symbols a and b were cleaved by FokI at only two different 'frames', the leftmost frame encoding the state S_1 and the rightmost frame encoding S_0. The exact next restriction site, and thus the next internal state were determined by the current state and the size of the spacers in an applicable transition molecule (Fig. 1). The computation proceeded until no transition molecule matched the exposed sticky end of the input or until the special terminator symbol was cleaved, forming an output molecule that has a sticky end encoding the final state. This sticky end ligated to one of two output detectors and the resultant output

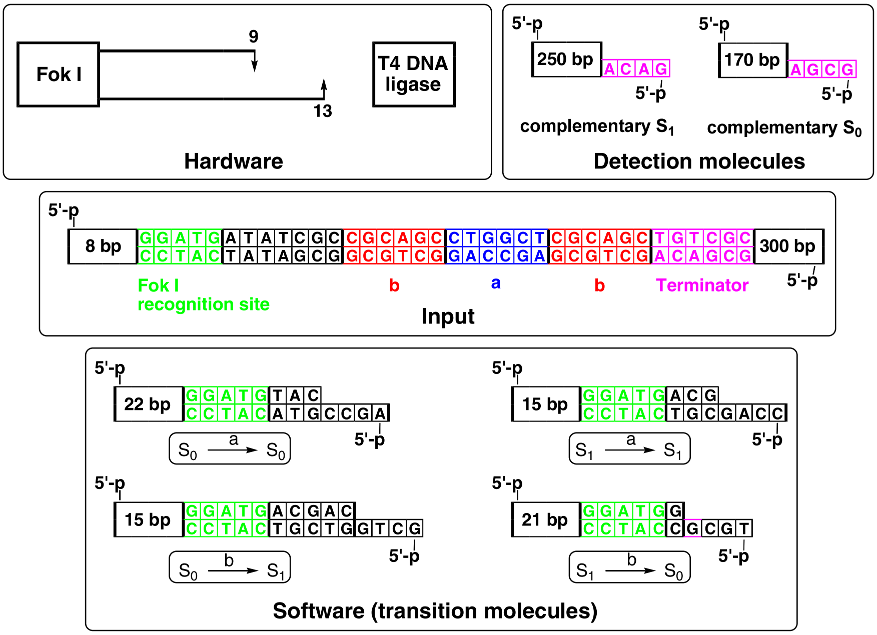


Fig. 1 Molecular design of a 2-symbol-2-state finite automaton. The 10 components of the automaton include an input molecule, 2 enzymes, ATP, 4 transition molecules, and 2 detection molecules. The 4 transition molecules shown here, which represent one specific program, are chosen from a library of 8 transition molecules, and construct the automaton presented in Scheme 1

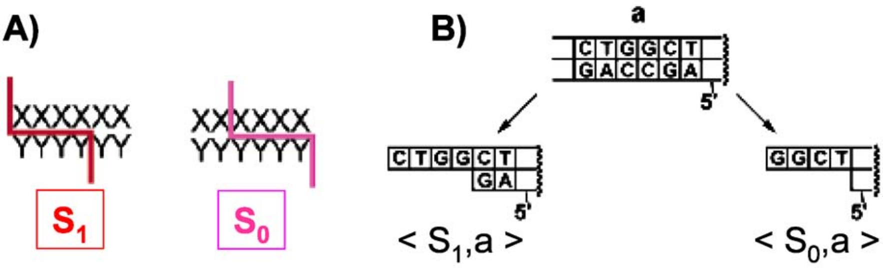
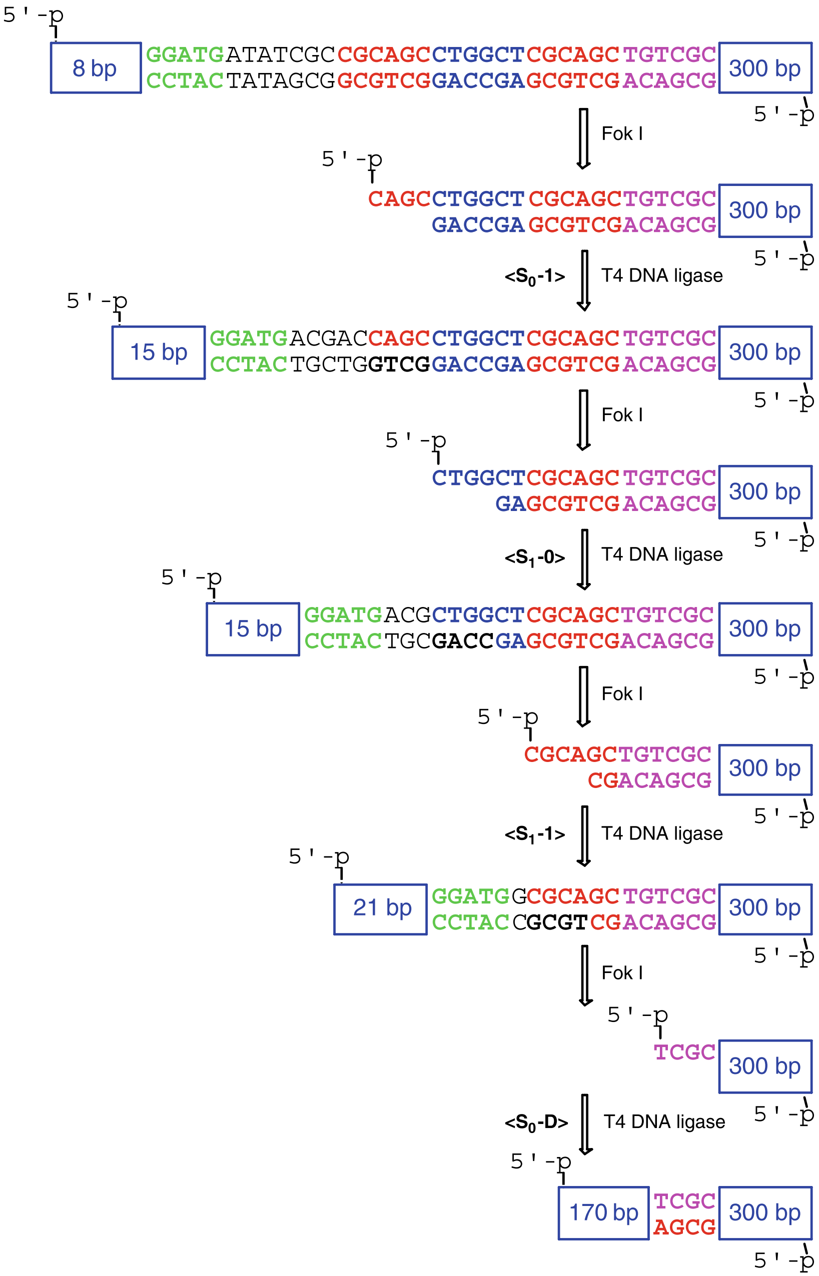


Fig. 2 Two different internal states. (**A**) Two restriction modes of a 6-bp domain by a 4-cutter enzyme. (**B**) The symbols 'a' with its restriction products produced by a 4-cutter endonuclease

reporter was identified after purification by gel electrophoresis. The operation of the automata was tested by running it with a selection of programs on various inputs and detected by gel electrophoresis (Fig. 3).

Based on this design, a ligase-free system was also demonstrated. In this manner, software molecules were not consumed but the input consumption drove the biomolecular computations [23].



Scheme 2 Description of a computing process with input '*bab*' [24]. The input is cleaved by FokI, forming the initial state. The complementary transition molecule (⟨S_0-1⟩) is hybridized and in the presence of T4 DNA ligase it is ligated to the restricted input. Similarly, the input is processed via repetitive cycles of restriction, hybridization and ligation with complementary transition molecules. In the final step, the output is ligated to S_0 detection molecule (⟨S_0-D⟩)

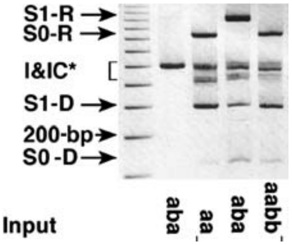


Fig. 3 Running automaton programs on various inputs. In each lane, the injected molecules are the detection-molecules S_0–D and S_1–D, molecules encoding intermediate configurations capped by yet-to-be-processed input molecules and output-reporting molecules S_0–R or S_1–R. Expected locations of different species and a size marker of 200 bp are indicated by *arrows to the left*. Input molecules are specified below each lane. Expected outputs (S_0 or S_1) are S_1–R for input aa, S_1–R for input aba, and S_0–R for input aabb

Immobilized DNA-Based Automata

We have demonstrated that a type-II, 4-cutter endonuclease, can restrict a 6-bp symbol in three distinguishable modes [24]. Three internal states, S_0, S_1, or S_2, could thus be represented by restriction at the beginning of the symbol domain, 1-bp deeper or 2-bp deeper into that domain, respectively (Fig. 4). Thus, a 3-state automaton with 2 symbols would have a total library of 12 transition rules with a much broader spectrum of possible programs as compared with the previously reported 2-symbol-2-state case [18].

The increased number of states and symbols resulted in significant enhancement of the computing power. For example, with a 3-symbol-3-state device (Scheme 3) there is a library of 27 possible transition rules and 134,217,727 possible selections of transition-rule subsets from that library. Since there are 7 possible selections of the accepting states (S_0, S_1, S_2, any combination of two and a combination of all three), the outcome becomes a remarkably large number of 939,524,089 syntactically distinct programs. This number is considerably larger than the corresponding number of 765 possible programs offered by our previously reported 2-symbol-2-state device [18]. Theoretically, the bio-molecular automata can be further expanded to a 37-symbol-3-state automaton.

The applicability of this design was further enhanced by employing surface anchored input molecules, using the surface plasmon resonance (SPR) technology to monitor the computation steps in real-time. Computation was performed by alternating the feed solutions between endonuclease and a solution containing the ligase, ATP, and appropriate transition molecules. The output detection involved final ligation with one of three detection molecules solubles. The process occurred in a stepwise manner, with automatic detection of each step, on a Biacore® chip. The binding and computing events were monitored while taking place at the sensor surface.

The realization of this computational design was achieved as follows: the chip was loaded with biotinylated input molecules. Computation was carried out by al-

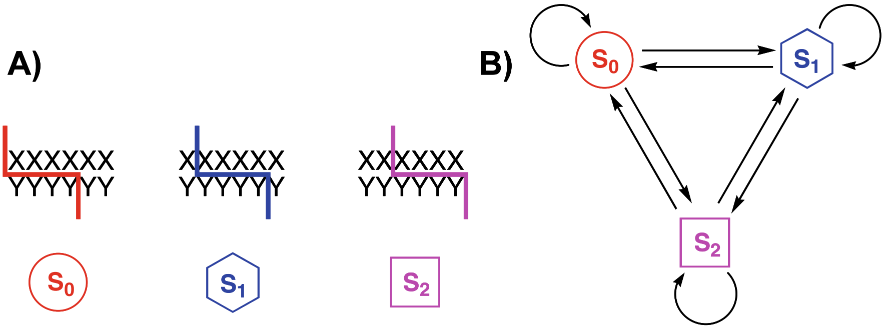
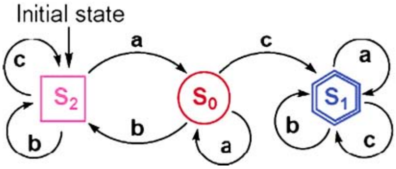


Fig. 4 Three-state automaton design. (**A**) Three restriction modes by a 4-cutter enzyme representing three internal states. (**B**) Nine transition rules for each symbol lead to a total library of 27 possible rules



$S_2 \xrightarrow{a} S_0$	$S_0 \xrightarrow{a} S_0$	$S_1 \xrightarrow{a} S_1$
$S_2 \xrightarrow{c} S_2$	$S_0 \xrightarrow{c} S.$	$S_1 \xrightarrow{c} S_1$
$S_2 \xrightarrow{b} S_2$	$S_0 \xrightarrow{b} S_2$	$S_1 \xrightarrow{b} S_1$

Scheme 3 A 3-symbol-3-state finite automaton. The automaton is shown both graphically (*left*) and in the form of a table of nine transition rules (*right*)

ternating the feed solution between the above-described restricting and ligating solutions. The flow cell was first fed with a solution of BbvI endonuclease, then fed with a mixture of transition molecules and ligase, and so forth. Detection of the final state was carried out by sequential feed of three mixtures, each containing one of the detection molecules, D–S_0, D–S_1, or D–S_2, together with T4-DNA ligase and ATP. As the Biacore chip contains 4 independent sectors, it was possible to perform parallel computing with 4 different input molecules. This advantage was demonstrated by stepwise monitoring of the computation using one automaton and four different inputs: bc, a, ac, acb (Fig. 5).

DNA-Based Automaton with Bacterial Phenotype Output

We have recently demonstrated for the first time that the output of a computation produced by a molecular finite automaton can be a visible bacterial phenotype [25]. The 2-symbol-2-state finite automaton, which was described in Scheme 1 and Fig. 1, utilized linear dsDNA inputs that contained strings of six base pair symbols. We prepared encased inputs by inserting such strings into the lacZ gene on the pUC18 plasmid (Scheme 4). The computation resulted in a circular plasmid that differed from the original pUC18 by either a 9 base pair (accepting state) or 11 base pair

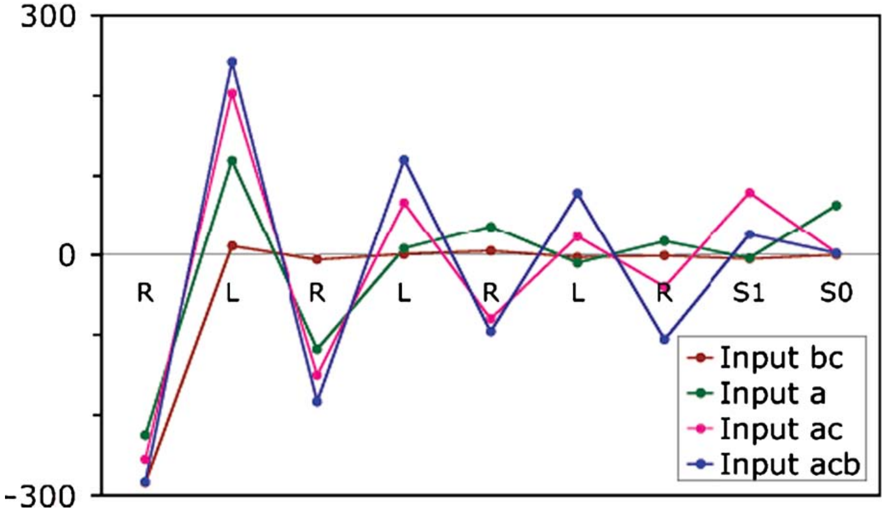
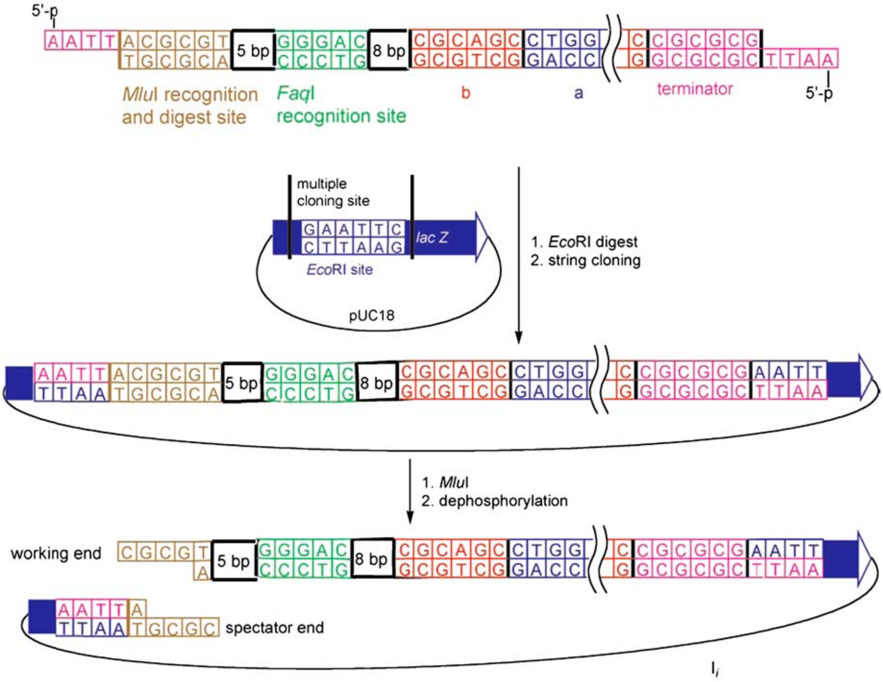


Fig. 5 Monitoring the computation process by SPR [24]. Stepwise computing with one automaton (Scheme 3) and four input molecules: bc, a, ac, and acb. The transition molecules were supplied only to satisfy the computation needs of the latter three inputs, while no transition molecules were available for computation with the bc input. The differential RU values represent the changes in the SPR response between two consecutive steps. R represents restriction; L represents ligation. The computation was followed by detection with the soluble detection molecules, D–S_0 and D–S_1

insert (unaccepting state) within the lacZ α region gene. Upon transformation and expression of the resultant plasmids in E. coli, the accepting state was represented by production of functional β-galactosidase and formation of blue colonies on X-gal medium. In contrast, the unaccepting state was represented by white colonies due to a shift in the open reading frame of lacZ.

Construction of the extended input molecules was carried out by cloning an insert into a multiple cloning site (MCS) of the vector pUC18 (Scheme 4). This insert comprised all the previously described components, including several 6-bp symbols, a 6-bp terminator, and recognition sites of restriction enzymes. The inserted computing cassette contains a restriction site for MluI (ACGCGT), upstream of FaqI recognition site (the working enzyme). Thus, digestion of the plasmid with MluI converted the plasmid into a linear dsDNA with a GCGC-5′ sticky end (the spectator end). This choice dictated that the specific sequence of the terminator would be CGCGCG so that the final obtained sticky end will be complementary to the spectator one independent of the final state (5′-CGCG).

The chosen automaton accepted inputs with an even number of b symbols, which means that if the dsDNA molecule input contained an even number of b segments the computation would lead to a 9-bp insert in the pUC18 plasmid and formation of blue colonies on X-gal medium. In contrast, if the input string contained an odd number of b symbols, the computation would result in an 11-bp insert, ORF shift, and hence formation of white colonies on X-gal medium (Scheme 5).

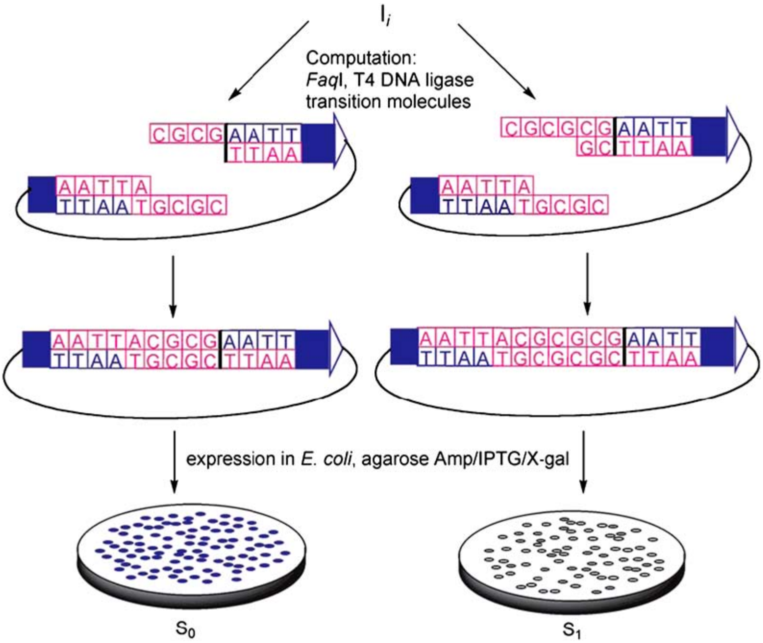


Scheme 4 General preparation of a circular input molecule [25]. Cloning an input insert into the multiple cloning site of pUC18. Digestion of the plasmid with MluI converted the plasmid into a linear dsDNA, with a 5′-CGCG overhang on the upper string and GCGC-5′ on the lower string

Parallel Computation by DNA Array

As discussed previously, one of the main advantages of these bio-molecular devices is their ability to perform parallel computing. In addition, it has been demonstrated that the use of SPR technology allows for real-time detection of the output signal, as well as for real-time monitoring of all computation intermediates [24]. One drawback of this type of molecular finite automata [18, 23–25] is the complete consumption of the input information. Consequently, although simultaneous multiple computations were possible, they all involved loss of the original input information and, therefore, had an inability to correlate that information directly to the output. Nevertheless, the successful monitoring of the computation processes by using immobilized input molecules suggested that input immobilization could enable parallel computation, where the input location on the chip would represent specific tagging. The immobilization of multiple inputs on a chip would allow for the detection of all outputs, as well as direct assignment of each output to its original input.

We are currently focusing on the unprecedented design of more advanced computation arrays, based on combinations of the SPR technology with broadened types of finite automata. The application of the SPR technology is implemented with the ProteOn, which has been recently developed by Biorad-Haifa that enables parallel



Scheme 5 Computation resulting in bacterial phenotype output. *Left*: computation with an input that contains an even number of b symbols leads to blue bacterial colonies output (S_0); *right*: computation with an input that contains an odd number of b symbols leads to white bacterial colonies output (S_1)

processing in six independent microfluidic channels. The ProteOn chip is capable of rotating 90°, resulting in six plumb channels, allowing for simultaneous computations with six different immobilized inputs in six horizontal rows. Each input can be processed with six different soluble automata in six vertical columns (Fig. 6). In this work, we implement 3-symbols-3-state automata processing 4-symbols-long input. By using a microfluidic system, the vertical grooves are each fed a different solution containing transition molecules from six different automata, one per groove. The soluble automata process each input in the same manner that has been described previously, to finally produce an immobilized output, which is detectable by the SPR technology. Moreover, this work expands on the previously described finite automata, by performing computations utilizing some 3-symbol-3-state finite non-complete and non-deterministic automata (stochastic).

This matrix approach demonstrates the power of parallel computation with 36 different computations occurring simultaneously, each can result in one of eight possible output signals: any one of the three states; any combination of two of them; all three; or none (in case the computation is suspended), allowing a total of 8^{36} different chip designs. If each one of the potential 8 outputs in every junction encodes for a different color of a specific pixel, then each chip design represents a different image. Thus, the overall number of possible images can be as large as 8^{36}

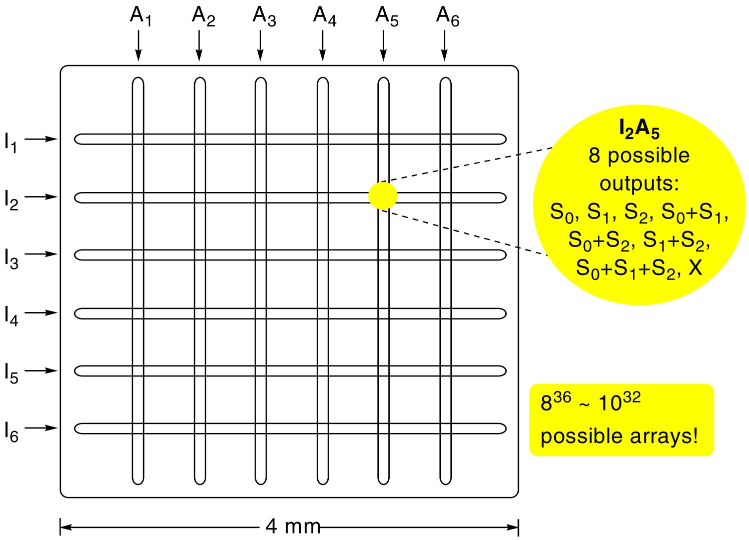

A1
A2
A3
A4
A5
A6
I1
I2
I3
I4
I5
I6
I2A5
8 possible outputs:
S0, S1, S2, S0+S1, S0+S2, S1+S2, S0+S1+S2, X
8^36 ~ 10^32 possible arrays!
4 mm

Fig. 6 A 6 × 6 array of different inputs, $I_{1\text{–}6}$, (*rows*) and different automata, $A_{1\text{–}6}$, (*columns*). The computation results in one out of eight possible output signals for each junction (pixel), either one of the three states, any combination of two of them, all three, or none (in case the computation is suspended). Thus, the overall number of possible pictures can be as large as 10^{32}

or 10^{32}. Since in this work, each computation includes 4 sequential steps, there are 144 computational steps per chip.

3 Conclusions

This work presents significant enhancement of the computational power in comparison with our initially reported 2-symbol-2-state automata [18]. The major improvements include (a) the increase of the number of internal states, (b) the increase of the number of symbols, (c) the use of the SPR technology, which permits real time detection of the output signal as well as real time monitoring of stepwise computation, and (d) the immobilization of the input molecules on a chip, thus enabling performing parallel computing on different inputs. The first two improvements increase the overall number of syntactically distinct programs from 48 to 137,781. Theoretically, automata with up to 37 different symbols, 6 bp each, could be created on the basis of this technology.

The ever-increasing interest in BMC devices has not arisen from the hope that such machines could compete with their electronic counterparts by offering greater computation speed, fidelity, and power or performance in traditional computing tasks. The main advantage of autonomous BMC devices over the electronic computers arises from their ability to interact directly with biological systems and even with living organisms without any interface. We describe, for the first time, an automaton that produces the computational output in the form of meaningful, expressible in-vivo dsDNA [25]. These results demonstrate that an appropriately designed computing machine can produce an output signal in the form of a specific biological function via direct interaction with living organisms. The next steps along this line would be the insertion of a complete computing device into a living cell or a tissue, with the long-term goal of utilizing BMC devices for in-vivo diagnostics and disease control, or a design of new types of biological regulation.

References

1. Reif JH (2002) Science 296:478–479
2. Chen J, Wood DH (2000) Proc Natl Acad Sci USA 97:1328–1330
3. Seeman NC (2003) Chem Biol 10:1151–1159
4. Shapiro E, Benenson Y (2006) Scientific American, INC 45–51
5. Livstone S, Van Noort D, Landweber LF (2003) Molecular computing revisited: a Moore's law? Trends Biotechnol 21:98–101
6. Ruben AJ, Landweber LF (2000) Nat Rev Mol Cell Biol 1:69–72
7. Feynman R (1961) In: Gilbert D (ed) Miniaturization. Reinhold, New York, pp 282–296
8. Adleman LM (1994) Science 266:1021–1024
9. Lipton RJ (1995) Science 268:542–545
10. Liu Q, Wang L, Frutos AG, Condon AE, Corn RM, Smith LM (2000) Nature 403:175–179
11. Sakamoto K, Gouzu H, Komiya K, Kiga D, Yokoyama S, Yokomori T, Hagiya M (2000) Science 288:1223–1226
12. Faulhammer D, Cukras AR, Lipton RJ, Landweber LF (2000) Proc Natl Acad Sci USA 97:1385–1389
13. Braich RS, Chelyapov N, Johnson C, Rothemund PW, Adleman L (2002) Science 296:499–502
14. Roweis S, Winfree E, Burgoyne R, Chelyapov NV, Goodman MF, Rothemund PW, Adleman LM (1998) J Comput Biol 5:615–629
15. Winfree E, Liu F, Wenzler LA, Seeman NC (1998) Nature 394:539–544
16. LaBean TH, Winfree E, Reif JH (1999) In: Winfree E, Gifford D (eds) DNA based computers V. American Mathematical Society, Cambridge, pp 123–140
17. Winfree E (1999) J Biomol Struct Dyn, 263–270
18. Benenson Y, Paz-Elizur T, Adar R, Keinan E, Livneh Z, Shapiro E (2001) Nature 414:430–434
19. Mao C, LaBean TH, Relf JH, Seeman NC (2000) Nature 407:493–496
20. Rose JA, Deaton RJ, Hagiya M, Suyama A (2002) Phys Rev E Stat Nonlinear Soft Matter Phys 65:021910
21. Komiya K, Sakamoto K, Gouzu H, Yokoyama S, Arita M, Nishikawa A, Hagiya M (2001) In: 6th international workshop on DNA-based computers. Springer, Leiden, pp 19–26
22. Turing AM (1936–1997) Proc Lond Math Soc 42:230–265
23. Benenson Y, Adar R, Paz-Elizur T, Livneh Z, Shapiro E (2003) Proc Natl Acad Sci USA 10:2191–2196
24. Soreni M, Yogev S, Kossoy E, Shoham Y, Keinan E (2005) J Am Chem Soc 127:3935–3943
25. Kossoy E, Lavid N, Soreni-Harari M, Shoham Y, Keinan E (2007) Chem Biol Chem 8:1255–1260

Part VIII Biochemical Reactions

A Multi-volume Approach to Stochastic Modeling with Membrane Systems

Daniela Besozzi, Paolo Cazzaniga, Dario Pescini, and Giancarlo Mauri

Abstract In the last decades, experimental investigations have evidenced the role of biological noise in cellular processes, and several stochastic approaches have been proposed to modeling and simulation of biochemical networks. Here, we review the main stochastic procedures defined for single-volume biochemical systems (SSA, tau-leaping), and discuss their practical utility and limitations. Then, within the framework of membrane systems, we propose a multi-volume generalization of the tau-leaping algorithm, called τ-DPP, feasible for the stochastic analysis of complex biochemical systems. Finally, we present a case-study application of τ-DPP to an intracellular genetic oscillator, coupled with an intercellular communication mechanism.

1 Introduction

After the completion of the human genome sequencing (and of a lot of other genomes), the main challenge for the modern biology is to understand complex biological processes such as metabolic pathways, gene regulatory networks and cell signaling pathways, which are the basis of the functioning of the living cell. This goal can only be achieved by using mathematical modeling tools and computer simulation techniques, to integrate experimental data and to make predictions on the system behavior that will be then experimentally checked, so as to gain insights into the working and general principles of organization of biological systems.

In fact, the formal modeling of biological systems allows the development of simulators, which can be used to understand how the described system behaves in normal conditions, and how it reacts to (simulated) changes in the environment and to alterations of some of its components. Simulation presents many advantages over conventional experimental biology in terms of cost, ease to use and speed. Also, experiments that are infeasible in vivo can be conducted in silico, e.g. it is possible to knock out many vital genes from the cells and monitor their individual and collective impact on cellular metabolism. Evidently such experiments cannot be done in vivo because the cell may not survive. Development of predictive in silico models offers opportunities for unprecedented control over the system.

D. Besozzi (✉)
Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano,
Via Comelico 39, 20135 Milano, Italy
e-mail: besozzi@dico.unimi.it

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_26, © Springer-Verlag Berlin Heidelberg 2009

In the last few years, a wide variety of models of cellular processes have been proposed, based on different formalisms. For example, chemical kinetic models attempt to represent a cellular process as a system of distinct chemical reactions. In this case, the network state is defined by the instantaneous quantity (or concentration) of each molecular species of interest in the cell, and molecular species may interact via one or more reactions. Often, each reaction is represented by a differential equation relating the quantity of reactants to the quantity of products, according to a reaction rate and other parameters.

Recently, it has been pointed out that transcription, translation, and other cellular processes may not behave deterministically but instead are better modeled as random events [20]. Models have been investigated that address this concern by abandoning differential equations in favor of stochastic relations to describe each chemical reaction [1, 16].

At a different level of abstraction many formalisms, originally developed by computer scientists to model systems of interacting components, have been applied to biology. Among these, there are Petri nets, Hybrid Systems, and the π-calculus. Moreover, formalisms such as P systems, originally proposed as a model of computation inspired by biology, have recently found application to the formal description and modeling of biological phenomena.

In the next section, we briefly discuss the role of stochasticity in biology, and present the main stochastic approaches to modeling and simulation of biochemical networks (proposed by Gillespie [6, 16]), and their limitations, mainly due to computational burden. In Sect. 3, the basic definitions concerning P systems and their dynamical probabilistic version, DPP [27–29], will be given, and then it will be shown how they can constitute the basic formalism to support a multi-volume generalization of Gillespie's tau-leaping algorithm [7]. In Sect. 4, a model of a genetic oscillator coupled with a quorum sensing intercellular mechanism [11, 14] will be considered as a case study, and the results of simulations of this model will be presented. We will conclude with a final discussion of some issues concerning possible improvements and future extensions of our multi-volume approach to stochastic modeling.

2 Stochastic Approaches to Modeling and Simulation of Biological Systems

In this section, we treat of the role of biological noise as the triggering mechanism of stochasticity in biochemical systems, and discuss about the deterministic formulation and the stochastic approach in the modeling and simulation of biological systems. Then we review some main stochastic algorithms for single-volume biochemical systems, i.e. the stochastic simulation algorithm and tau-leaping, and comment on their strength and limitations.

2.1 Stochasticity in Chemical and Biological Systems

If we model a biological process as a network of biochemical reactions, our goal will be to simulate the evolution in time of the quantities of different chemical species in the system. To this aim, it is usual to give a deterministic formulation through a set of coupled ordinary differential equations (ODEs). This traditional approach has at its core the *law of mass action*, an empirical law giving a simple relation between reaction rates and molecular component concentrations. Hence, given the knowledge of initial molecular concentrations, the law of mass action provides a complete picture of the component concentrations at all future time points. So doing, we assume a macroscopic point of view, which deals with the aggregated quantity of substances and the rate of change of concentrations. Over many years, powerful algorithms with sound mathematical basis have been developed for solving these equations.

This is sufficient in many cases but, for example, fails to capture the inherent stochasticity of biochemical systems in which the small population of a few critical reactant species can result in stochastic behaviors. In this case, we should consider a microscopic point of view, keeping track of every molecule in the system and of the single reactions in which it is involved, which are chosen in a way that is random or stochastic in nature. For this reason, stochastic modeling is recently gaining increasing attention in the study of biological systems, since "noise" and discreteness have been revealed to play an important role in cellular processes involving a few molecules.

Indeed, the number of publications that investigate the experimental evidences of noise is continuously growing (see, e.g. [4, 13, 21, 33] and references therein). There exist two main sources of biological noise. The *extrinsic* source is due to the experimental conditions, such as the fluctuations in temperature, pressure and light, or due to other cellular factors. The *intrinsic* source is due to the variability of the chemical reactions which, from a microscopic point of view, correspond to a scattering event among reactants, and consequently it undergoes temporal and spatial fluctuations. Works like [12, 30, 31] quantify the role of noise in gene expression, and show the inadequacy of the classical deterministic and continuous modeling approach to describe phenomena such as inter and intra-cellular signaling pathways (especially when the transcription and translation machineries are involved), therefore supporting the need of stochastic modeling approaches.

2.2 Single-Volume Stochastic Simulation Algorithms

The *stochastic simulation algorithm* (SSA), presented by Gillespie in [16], is one of the first methods developed and used to describe the dynamical evolution of chemical systems. This procedure provides the exact behavior of a system, since it is proved to be equivalent to the Chemical Master Equation (CME). The requisites needed for the application of the SSA are related to the definition of the volume where all the reactions take place. The SSA can deal only with single volume systems. The conditions (pressure, temperature, etc.) of the volume must be kept constant during the evolution and, moreover, all the molecular species are considered well mixed (uniformly distributed inside the volume).

The description of the system is given by means of a set of chemical species, coupled with their quantities, to denote the current state. On the other hand, a set of chemical reactions specifies how the chemical species interact and evolve. Moreover, there is a stochastic constant associated with each reaction. Each constant reflects the physical and chemical properties of the species involved in the reaction, and it is used to compute the probabilities of the reactions application (called *propensity functions*).

Gillespie, in [16], has formally shown how starting from the application probabilities of the reactions, it is possible to find, in each step of the algorithm, the time increment τ and the index μ of the reaction that will be executed during τ. Therefore, finding τ and μ at each iteration of the algorithm, makes it possible to *sequentially* describe the behavior of the system. This is achieved by considering every single reaction that happens inside the volume, and modifying the quantities of the species involved in the selected reaction.

The main problem of the SSA regards the computational time needed to perform the simulation of a chemical system. The algorithm complexity increases with the number of species and the number of reactions of the system, hence many real problems cannot be efficiently solved using this technique.

In order to work out the problem related to the huge computational time required for the execution of SSA, Gibson and Bruck presented in [15] a novel procedure, called "Next Reaction Method", able to describe the exact behavior of a chemical system, with better performances than SSA. Gibson and Bruck proved that it is possible to use a single random number for each reaction event, instead of extracting new random numbers at each iteration of the algorithm (as in SSA). Moreover, the complexity of the next reaction method is proportional to the logarithm of the number of reactions (and not to the number of reactions, as in SSA). The basic idea is to compute the propensity functions of the reaction, and to associate to each reaction a putative time. The evolution is then described executing the reaction with the shortest time. After the execution of the rule, the only propensity functions and putative times that are recalculated are those affected by the executed reaction.

Another technique used to speed up stochastic simulation performed by means of SSA was first introduced by Gillespie and Petzold in [17] and it is called *tau-leaping procedure*. Here, instead of describing the dynamics of the system by tracing every single reaction event occurring inside the volume, a time increment τ is computed and a certain number of reactions are selected (according to specified conditions) and executed in parallel. The obtained behavior of the chemical system is not exact, as in SSA, but it is approximated.

Several different versions of the tau-leaping algorithm have been proposed, aimed at improving the procedure to compute the τ value and to select the reactions to be applied in the current step, thus avoiding the possibility to obtain negative population of chemical species (we refer to [5, 8, 18, 32] for more details). Despite the improvements in the simulation method achieved by these techniques, they all present a problem related to the description of the system dynamics: though an error control parameter is used, they do not allow to uniformly bound the changes of the species quantities during the τ selection procedure therefore resulting in a poor approximation of the system behavior. Moreover, in order to compute the time increment at each step, they require the evaluation of a quadratic number of auxiliary quantities (relative to the number of chemical reactions).

For these reasons, we refer to the tau-leaping version presented in [6], where the authors worked out the problems related to the approximation and to the evaluation of the step size. Here, the approximation of the system dynamics is guaranteed by the *leap condition*, a condition used to uniformly bound the changes in the propensity functions of the reactions during the step. Moreover, a new τ selection strategy is exploited, and only a number of auxiliary quantities which increases linearly with the number of reactant species needs to be computed.

Furthermore, to avoid the possibility of obtaining negative molecular populations (due to the application of a number of reactions greater than the number of molecules currently present inside the volume), it is necessary to identify the subsets of *critical* and *non-critical reactions*: a reaction is called critical, if its reactants are present inside the system in very small amounts. Exploiting the definition of the two reactions subsets, it is then possible to select a number of reactions that will be applied during the iteration, preventing the negative quantities. In general, by using the tau-leaping procedure, a system can evolve following three different ways: (1) execute a SSA-like step if, according to the current system state, it is better to apply a single reaction; (2) execute a set of non-critical reactions and one critical reaction; and (3) execute a set of non-critical reactions only.

The SSA algorithm, the next reaction method, and the (different) tau-leaping procedures are only applicable to chemical systems enclosed in a single volume. In order to overcome this limitation, novel approaches have been introduced. One of these approaches, called τ-DPP algorithm [7], is a method based on the tau-leaping procedure that enables to describe the dynamics of multi-volume systems (where every volume is defined under the usual SSA conditions, such as constant temperature, pressure, etc., and uniformly distributed molecules). This procedure will be explained in Sect. 3.3.

Besides, other techniques have been developed to simulate systems with spatial heterogeneity, e.g. the next sub-volume method [10] and the binomial τ-leap spatial stochastic simulation algorithm [19]. Here, the purpose is to deal with single volume systems where the molecular species are not uniformly distributed. In order to exploit the Gillespie's method to evaluate the probabilities of the reactions and to manage the diffusion of molecules, the volume of the system is divided into separated sub-volumes. Each sub-volume is taken small enough to be considered well mixed. Therefore, it is possible to identify the reactions and the diffusion processes that take place inside every single sub-volume of the system. The difference between these two methods is that, the next sub-volume method is based on the next reaction method by Gibson and Bruck (hence it provides the exact behavior of the system), while the binomial τ-leap spatial stochastic simulation algorithm is based on the tau-leaping method (hence it provides an approximated behavior, but it improves the simulation performances).

3 A Multi-volume Approach with Membrane Systems

In this section, we recall the basic notions of membrane systems, or P systems, and discuss some issues about their possible application in the study of biochemical systems. In particular, we consider the problem of maximal parallelism, that is inherent in the basic definition of P systems, and we describe some possible ways to overcome this obstacle. In particular, we discuss about the introduction of probabilities for rule application.

Then we present τ-DPP, a computational method recently appeared in [7], which can be used to describe and perform stochastic simulations of complex biological or chemical systems. The "complexity" of the systems that can be managed by means of τ-DPP, resides not only in the number of the (chemical) reactions and of the species involved, but mainly in the topological structure of the system, that can be composed by many volumes. For instance, cellular pathways involving several spatial compartments (as the extracellular ambient, the cytoplasm, the nucleus, etc.), or multicellular systems like bacterial colonies, or multi-patched ecological systems as meta-populations, are all examples of biological systems that could be investigated with τ-DPP. Since τ-DPP represent a general simulating framework for a broad range of complex systems, in the following we will use the generic terms *volume*, *object,* and *rule*, to denote the compartment or region, where the molecular species, or any other elemental "species", can be modified in some way by a biochemical or, more generally, an interspecies reaction.

3.1 Membrane Systems

P systems, or membrane systems, were introduced in [25] as a class of unconventional computing devices of distributed, parallel, and non-deterministic type, inspired by the compartmental structure and the functioning of living cells. A basic P system is defined by a membrane structure where multi-sets of objects evolve according to given evolution rules, which also determine the communication of objects between membranes. A *membrane structure* consists of a set of membranes hierarchically embedded in a unique membrane, called the skin membrane. The membrane structure is represented by a string of correctly matching square parentheses, placed in a unique pair of matching parentheses. Each pair of matching parentheses corresponds to a membrane, and usually membranes are univocally labeled with distinct numbers. For instance, the string $\mu = [_0\ [_1\ ]_1\ [_2\ [_3\ ]_3\ [_4\ ]_4\ ]_2\ ]_0$ corresponds to a membrane structure consisting of 4 membranes placed at three hierarchical levels. Moreover, the same membrane structure can be also represented by the string $\mu' = [_0\ [_2\ [_4\ ]_4\ [_3\ ]_3\ ]_2[_1\ ]_1\ ]_0$, that is, any pair of matching parentheses at the same hierarchical level can be interchanged, together with their contents; this means that the order of pairs of parentheses is irrelevant, what matters is their respective relationship. Each membrane identifies a *region*, delimited by it and the membranes (if any) immediately inside it. The number of membranes in a membrane structure is called the *degree* of the P system. The whole space outside the skin membrane is called the *environment*.

An *object* can be a symbol or a string over a specified finite alphabet V; multisets of objects are usually considered in order to describe the presence of multiple copies of any given object inside a membrane. A multi-set associated with membrane i is a map $M_i : V^* \to \mathbf{N}$ which associates a multiplicity to each object in the multi-set itself. Inside any membrane i, objects in M_i are modified by means of *evolution rules*, which are multi-set rewriting rules of the form $r_i : u \to v$, where u and v are multi-sets of objects. The objects from v have associated target indications which determine the regions where they are to be placed (communicated) after the application of the rule: if $tar = here$, then the object remains in the same region; if $tar = out$, then the object exits from the region where it is placed and enters the outer region (or even exits the system, if the rule is applied in the skin membrane); if $tar = in_j$, then the object enters the membrane labeled with j, $j \neq i$, assumed it is placed immediately inside the region where the rule is applied (otherwise the rule cannot be applied).

When considering P systems as computing devices, a computation is obtained starting from an initial configuration (described by a fixed membrane structure containing a certain number of objects and rules) and letting the system evolve. A universal clock is assumed to exist: at each step, all rules in all regions are simultaneously applied to all objects which can be the subject of an evolution rule. That is, we say that rules are applied in a maximal parallel manner, as well membranes evolve simultaneously. If no further rule can be applied, the computation halts and its result is read in a prescribed way.

Recently, P systems have been applied in various research areas, ranging from biology to linguistics and to computer science (see, e.g. [9]). For a comprehensive overview of basic P systems, of other classes lately introduced and their application in computer science and biology, we refer the interested reader to [26], and to the P Systems web page (http://ppage.psystems.eu).

3.2 DPP

Dynamical probabilistic P systems (DPP) were introduced in [29]: they are membrane systems where probabilities are associated with the rules, and such values vary during the evolution of the system, according to a prescribed strategy. The method for evaluating probabilities and the way the system works are here briefly explained.

In what follows, we are not interested in the computation of DPP, but in its *evolution*, since we are interested in the application of DPP as a discrete and stochastic tool for the analysis of dynamical systems. For more details about DPP and examples of simulated systems, we refer also to [2, 3, 27, 28].

Formally, a *dynamical probabilistic P system* of degree n is a construct $\Pi = (V, O, \mu, M_0, \dots, M_{n-1}, R_0, \dots, R_{n-1}, E, I)$ where:

- V is the alphabet of the system, $O \subseteq V$ is the set of analyzed symbols.
- μ is a membrane structure consisting of n membranes labeled with the numbers $0, \dots, n-1$. The skin membrane is labeled with 0.
- M_i, $i = 0, \dots, n-1$, is the initial multi-set over V inside membrane i.
- R_i, $i = 0, \dots, n-1$, is a finite set of evolution rules associated with membrane i. An evolution rule is of the form $r : u \to v$, where u is a multi-set over V, v is a string over $V \times (\{here, out\} \cup \{in_j \mid 1 \leq j \leq n-1\})$. A constant $k \in \mathbb{R}^+$ is also associated with the rule.
- $E = \{V_E, M_E, R_E\}$ is called the environment, and consists of an alphabet $V_E \subseteq V$, a feeding multi-set M_E over V_E, and a finite set of feeding rules R_E of the type $r : u \to (v, in_0)$, for u, v multi-sets over V_E.
- $I \subseteq \{0, \dots, n-1\} \cup \{\infty\}$ is the set of labels of the analyzed regions (the label ∞ corresponds to the environment).

The alphabet O and the set I specify which symbols and regions are of peculiar importance in Π, namely those elements whose evolution will be actually considered during the analysis and the simulation of the system. On the other hand, we can also identify the *set of parameters* $\mathcal{P}$ of Π, which consists of: (1) the multiplicities of all symbols appearing in the multi-sets $M_0, \dots, M_{n-1}$ initially present in μ, and of those appearing in the feeding multi-set M_E, and (2) the constants associated to all rules in $R_0, \dots, R_{n-1}$.

The alphabets V, O, V_E, the membrane structure μ, the form of the rules in $R_0, \dots, R_{n-1}, R_E$, and the set I of analyzed regions—which do not belong to the set of parameters of Π—are the components that determine the *main structure* of Π.

A *family* of DPP can then be defined as $\mathcal{F} = \{(\Pi, \mathcal{P}_i) \mid \Pi$ is a DPP and $\mathcal{P}_i$ is the set of parameters of $\Pi, i \geq 1\}$, that is, it is a class of DPPs where all members have the same main structure, but the parameters can change from member to member. We assume that, given any two elements $(\Pi, \mathcal{P}_1), (\Pi, \mathcal{P}_2) \in \mathcal{F}$, it holds $\mathcal{P}_1 \neq \mathcal{P}_2$ for the choice of at least one value in $\mathcal{P}_1$ and $\mathcal{P}_2$.

The definition of a family of DPP is useful if one wants to analyze the same DPP with different settings of initial conditions, such as different initial multi-sets and/or different rule constants (for instance, when not all of them are previously known and one needs to reproduce a given behavior) and/or different feeding multi-sets. In other words, the family $\mathcal{F}$ describes a general model of the biological or chemical system of interest and, for any fixed choice of the parameters, we can investigate the evolution of the corresponding specified DPP.

We describe now how *stochasticity* plays its role in DPP. To each rule in the set R_i, for all $i = 0, \dots, n-1$, we associate a *probability* for the rule application, which is computed considering the rule constant and the current multi-set occurring in membrane i. Namely, let $V = \{a_1, \dots, a_l\}$, M_i be the multi-set inside membrane i and $r : u \to v$ a rule in R_i with constant k. Let also be $u = a_1^{\alpha_1} \dots a_s^{\alpha_s}$, $alph(u) = \{a_1, \dots, a_s\}$ (the set of symbols from V appearing in u) and $H = \{1, \dots, s\}$. To obtain the actual normalized probability p_i of applying r with respect to all other rules that are applicable in membrane i at the same evolution step, we need to evaluate the (non-normalized) pseudo-probability $\tilde{p}_i(r)$ of r. The pseudo-probability depends on the constant associated with r and on the left-hand side of r, and is defined as

$$\tilde{p}_i(r) = \begin{cases} 0 & \text{if } M_i(a_h) < \alpha_h \text{ for some } h \in H, \\ k \cdot \prod_{h\in H} \frac{M_i(a_h)!}{\alpha_h!(M_i(a_h)-\alpha_h)!} & \text{if } M_i(a_h) \geq \alpha_h \text{ for all } h \in H. \end{cases} \tag{1}$$

In other words, whenever the current multi-set inside membrane i contains a sufficient number of each symbol appearing in the left-hand side of the rule (second case in (1)), then $\tilde{p}_i(r)$ is dynamically computed according to the current multi-set inside membrane i. For each symbol a_h appearing in u, we choose α_h copies of a_h among all its $M_i(a_h)$ copies currently available in the membrane, that is, we consider all possible distinct combinations of the symbols appearing in $alph(u)$. Thus, $\tilde{p}_i(r)$ corresponds to the probability, computed during the current evolution step, of having an interaction among the objects (placed on the left-hand side of the rule), which are considered indistinguishable.

Once the pseudo-probability of all rules inside a membrane have been given, we evaluate the normalized probabilities for each rule: this value gives the actual probability to execute that rule inside the membrane. The normalized probability for any rule $r_j \in R_i$ is

$$p_i(r_j) = \frac{\tilde{p}_i(r_j)}{\sum_{j=1}^{m} \tilde{p}_i(r_j)}. \tag{2}$$

A DPP works as follows. An initial configuration of Π is fixed depending on the choice of the values in $\mathcal{P}$: it consists of the multi-sets initially present inside the membrane structure, the chosen rule constants and the feeding multi-set (this is given as an input to the skin membrane from the environment at each step of the evolution by applying the feeding rules).[1] Then we assume that the system evolves according to a universal clock, that is, all membranes and the application of all rules are synchronized. At each step of the evolution, all applicable rules are simultaneously applied and all occurrences of the left-hand sides of the rules are consumed. The applied rules are chosen according to the probability values dynamically assigned to them; the rules with the highest normalized probability value will be more frequently tossed. If some rules compete for objects and have the same probability values, then objects are nondeterministically assigned to them. Note that, by the assignment of a dynamic probability to each evolution rule, DPP mitigate the maximal parallelism of membrane systems, a feature that makes them more feasible for the study of biological systems.[2]

[1] Different strategies in the feeding process have been defined in [2, 3], where the role of the environment was replaced with feeding rules of a stochastic type, directly defined and applied inside the internal membranes.

[2] In [3], we also introduced the use of "mute rules" (of the form $u \to (u, here)$) which can be used as a trick to maintain the maximal parallelism at the level of rule application, but not at the level of object consumption.

The dynamics generated by DPP is *qualitative*, that is, they do not assign a time increment to the simulation steps. The need for an accurate *quantitative* tool led to the definition of τ-DPP [7], designed to share a common time increment among all the membranes, which allows to generate an accurate distribution of rules in each compartment.

3.3 τ-DPP

τ-DPP is a stochastic simulation approach introduced in [7] to the aim of extending the single-volume algorithm of tau-leaping [6]. τ-DPP is able to handle multivolume systems where the distinct volumes are arranged according to a specified hierarchy, and with the additional assumption that the topological structure and the volume dimensions do not change during the system evolution. As already seen in Sect. 3.1, this condition is well satisfied by the membrane structure of a P system, which is also the underlying spatial arrangement used within τ-DPP. Indeed, τ-DPP presents a close correspondence to DPP, but it exploits a (modified) version of the tau-leaping procedure to simulate the evolution dynamics of each volume, as well as the one of the whole system.

The correct behavior of the whole system is achieved by letting all volumes evolve in parallel, and by using the following strategy for the choice of time increments. At each iteration of the algorithm execution, we consider the current state of each volume (determined by the current number of objects), and we calculate a time increment independently in each volume (according to the standard tau-leaping algorithm). Then the smallest time increment is selected and used to evaluate the next-step evolution of the entire system. Since all volumes *locally* evolve according to the same time increment, τ-DPP is able to correctly work out the *global* dynamics of the system. Moreover, by adopting this procedure, the simulated evolutions of all volumes get naturally *synchronized* at the end of each iterative step. The synchronization is also necessary—and exploited together with a parallel update of all volumes—to manage the communication of objects among volumes, whenever prescribed by specific (communication) rules.

Besides the length of the local and global time increments, we need to check which kind of evolution will be performed inside each volume, during the current iteration, by looking only at the volume internal state. The types of evolutionary step are those defined in the tau-leaping procedure and described in Sect. 2.2: a volume can evolve executing either (1) a SSA-like step, or (2) non-critical reactions only, or (3) a set of non-critical reactions plus one critical reaction. When the global time increment has been chosen, it is possible to extract the rules that will be applied inside each volume at the end of the current iterative step. A detailed description of τ-DPP procedure is presented later on.

Internal and Communication Rules

In the description of a biological or chemical system by means of τ-DPP, each volume $V_1, \ldots, V_n$ can contain two different kinds of rules, termed *internal* and *communication* rules. An internal rule describes the modification or evolution of the objects inside the single volume where it is applied, while a communication rule sends the objects from the volume where it is applied to an adjacent volume (possibly modifying the form of these objects during the communication step). In both cases, a stochastic constant has to be associated with the rule, which is needed to compute the probability of applying that rule, together with the form of the rule itself (algebraically, the stochastic constant is related to the deterministic rate constant of biochemical systems).

Internal rules have the general form $\alpha_1 S_1 + \alpha_2 S_2 + \cdots + \alpha_k S_k \rightarrow \beta_1 S_1 + \beta_2 S_2 + \cdots + \beta_k S_k$, where $S_1, \ldots, S_k \in \mathcal{S}$ are distinct object types (e.g. molecular species), and $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k \in \mathbb{N}$ (e.g. stoichiometric coefficients). The objects appearing in the left-hand side of the rule are called *reagents*, while the object on the right-hand side are called *products*. Note that usually we will consider the case where (at most) two objects appear in the reagents group. The rational behind this is that we require biochemical reactions to be (at most) of the second-order, since the simultaneous collision and chemical interaction of more than two molecules at a time, has a probability to occur close to zero in real biochemical systems.

When dealing with communication rules inside a volume, besides defining the sets of reagents and products, it is necessary to specify the target volume where the products of this rule will be sent. Formally, a communication rule has the form[3] $\alpha_1 S_1 + \alpha_2 S_2 + \cdots + \alpha_k S_k \rightarrow (\beta_1 S_1 + \beta_2 S_2 + \cdots + \beta_k S_k, tar)$, where $S_1, \ldots, S_k \in \mathcal{S}$ are distinct object types, $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k \in \mathbb{N}$, and *tar* can be equal to:

- *out*: this means that the products of the rule are "sent outside" the source volume, where the rule is applied, to the adjacent outer volume.
- in_{label}: this means that the products of the rule are "sent inside" the volume with the label specified in the target. This kind of rules are only allowed if the target volume is placed inside the source membrane, and the two volumes are adjacent (that is, there exists no other volume placed between the source and the target volume).
- *in*: this means that the products of the rule are non-deterministically sent to any of the volumes placed inside the source membrane. This kind of rule can be used instead of a set of rules with specific targets in_{label} (one rule for each inner volume).

Communication rules are considered special rules for what concerns the time increment (τ) selection procedure, applied in the first stage of the τ-DPP algorithm. For internal rules, this procedure is exactly the same of the tau-leaping algorithm [6], and exploits the species variation to compute the length of the step. Namely, in order to correctly evaluate the simulation time increment and to describe the behavior of the system with a good approximation, we need to choose the "largest" value of τ that also satisfies the leap condition [6]. This condition informally says that the change in the system state, during the chosen time increment τ, must be

[3] The condition that at most two objects appear as reagents is usually required also for communication rules.

"small enough" so that the variation of the propensity functions of the rules will be slight. The leap condition is used to bound—by means of an error control parameter [6]—the computed τ value because, for *arbitrary* values of the time increment, the computation of the *exact* distribution of the rules to apply is as hard to solve as the CME. On the other hand, when computing the τ value under the leap condition (also considering the changes in the species according to both the left-hand and right-hand sides of the rules), it is possible to find a good approximation of the rules distribution, by means of Poisson random variables related to τ and to the propensity functions of the rules.

As for as internal rules are concerned, this procedure can be applied without restrictions, since they involve a variation only in the volume where they are applied. On the contrary, the variation due to communication rules implies a change in the quantities of objects inside *two* different volumes: the reagents inside the source volume, and the products sent to the destination volume. To correctly estimate the value of τ when dealing with communication rules, instead of looking at the variation of both reagents and products (as it is done for internal rules), we only consider the variation of the objects inside the source volume (that is, we only look at the left-hand side of the communication rule). Indeed, the value of τ is independent from any objects that has to be communicated, since these products will be received in the target volume only at the end of the iteration step (see below). For this reason, for the τ selection procedure, the right-hand side of a communication rule is neither considered in the source, nor in the target volume. Obviously, the communicated objects will contribute to the update of the system state, which takes place at the end of the iteration step, and will be therefore considered to determine the state of the target volume for the next iteration step.

The τ-DPP Algorithm

The τ-DPP algorithm proceeds by executing iterative steps to simulate the evolution of the entire system. Each step consists of several substeps that are executed *independently* and *in parallel* for each single volume V_i, $i = 1, \dots, n$, of the system. In the following description, the algorithm execution naturally proceeds according to the order of instructions, when not otherwise specified by means of "go to" commands.

Step 1. Initialization: load the description of volume V_i, which consists of the initial quantities of all object types, the set of internal and communication rules and their respective stochastic constants, the set of targets for each communication rule.

Step 2. Compute the propensity function a_μ of each rule R_μ, $\mu = 1, \dots, m$, and evaluate the sum of all the propensity functions in V_i, $a_0 = \sum_{\mu=1}^{m} a_\mu$. If $a_0 = 0$, then go to *step 3*, otherwise go to *step 5*.

Step 3. Set τ_i, the length of the step increment in volume V_i, to ∞.

Step 4. Wait for the communication of the smallest time increment $\tau_{\min} = \min\{\tau_1, \dots, \tau_n\}$ among those generated independently inside all volumes $V_1, \dots, V_n$, during the current iteration, then go to *step 13*.

Step 5. Generate the step size τ_i according to the internal state, and select the way to proceed in the current iteration (i.e. SSA-like evolution, or tau-leaping evolution with non-critical reactions only, or tau-leaping evolution with non-critical reactions and one critical reaction), using the selection procedure defined in [6].

Step 6. Wait for the communication of the smallest time increment $\tau_{\min} = \min\{\tau_1, \ldots, \tau_n\}$ among those generated independently inside all volumes, during the current iteration. Then

- if the evolution is SSA-like and the value $\tau_i = \tau_{\text{SSA}}$ generated inside the volume is greater than $\tau_{\min}$, then go to *step 7*;
- if the evolution is SSA-like and the value $\tau_i = \tau_{\text{SSA}}$ is equal to $\tau_{\min}$, then go to *step 10*;
- if the evolution is tau-leaping with non-critical reactions plus one critical reaction, and the value $\tau_i = \tau_{\text{nc1c}}$ is equal to $\tau_{\min}$, then go to *step 11*;
- if the evolution is tau-leaping with non-critical reactions plus one critical reaction, and the value $\tau_i = \tau_{\text{nc1c}}$ is greater than $\tau_{\min}$, then go to *step 12*;
- if the evolution is tau-leaping with non-critical reactions only ($\tau_i = \tau_{\text{nc}}$), then go to *step 12*.

Step 7. Compute $\tau_{\text{SSA}} = \tau_{\text{SSA}} - \tau_{\min}$.

Step 8. Wait for possible communication of objects from other volumes, by means of communication rules. If some object is received, then go back to *step 2*, otherwise go to *step 9*.

Step 9. Set $\tau_i = \tau_{\text{SSA}}$ for the next iteration, then go back to *step 6*.

Step 10. Using the SSA strategy [16], extract the rule that will be applied in the current iteration, then go to *step 13*.

Step 11. Extract the critical rule that will be applied in the current iteration.

Step 12. Extract the set of non-critical rules that will be applied in the current iteration.

Step 13. Update the internal state by applying the extracted rules (both internal and communication) to modify the current number of objects, and then check for objects (possibly) received from the other volumes. Then go back to *step 2*.

The initialization phase of the algorithm is executed in parallel inside every single volume of the system, where distinct rules and amounts of objects may appear. The computation of the propensity functions a_μ of the rules R_μ is performed exploiting the expression presented in [16]. In step 2, we check whether the sum a_0 of all the propensity functions of the volume is equal to zero. If so, then no rule can be executed inside this volume; in this case, we set $\tau_i = \infty$ and let the volume wait for the current $\tau_{\min}$ value, chosen among the time increments computed inside the other volumes. This step is necessary for the synchronization between this volume and the other ones where, possibly, some rules will be applied. So doing, during the update at the final step of the algorithm, it will also be possible to check whether the volume is the target of some communication rules applied in other volumes (that is, whether it will receive new objects), and hence properly update its internal state.

On the other hand, if $a_0 > 0$, the value of τ_i is computed inside the volume considering only its internal state (this is done exploiting the first part of the tau-leaping procedure presented in [6]). The τ_i computation also implies the selection of the kind of evolution for the current iteration inside each volume V_i, independently from the kind of evolution selected in the other volumes.

Once every volume of the system has computed its τ_i value, the smallest one is then selected and used to generate the evolution of the *whole* system (step 6 of the algorithm) during the current iteration. This means that each volume will not evolve according to the internally computed τ_i, but according to the common value $\tau_{\min}$. The rational behind this is that, we let all volumes proceeds along a common timeline, therefore, avoiding paradoxical situations where one volume will execute rules that take place in the future or in the past time of another volume.

When every volume has received the common value $\tau_{\min}$, according to the evolution strategy selected at step 5 of the algorithm, it extracts the rules that will be applied in the final stage of the iteration. If the evolution of the volume is governed by the SSA strategy, we have to check if $\tau_{\min}$ is equal to τ_i (here called τ_{SSA}). If so, it means that the chosen $\tau_{\min}$ value was actually generated inside this volume, and that the current iteration is "long enough" to allow the application of *one* rule inside this volume. On the other hand, if $\tau_{\min} < \tau_{\mathrm{SSA}}$, then it is not possible to apply any rule inside this volume. For this reason, we only have to update the value of τ_i (step 7 of the algorithm).

Afterward, the volume verifies for possible incoming objects: if something is received from other volumes, then the internal state of the volume is changed, and in the next iteration the volume will compute new values of the propensity functions and a new τ_i value. On the contrary, if the volume does not receive anything from its adjacent volumes, then its internal state remains unchanged. For this reason, the τ_i value used in the next iteration will be the current τ_{SSA} value (updated during the step 7 of the algorithm). Notice that the value $\tau_i = \tau_{\mathrm{SSA}} - \tau_{\min}$ is used again in the next iteration because in the case τ_i is the smallest time increment of the whole system, then the volume will be able to complete the execution of one rule.

The situation is easier for the other kinds of evolution. If the volume is executing a tau-leaping step with the application of a set of non-critical reactions and one critical reaction, we have to check if $\tau_i = \tau_{\min}$. If this is true, besides the execution of the non-critical reactions extracted from the Poissonian distributions (we refer to [6] for details), it is possible to execute one critical reaction. Indeed, the value τ_i computed inside the volume corresponds to the time needed to execute such critical reaction (this is the reason why its execution is allowed). On the contrary, if $\tau_i > \tau_{\min}$, the volume will execute non-critical reactions only.

Finally, if the volume is evolving according to the tau-leaping procedure with the execution of non-critical reactions only, we use $\tau_{\min}$ for the sampling of the number of rules that will be executed (according to the strategy described in [6]). The last step of the algorithm corresponds to the system update. Every volume executes its own rules during this step, both internal and communication. The internal states of all volumes are hence updated in parallel and, thanks to the choice of the common time increment $\tau_{\min}$, also in a synchronized fashion.

4 Application of τ-DPP to Coupled Genetic Oscillators

The control of gene expression in living cells can occur through distinct mechanisms, e.g. positive or negative regulation, which enhance (or inhibit, respectively) the binding between the RNA polymerase and the promoter site of a gene. One goal in gene regulation analysis is to understand how functional oscillations (e.g. in the circadian clock) can emerge in a complex system as the macroscopic effect of the interactions and the coupling of basic and microscopic elements.

In particular, the capability to construct and control *synthetic* gene circuits in laboratory experiments, allows to investigate the issue of gene regulation in simplified systems. An example of such systems is the *repressilator*, a synthetic oscillator implemented in *Escherichia coli* cells by means of specifically constructed plasmids [11]. It consists of a network of three genes, *lacI* (from *E. coli*), *tetR* (from the tetracycline-resistant transposon Tn10), and *cI* (from λ phage), whose codified proteins act as repressors of each other's gene, in a cyclic way. Namely, the product of *lacI* inhibits the transcription of *tetR*, the product of *tetR* inhibits the transcription of *cI*, whose product in turn inhibits the transcription of *lacI*, thus closing the repression cycle. To detect and readout the network behavior, a green fluorescent protein—whose synthesis is periodically triggered by the *tetR* product—has been used as the reporter part of this system. Observations on a growing *E. coli* culture evidenced the emergence of spontaneous oscillations in individual cells, as well as the effect of noise through the variability between different cells, probably due to the stochastic fluctuations of the network components [11].

Bacteria are able to synthesize a diffusive molecule, called *autoinducer*, which is used to perform intercellular signaling. This communication mechanism, termed *quorum sensing* [34], allows bacterial cells to sense whether a critical cell density has been reached in their surrounding, thereby switching on whole-population behaviors through the synchronization of all individuals.

Gram-positive and Gram-negative bacteria synthesize different signaling molecules. In general, the first use oligopeptides as autoinducers, while the latter (e.g. *E. coli*) use acyl-homoserine lactone (acyl-HSL) molecules. The quorum sensing circuits in Gram-negative bacteria usually contain two main families of proteins, which have been found to be homologous to two well-characterized protein in the bioluminescent bacterium *V. fischeri*: the LuxI and LuxR families. LuxI-like proteins are needed for the synthesis of the autoinducers (they acts as acyl-HSL synthases when provided with an amino donor and an appropriate acyl donor [24]), while LuxR-like proteins form complexes with the autoinducer, which then regulate the transcription of target genes [22].

In [14], a synthetic multicellular clock has been investigated, by coupling the repressilator system with bacterial quorum sensing. This intercellular communication mechanism is able to lead the *local* genetic oscillators, within a noisy and non-identical population, to *global* oscillatory rhythms. In particular, it was shown that individual repressilator systems can self-synchronize, even when their periods are broadly distributed [14].

In this section, we propose a multi-volume model based on τ-DPP that considers a repressilator-like system (RL) coupled with a quorum sensing-like circuit (QSL).

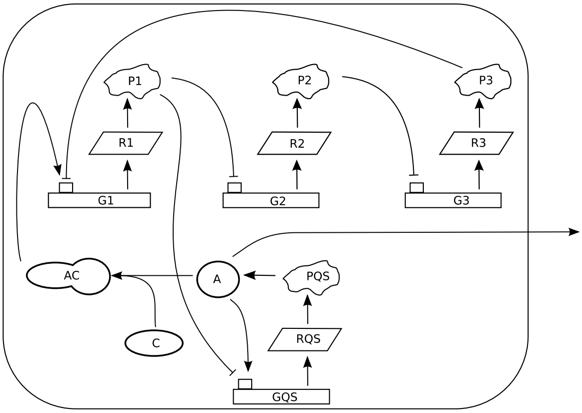


Fig. 1 The (single volume) network of the genetic oscillator system coupled with quorum sensing mechanism

Namely, the RL consists of three genes (G_1, G_2, G_3) which cyclically inhibit each other's expression, while the QSL consists of a genetic component that triggers the production of the auto-inducer (through the synthesis of the intermediate auto-inducer-synthases), plus a "control" molecule that mimics the role of the protein in the LuxR family. The two systems are interlaced through one *positive feedback* and one *negative feedforward* loop: the first operates on the promotion of gene G_1 in the RL by means of the QSL auto-inducer molecule, while the second operates on the inhibition of the QSL gene by means of G_1 product (see Fig. 1).

4.1 A Multi-volume Model for Coupled Genetic Oscillators

The multi-volume model for RL + QSL system consists of n volumes, each one corresponding to a cell, a set of 34 rules defined within each cell (Table 1) and 1 communication rule defined in the environment (Table 2), and an alphabet of 21 molecular species. Among these, we can recognize 14 "individual" species, and 7 "complex" species.

The set $\mathcal{S}$ of individual species consists of G_i, R_i, P_i, which denote, respectively, the ith gene, mRNA and protein in the RL, for $i = 1, 2, 3$; G_{QS}, R_{QS}, P_{QS} which denote, respectively, the gene, mRNA and protein in the QLS; A, the auto-inducer molecule; C, the control molecule used to trigger on, along with A, the over-expression of gene G_1. The set of complex species is denoted by $\mathcal{C}$, and consists of the (bound) species $G_1 \cdot P_3, G_2 \cdot P_1, G_3 \cdot P_2, A \cdot C, A \cdot C \cdot G_1, A \cdot G_{QS}, G_{QS} \cdot P_1$. Each complex species is formed when two individual species interact and get bound. Formally, this process is represented by means of rules of the form $X + Y \to X \cdot Y$, where $X, Y \in \mathcal{S}$. We also consider the possibility to have complex species consisting of more than two individual species; in this case the complex is formed in two sequential steps, that is, $X + Y \to X \cdot Y$, followed by $X \cdot Y + Z \to X \cdot Y \cdot Z$, where $X, Y, Z \in \mathcal{S}$.

The rules in each cell work as follows. Rules r_1 and r_2 describe, respectively, the transcription of gene G_1 into one molecule of mRNA, R_1, and its translation

Table 1 Reactions inside the single cell

Reaction	Reagents → products	Constant
r_1	$G_1 \rightarrow G_1 + R_1$	$c_1 = 1 \times 10^{-2}$
r_2	$R_1 \rightarrow R_1 + P_1$	$c_2 = 1 \times 10^{-1}$
r_3	$R_1 \rightarrow \lambda$	$c_3 = 1 \times 10^{-4}$
r_4	$P_1 \rightarrow \lambda$	$c_4 = 1 \times 10^{-2}$
r_5	$G_2 \rightarrow G_2 + R_2$	$c_5 = 1 \times 10^{-2}$
r_6	$R_2 \rightarrow R_2 + P_2$	$c_6 = 1 \times 10^{-1}$
r_7	$R_2 \rightarrow \lambda$	$c_7 = 1 \times 10^{-4}$
r_8	$P_2 \rightarrow \lambda$	$c_8 = 1 \times 10^{-2}$
r_9	$G_3 \rightarrow G_3 + R_3$	$c_9 = 1 \times 10^{-2}$
r_{10}	$R_3 \rightarrow R_3 + P_3$	$c_{10} = 1 \times 10^{-1}$
r_{11}	$R_3 \rightarrow \lambda$	$c_{11} = 1 \times 10^{-4}$
r_{12}	$P_3 \rightarrow \lambda$	$c_{12} = 1 \times 10^{-2}$
r_{13}	$G_1 + P_3 \rightarrow G_1 \cdot P_3$	$c_{13} = 1 \times 10^{-1}$
r_{14}	$G_1 \cdot P_3 \rightarrow G_1 + P_3$	$c_{14} = 1 \times 10^{-3}$
r_{15}	$G_2 + P_1 \rightarrow G_2 \cdot P_1$	$c_{15} = 1 \times 10^{-1}$
r_{16}	$G_2 \cdot P_1 \rightarrow G_2 + P_1$	$c_{16} = 1 \times 10^{-3}$
r_{17}	$G_3 + P_2 \rightarrow G_3 \cdot P_2$	$c_{17} = 1 \times 10^{-1}$
r_{18}	$G_3 \cdot P_2 \rightarrow G_3 + P_2$	$c_{18} = 1 \times 10^{-3}$
r_{19}	$G_{QS} \rightarrow G_{QS} + R_{QS}$	$c_{19} = 1 \times 10^{-2}$
r_{20}	$R_{QS} \rightarrow R_{QS} + P_{QS}$	$c_{20} = 1 \times 10^{-1}$
r_{21}	$R_{QS} \rightarrow \lambda$	$c_{21} = 1 \times 10^{-4}$
r_{22}	$P_{QS} \rightarrow \lambda$	$c_{22} = 1 \times 10^{-2}$
r_{23}	$P_{QS} \rightarrow P_{QS} + A$	$c_{23} = 5 \times 10^{-3}$
r_{24}	$G_{QS} + A \rightarrow G_{QS} \cdot A$	$c_{24} = 1 \times 10^{-7}$
r_{25}	$G_{QS} \cdot A \rightarrow G_{QS} + A$	$c_{25} = 1 \times 10^{-3}$
r_{26}	$G_{QS} \cdot A \rightarrow G_{QS} \cdot A + R_{QS}$	$c_{26} = 3 \times 10^{-2}$
r_{27}	$G_{QS} + P_1 \rightarrow G_{QS} \cdot P_1$	$c_{27} = 1 \times 10^{-2}$
r_{28}	$G_{QS} \cdot P_1 \rightarrow G_{QS} + P_1$	$c_{28} = 1 \times 10^{-3}$
r_{29}	$A + C \rightarrow A \cdot C$	$c_{29} = 1 \times 10^{-3}$
r_{30}	$A \cdot C \rightarrow A + C$	$c_{30} = 1 \times 10^{-2}$
r_{31}	$G_1 + A \cdot C \rightarrow G_1 \cdot A \cdot C$	$c_{31} = 1 \times 10^{-6}$
r_{32}	$G_1 \cdot A \cdot C \rightarrow G_1 + A \cdot C$	$c_{32} = 1 \times 10^{-3}$
r_{33}	$G_1 \cdot A \cdot C \rightarrow G_1 \cdot A \cdot C + R_1$	$c_{33} = 5 \times 10^{-2}$
r_{34}	$A \rightarrow (A, out)$	$c_{34} = 1$

Table 2 Reaction in the environment

Reaction	Reagents → products	Constant
r_E	$A \rightarrow (A, in)$	$c_E = 1$

into one copy of protein P_1. Rules r_3, r_4 describe the degradation of R_1 and P_1, respectively. The same holds for the sets of rules $r_5, \ldots, r_8$ and $r_9, \ldots, r_{12}$, defined for the other two genetic elements in the RL.[4]

Rules r_{13}, r_{15}, r_{17} describe the cyclic repression relation between the 3 genes in the RL: namely, the protein synthesized by gene G_1, G_2, G_3, acts as repressor for the expression of gene G_2, G_3, G_1, respectively. When the protein binds to the gene (e.g., rule r_{13} forms the complex $G_1 \cdot P_3$), it blocks gene expression by avoiding the application of the respective transcription rule (e.g., rule r_1). The gene repression ends whenever the repressor is released from the complex and the gene returns in an unbound form, that is, by the application of the inverse rules r_{14}, r_{16}, r_{18}.

The other rules consider the coupling between the RL and the QSL, which allows cells to communicate each other and possibly synchronize their behavior. Rules $r_{19}, \ldots, r_{22}$ describe the transcription and translation of the quorum sensing gene G_{QS}, and the degradation of the respective synthesized mRNA and protein, R_{QS} and P_{QS}. Rule r_{23} describes the synthesis of the auto-inducer molecule A, by means of the quorum sensing protein P_{QS}.[5] When the auto-inducer is present in the membrane, it can undergo three different processes. First, it can bind to the quorum sensing gene (rule r_{24}, and its inverse r_{25}) and promote its expression (rule r_{26}); note that the expression of G_{QS} can also be inhibited, if protein P_1 binds to it (rule r_{27}, and its inverse r_{28}). Second, the auto-inducer can form a complex with the control molecules C (rule r_{29}, and its inverse r_{30}). In this case, the complex $A \cdot C$ can bind to gene G_1 (rules r_{31} and r_{32}) and enhance its expression (rule r_{33}). Third, the auto-inducer can exit the cell (rule r_{34}) and diffuse into the environment, from where it can enter again any cell, by the application of the non-deterministic environmental rule r_E.

4.2 Results of Simulations

In the following, we report the results of the simulations performed with τ-DPP using, when not otherwise specified, the stochastic constant values given in Tables 1 and 2.

We start by presenting in Fig. 2 the oscillatory dynamics of the repressor proteins P_1, P_2, P_3 in the single cell, when the quorum sensing circuit is silenced (that is, only rules $r_1, \ldots, r_{18}$ are active inside the cell). The expected outcome resembles the behavior obtained in [11], where the order of oscillations of the three proteins proceed in a sequential, cyclic manner, dictated by the structure of the genetic repressor systems.

[4] Note that we do not explicitly describe the presence and the role played by transcriptional, translational, and degradation machineries occurring in real cells, tacitly assuming their constant availability.

[5] In bacteria, the synthesis of these molecules require the additional components described in Sect. 4. We assume that these components are always available and we do not explicitly include them in the model.

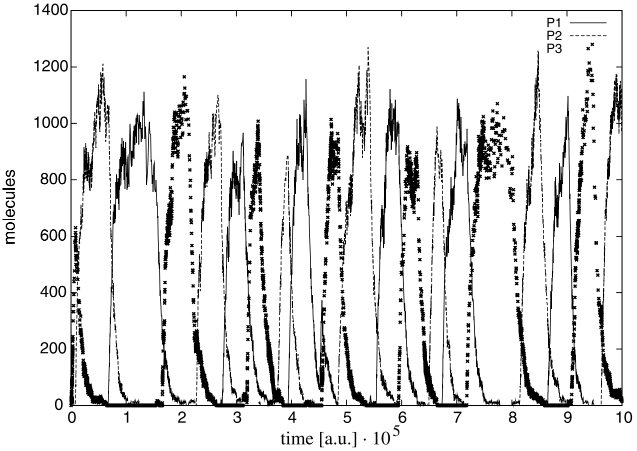


Fig. 2 Stochastic oscillations of the repressor proteins P_1, P_2, P_3 in the single cell

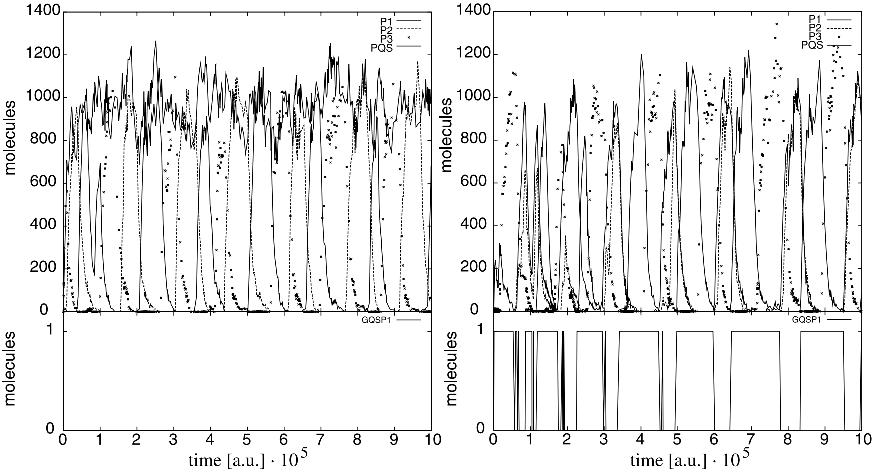


Fig. 3 Dynamics of quorum sensing protein P_{QS} (see text). *Left side*: steady state. *Right side*: controlled oscillations

In Fig. 3, we show the variation of dynamics of the quorum sensing protein, P_{QS}. On the left side, we show the situation where only rules $r_1, \ldots, r_{22}$ are active, hence the RL and the QSL are not interlaced. In this case, P_{QS} reaches a steady state (top part), and the complex $G_{\mathrm{QS}} \cdot P_1$ is never formed (bottom part). On the contrary, if also rules r_{27}, r_{28} are active, that is, the negative feedforward occurs, then P_{QS} undergoes controlled oscillations (right side, top part). The right bottom part indicates how frequently protein P_1 regulates the expression of G_{QS} (the complex $G_{\mathrm{QS}} \cdot P_1$ is formed).

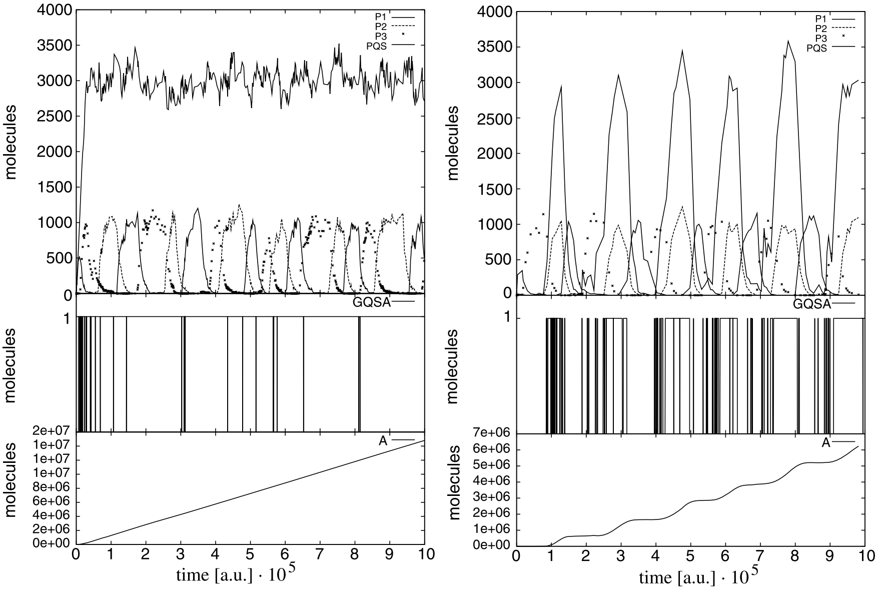


Fig. 4 Dynamics of quorum sensing protein P_{QS} and auto-inducer (see text). *Left side*: steady state. *Right side*: controlled oscillations

Then we consider the communication between the single cell and the environment, and investigate the oscillatory dynamics within the cell through the activation, from time to time, of different subsets of rules.

In Fig. 4, we show the situation within the single cell when the quorum sensing circuit is activated and interlaced with the genetic oscillator. On the left side, all rules $r_1, \dots, r_{26}, r_{34}, r_E$ are active, but the control of P_1 over G_{QS} is silenced. In this case, P_{QS} reaches again a steady state (top part), and the auto-inducer molecule is constantly accumulated inside the cell (bottom part), thus promoting the expression of G_{QS} though the formation of the complex $G_{\text{QS}} \cdot A$ (middle part). If also rules r_{27}, r_{28} are activated (right side), then P_{QS} starts oscillating (top part), thanks to the positive regulation of the auto-inducer (through more frequent formations of the complex $G_{\text{QS}} \cdot A$, middle part), whose amount is no more constantly increasing now (bottom part).

Finally, in Fig. 5, we show the dynamics within the single cell when all rules (left side), and all rules but the environmental one (right side), are activated. Therefore, in this case, we are also considering the positive regulation of gene G_1 by means of the auto-inducer (controlled, in turn, by C, which is assumed to be initially present in 1000 copies inside the cell). On the left side, we show how the amount of P_1 gets notably increased, together with P_{QS} (top part), since there is a high auto-induction of G_{QS} (middle part). The auto-inducer shows the same behavior of the previous figure. On the right side, where the communication of the auto-inducer from the environment toward the cell is silenced, we see that the auto-inductive regulation of G_{QS} is not triggered (middle part), since few molecules A remain inside the cell

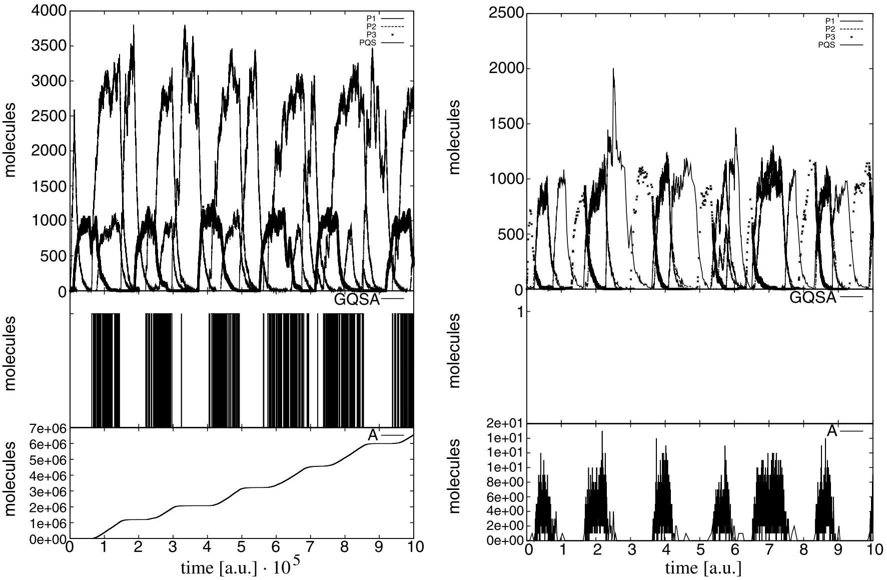


Fig. 5 Genetic oscillator interlaced with quorum sensing circuit inside the single cell. *Left side*: active auto-induction. *Right side*: no auto-induction

(bottom part). Therefore, the dynamics of proteins P_1, P_2, P_3, P_{QS} resembles the normal oscillations of Fig. 3.

In order to better validate the effective application of communication rules in τ-DPP, and to investigate the role of the intercellular signaling mechanism, we consider a simple example where two cells (plus the environment) are coupled together. In particular, we analyze the case where one cell, say C_1, is unable to produce the auto-inducer (that is, rule r_{23} is silenced by setting $c_{23} = 0$), while the other cell (C_2) is fully functional. Note that if C_1 is placed in an empty environment (where neither cells nor auto-inducer molecules are present) under these conditions, then all its internal rules triggered by the auto-inducer (namely, $r_{24}, \ldots, r_{34}$) cannot be switched on. Instead, by placing C_1 together with C_2, we show how the correct application of the communication of the auto-inducer from C_2 to the environment, and from here to C_1, is able to activate all other rules $r_{24}, \ldots, r_{34}$ in C_1. Therefore, the intercellular (quorum sensing like) mechanism is effectively working in a proper way. The results of this simulation are shown in Fig. 6 where, on the left side, we present the oscillations occurring along a common timeline in cells C_1 (top part, where P_{QS} is not as high as P_1) and C_2 (bottom part, where both P_{QS} and P_1 are elicited). On the right side, instead, we show how frequently the gene G_{QS} gets negatively regulated by P_1 and positively regulated by A (note that the value of $G_{\mathrm{QS}} \cdot A$ is multiplied by 1.1 in order to make the graphic more readable).

Finally, it would be interesting to check whether a more complex system, consisting of a population of coupled cells, can show (emergent) synchronization events with respect to the oscillations of the three repressor proteins and the quorum sens-

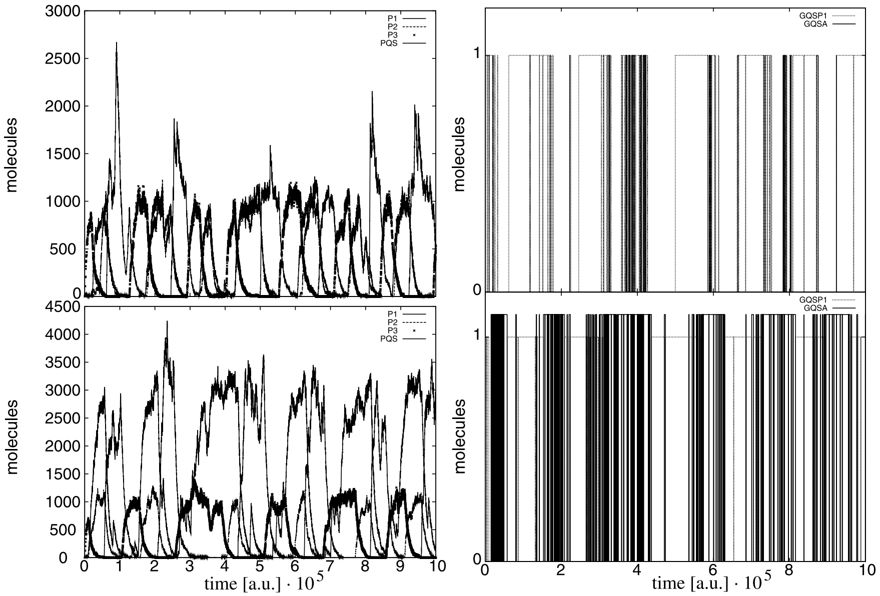


Fig. 6 Comparison of dynamics between two coupled cells. *Left side*: oscillations in C_1 (*top*) and C_2 (*bottom*). *Right side*: gene regulation in C_1 (*top*) and C_2 (*bottom*)

ing protein. Since the modeling and simulation approach we use here is stochastic, the gain of such a synchronized behavior in a noisy system is harder than doing this with a deterministic approach (see, e.g. [14]). We will deal with this issue in a future extension of this work.

5 Discussion

τ-DPP provides the description of the dynamics of systems characterized by a complex structure, composed by *several volumes* organized in a specified hierarchy. Both the dynamics of each single volume and of the system as a whole, can be achieved in an accurate way, by (independently) considering the current state of the volumes and by choosing a common time increment of the system for each iteration. This strategy is implemented exploiting the tau-leaping procedure. On the other hand, it is known that the complexity of the tau-leaping algorithm, though in general performing better than SSA, increases with the number of the species and of the reactions. In addition, in τ-DPP we have to consider the computational cost raised by the number of volumes in the system structure. Another problem of the τ-DPP algorithm, is related to the τ selection procedure: using the smallest τ computed at each iteration means that the volumes with an internal τ greater than $\tau_{\min}$ will execute a larger number of steps to describe their internal behavior. Despite that, this kind of τ selection procedure is needed in order to manage the communication and the synchronization among the compartments at the end of each iteration.

Another issue concerns the stochastic constants associated with communication rules. The application of a communication rule means that an object (or a set of objects) is *instantaneously* sent to the target volume, without considering the "time" that the objects could need to move from one volume to another one. For instance, a communication rule might represent the passage of a solute through a cellular membrane mediated by a transport protein. This biological process can require a conformational change in the transport protein structure, which necessarily prolong the time the solute would take to virtually cross the membrane in a direct way [23]. To this aim a tuning of the stochastic constants of the communication rules would be necessary, in order to better represent the behavior of the objects that move between adjacent volumes. Another solution could consist in defining intermediate volumes in τ-DPP (that is, to consider the biological membrane as a volume in itself), or also to include diffusion processes for the movement of objects.

In addition, the problem of the volume changing in time should also be addressed, especially when dealing with cellular processes spanning the cell cycle period. In fact, in this case the dimension of the volume can increase, and also its internal conditions can be modified, therefore, the requisite condition of fixed volume for the applicability of standard stochastic algorithms fails.

References

1. Arkin A, Ross J, McAdams HH (1998) Genetics 149:1633–1648
2. Besozzi D, Cazzaniga P, Pescini D, Mauri G (2007) Prog Nat Sci 17(4):392–400
3. Besozzi D, Cazzaniga P, Pescini D, Mauri G (2008) Biosystems 91(3):499–514
4. Blake WJ, Kærn M, Cantor CR, Collins JJ (2003) Nature 422:633–637
5. Cao Y, Gillespie DT, Petzold LR (2005) J Chem Phys 123:054104
6. Cao Y, Gillespie DT, Petzold LR (2006) J Chem Phys 124:044109
7. Cazzaniga P, Pescini D, Besozzi D, Mauri G (2006) In: Hoogeboom HJ, Păun G, Rozenberg G, Salomaa A (eds) Lecture notes in computer science, vol 4361. Springer, Berlin, pp 298–313
8. Chatterjee A, Vlachos DG, Katsoulakis MA (2005) J Chem Phys 122:024112
9. Ciobanu G, Păun G, Pérez-Jiménez MJ (eds) (2005) Applications of membrane computing. Springer, Berlin
10. Elf J, Ehrenberg M (2004) Syst Biol 1(2):230–236
11. Elowitz MB, Leibler S (2000) Nature 403:335–338
12. Elowitz MB, Levine AJ, Siggia ED, Swain PS (2002) Science 297:1183–1186
13. Fedoroff N, Fontana W (2002) Science 297:1129–1131
14. Garcia-Ojalvo J, Elowitz MB, Strogatz SH (2004) Proc Natl Acad Sci 101(30):10955–10960
15. Gibson M, Bruck J (2000) J Phys Chem 104:1876–1889
16. Gillespie DT (1977) J Phys Chem 81:2340–2361
17. Gillespie DT, Petzold LR (2001) J Chem Phys 115:1716–1733
18. Gillespie DT, Petzold LR (2003) J Chem Phys 119:8229–8234
19. Marquez-Lago TT, Burrage K (2007) J Chem Phys 127:104101
20. McAdams HH, Arkin A (1997) Proc Natl Acad Sci 94:814–819
21. Meng TC, Somani S, Dhar P (2004) Silico Biol 4:0024
22. Miller MB, Bassler BL (2001) Annu Rev Microbiol 55:165–199
23. Nelson DL, Cox MM (2004) Lehninger principles of biochemistry, 4th edn. Freeman, New York
24. Parsek MR, Val DL, Hanzelka BL, Cronan JE, Greenberg EP (1999) Proc Natl Acad Sci 96:4360–4365

25. Păun G (2000) J Comput Syst Sci 61(1):108–143
26. Păun G (2002) Membrane computing. An introduction. Springer, Berlin
27. Pescini D, Besozzi D, Mauri G (2005) In: Proceedings of 7th international symposium on symbolic and numeric algorithms for scientific computing (SYNASC'05). IEEE Computer Press, Los Alamitos, pp 440–447
28. Pescini D, Besozzi D, Mauri G, Zandron C (2006) Int J Found Comput Sci 17(1):183–204
29. Pescini D, Besozzi D, Zandron C, Mauri G (2006) In: Pierce N, Carbone A (eds) Lecture notes in computer science, vol 3892. Springer, Berlin, pp 236–247
30. Rosenfeld N, Young JW, Alon U, Swain PS, Elowitz MB (2005) Science 307:1962–1965
31. Swain PS, Elowitz MB, Siggia ED (2002) Proc Natl Acad Sci 99(20):12795–12800
32. Tian T, Burrage K (2004) J Chem Phys 121:10356–10364
33. Turner TE, Schnell S, Burrage K (2004) Comput Biol Chem 28:165–178
34. Waters CM, Bassler BL (2005) Annu Rev Cell Dev Biol 21:319–346

Programmability of Chemical Reaction Networks

Matthew Cook, David Soloveichik, Erik Winfree, and Jehoshua Bruck

Abstract Motivated by the intriguing complexity of biochemical circuitry within individual cells we study Stochastic Chemical Reaction Networks (SCRNs), a formal model that considers a set of chemical reactions acting on a finite number of molecules in a well-stirred solution according to standard chemical kinetics equations. SCRNs have been widely used for describing naturally occurring (bio)chemical systems, and with the advent of synthetic biology they become a promising language for the design of artificial biochemical circuits. Our interest here is the computational power of SCRNs and how they relate to more conventional models of computation. We survey known connections and give new connections between SCRNs and Boolean Logic Circuits, Vector Addition Systems, Petri nets, Gate Implementability, Primitive Recursive Functions, Register Machines, Fractran, and Turing Machines. A theme to these investigations is the thin line between decidable and undecidable questions about SCRN behavior.

1 Introduction

Stochastic chemical reaction networks (SCRNs) are among the most fundamental models used in chemistry, biochemistry, and most recently, computational biology. Traditionally, analysis has focused on *mass action* kinetics, where reactions are assumed to involve sufficiently many molecules that the state of the system can be accurately represented by continuous molecular concentrations with the dynamics given by deterministic differential equations. However, analyzing the kinetics of small-scale chemical processes involving a finite number of molecules, such as occurs within cells, requires stochastic dynamics that explicitly track the exact number of each molecular species [1–3]. For example, over 80% of the genes in the *E. coli* chromosome are expressed at fewer than a hundred copies per cell [4], averaging, for example, only 10 molecules of Lac repressor [5]. Further, observations and computer simulations have shown that stochastic effects resulting from these small numbers may be physiologically significant [6–8].

M. Cook (✉)
Institute of Neuroinformatics, UZH, ETH Zürich, Zürich, Switzerland
e-mail: cook@ini.phys.ethz.ch

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_27, © Springer-Verlag Berlin Heidelberg 2009

In this paper, we examine the computational power of Stochastic Chemical Reaction Networks. Stochastic Chemical Reaction Networks are closely related to computational models such as Petri nets [9], Vector Addition Systems (VASs) [10], Fractran [11, 12], and Register Machines (sometimes called Counter Machines) [13], and for many of these systems we can also consider stochastic or nondeterministic variants. Our initial route into this subject came through the analysis of a seemingly quite unrelated question: What digital logic circuits are constructible with a given set of gate types when it is not possible to copy values (as is true, for example, in quantum circuits)? It turns out that this gate implementability question, as we will discuss in Sect. 4.1, is very closely related to the question of what states can be reached by a Stochastic Chemical Reaction Network.

Given the importance of stochastic behavior in Chemical Reaction Networks, it is particularly interesting that whereas most questions of *possibility* concerning the behavior of these models are decidable [10], the corresponding questions of *probability* are undecidable [14, 15]. This result derives from showing that Stochastic Chemical Reaction Networks can simulate Register Machines [16] efficiently [17] within a known error bound that is independent of the unknown number of steps prior to halting [14]. This result—that when answers must be guaranteed to be correct, computational power is limited, but when an arbitrarily small error probability can be tolerated, the computational power is dramatically increased—can be immediately applied to the other models (Petri nets and VASs) when they are endowed with appropriate stochastic rates. This result is surprising, in light of the relatively ineffective role the addition of probability plays in the widely held belief that $BPP = P$.

Several further results extend and refine this distinction.

- When endowed with special *fast* reactions guaranteed to occur before any *slow* reaction, Stochastic Chemical Reaction Networks become Turing universal, and thus can compute any computable function without error.
- Stochastic Chemical Reaction Networks with reaction rates governed by standard chemical kinetics can compute any computable function with probability of error less than ϵ for any $\epsilon > 0$, but for $\epsilon = 0$ universal computation is impossible [10, 14, 17].
- Stochastic Chemical Reaction Networks in which each reaction's probability of occurring depends only on what reactions are possible (but not on the concentrations) are not capable of universal computation with any fixed bounded probability of success.
- Taking the result of the longest possible sequence of reactions as the answer, Stochastic Chemical Reaction Networks are capable of computing exactly the class of primitive recursive functions without error.
- The time and space requirements for Stochastic Chemical Reaction Networks doing computation, compared to a Turing Machine, are a simple polynomial slowdown in time, but an exponential increase in space [14, 17].

This last result, regarding the complexity, is the best that can be expected, due to the unavoidable fact that information must effectively be stored in the bits comprising the number of molecules present of each species. For uniform computations, wherein the same finite set of chemical species and reactions are used to solve any instance of the problem, storing n bits requires the presence of $2^{\Omega(n)}$ molecules. In practice, keeping an exponentially large solution well stirred may take a correspondingly large amount of time, but in any event, due to the space constraint, Stochastic Chemical Reaction Networks will effectively be limited to logspace computations.

The intention of this paper is to review, present, and discuss these results at an intuitive level, with an occasional foray into formal exactitude. Enjoy.

2 Formalization of Chemistry

2.1 Stochastic Chemical Reaction Networks

A Stochastic Chemical Reaction Network is defined as a finite set of d reactions acting on a finite number m of species. Each reaction α is defined as a vector of nonnegative integers specifying the stoichiometry of the reactants, $\mathbf{r}_\alpha = (r_{\alpha,1}, \dots, r_{\alpha,m})$, together with another vector of nonnegative integers specifying the stoichiometry of the products, $\mathbf{p}_\alpha = (p_{\alpha,1}, \dots, p_{\alpha,m})$. The stoichiometry is the nonnegative number of copies of each species required for the reaction to take place, or produced when the reaction does take place. We will use capital letters to refer to various species and we will use standard chemical notation to describe reactions. So, for example, the reaction $A + D \to A + 2E$ consumes 1 molecule of species A and 1 molecule of species D and produces 1 molecule of species A and 2 molecules of species E (see Fig. 1). In this reaction, A acts catalytically because it must be present for the reaction to occur, but its number is unchanged when the reaction does occur.[1]

The state of the network is defined as a vector of nonnegative integers specifying the quantities present of each species, $\mathcal{A} = (q_1, \dots, q_m)$. A reaction is possible in state $\mathcal{A}$ only if there are enough reactants present, that is, $\forall i, q_i \geq r_{\alpha,i}$. When reaction α occurs in state $\mathcal{A}$, the reactant molecules are used up and the products are produced. The new state is $\mathcal{B} = \mathcal{A} * \alpha = (q_1 - r_{\alpha,1} + p_{\alpha,1}, \dots, q_m - r_{\alpha,m} + p_{\alpha,m})$. We write $\mathcal{A} \overset{C}{\to} \mathcal{B}$ if there is some reaction in the Stochastic Chemical Reaction Network C that can change $\mathcal{A}$ to $\mathcal{B}$; we write $\overset{C*}{\to}$ for the reflexive transitive closure of $\overset{C}{\to}$. We write $\Pr[\mathcal{A} \overset{C}{\to} \mathcal{B}]$ to indicate the probability that, given that the state is initially $\mathcal{A}$, the next reaction will transition to the state to $\mathcal{B}$. $\Pr[\mathcal{A} \overset{C*}{\to} \mathcal{B}]$ refers to the probability that at *some* time in the future, the system is in state $\mathcal{B}$.

Every reaction α has an associated rate constant $k_\alpha > 0$. The rate of every reaction α is proportional to the concentrations (number of molecules present) of each reactant, with the constant of proportionality being given by the rate constant k_α.

[1] In chemistry, catalysis can involve a series of reactions or intermediate states. In this paper, however, we will generally use the word catalyst to mean a species which participates in, but is unchanged by, a *single* reaction.

Specifically, given a volume V, for any state $\mathcal{A} = (q_1, \dots, q_m)$, the rate of reaction α in that state is

$$\rho_\alpha(\mathcal{A}) = k_\alpha V \prod_{i=1}^{m} \frac{(q_i)^{\underline{r_{\alpha,i}}}}{V^{r_{\alpha,i}}} \quad \text{where } q^{\underline{r}} \overset{\text{def}}{=} \frac{q!}{(q-r)!} = q(q-1)\cdots(q-r+1). \quad (1)$$

Since the solution is assumed to be well stirred, the time until a particular reaction α occurs in state $\mathcal{A}$ is an exponentially distributed random variable with the rate parameter $\rho_\alpha(\mathcal{A})$; i.e., the dynamics of a Stochastic Chemical Reaction Network is a continuous-time Markov process, defined as follows.

We write $\Pr[\mathcal{A} \overset{C}{\to} \mathcal{B}]$ to indicate the probability that, given that the state is initially $\mathcal{A}$, the next reaction will transition to the state to $\mathcal{B}$. These probabilities are given by

$$\Pr[\mathcal{A} \overset{C}{\to} \mathcal{B}] = \frac{\rho_{\mathcal{A}\to\mathcal{B}}}{\rho^{\text{tot}}_{\mathcal{A}}}$$
$$\text{where } \rho_{\mathcal{A}\to\mathcal{B}} = \sum_{\alpha \text{ s.t. } \mathcal{A}*\alpha=\mathcal{B}} \rho_\alpha(\mathcal{A}) \text{ and } \rho^{\text{tot}}_{\mathcal{A}} = \sum_{\mathcal{B}} \rho_{\mathcal{A}\to\mathcal{B}}. \quad (2)$$

The average time for a step $\mathcal{A} \to \mathcal{B}$ to occur is $1/\rho^{\text{tot}}_{\mathcal{A}}$, and the average time for a sequence of steps is simply the sum of the average times for each step. We write $\Pr[\mathcal{A} \overset{C*}{\to} \mathcal{B}]$ to refer to the probability that at *some* time in the future, the system is in state $\mathcal{B}$.

This model is commonly used for biochemical modeling [1–3]. When using this model as a language for describing real chemical systems, the reasonableness of the underlying assumptions are affirmed (or denied) by the model's accuracy with respect to the real system. However, in the work presented here, we will be using the model as a programming language—we will write down sets of formal chemical reactions that have no known basis in reality, and any network that is formally admitted by the model will be fair game. That is, while Stochastic Chemical Reaction Networks are usually used *descriptively*, we will be using them *prescriptively*: we imagine that if we can specify a network of interest to us, we can then hand it off to a talented synthetic chemist or synthetic biologist who will design molecules that carry out each of the reactions. Therefore, our concern is with what kinds of systems the formal model is capable of describing—because our philosophy is that if it can be described, it can be made. Of course, this might not be true. A similar issue arises in classical models of computation: It is often observed that Turing Machines cannot be built, because it is impossible to build an infinite tape or a machine that is infinitely reliable. Nonetheless, it is enlightening to study them. We believe the formal study of Stochastic Chemical Reaction Networks will be similarly enlightening. But before proceeding, it is worth considering just how unrealistic the model can become when we are given free reign to postulate arbitrary networks.

An immediate concern is that while we will consider SCRNs that produce arbitrarily large numbers of molecules, it is impossible that so many molecules can fit within a predetermined volume. Thus, we recognize that the reaction volume V must change with the total number of molecules present, which in turn will slow down all reactions involving more than one molecule as reactants. Choosing V to scale proportionally with the total number of molecules present (of any form) results in a model appropriate for analysis of reaction times. Note, however, that for any Stochastic Chemical Reaction Network in which every reaction involves exactly the same number of reactants, the transition probabilities $\Pr[\mathcal{A} \xrightarrow{C} \mathcal{B}]$ are *independent* of the volume. For all the positive results discussed in this paper, we were able to design Stochastic Chemical Reaction Networks involving exactly two reactants in every reaction and, therefore, volume needs to be considered only where computation time is treated. A remaining concern—which we cannot satisfactorily address—is that the assumption of a well-stirred reaction may become less tenable for large volumes. (However, this assumption seems intrinsically no less justified than the common assumption that wires in boolean circuits may be arbitrarily long without transmission errors, for example.)

A second immediate concern is that the reactions we consider are of a very general form, including reactions such as $A \to A + B$ that seem to violate the conservation of energy and mass. The model also disregards the intrinsic reversibility of elementary chemical steps. In other words, the model allows the reaction $A + B \to C$ without the corresponding reverse reaction $C \to A + B$. This is true, but it is necessary for modeling biochemical circuits within the cell, such as genetic regulatory networks that control the production of mRNA molecules (transcription) and of protein molecules (translation). Although no real reaction is strictly irreversible, many natural cellular reactions such as cleavage of DNA can be modeled as being effectively irreversible, or an implicit energy source (such as ATP) may be present in sufficiently high quantities to drive the reaction forward strongly. Thus, our models intrinsically assume that energy and mass are available in the form of chemical fuel (analogous to ATP, activated nucleotides, and amino acids) that is sufficient to drive reactions irreversibly and to allow the creation of new molecules. Together with the dependence of V on the total number of molecules, we envision the reaction solution as a two-dimensional puddle that grows and shrinks as it adsorbs fuel from and releases waste to the environment. This is very similar in spirit to computational models such as Turing Machines and Stack Machines that add resources (tape or stack space) as they are needed.

Another potentially unrealistic feature of the SCRN formalism is that it allows reactions of any order (any number of reactants), despite the generally accepted principle that all underlying physical chemical reactions are binary and that higher order reactions are approximations in situations with some very fast rate constants. For this reason, in our constructions, we restrict ourselves to use reactions with at most two reactants. Further, it is generally accepted that Michaelis–Menten kinetics are followed for catalytic reactions. For example, the above reaction $A + B \to C + B$ should be decomposed into two reactions $A + B \to M$ and $M \to C + B$ where M is some intermediate species, but the abbreviated intermediate-free form is also allowed in the model. Another principle involving catalysts is that if a reaction can occur in the presence of a catalyst, then it can usually also occur (albeit usually much more slowly) without the catalyst. For example, if $A + B \to C + B$ can occur, then so can $A \to C$. Continuing in this vein, a wealth of further restrictions, each applicable in certain contexts, could arise from detailed considerations of the types of molecules being used.

Instead of focusing on these or other restrictions, we focus on the cleanest and most elegant formalism for Stochastic Chemical Reaction Networks and treat it as a programming language. We happily leave the task of accurately implementing our networks to the synthetic chemists and synthetic biologists.

2.2 *Other Models of Chemical Computing*

It is worth noting that several other flavors of chemical system have been shown to be Turing universal. Bennett [18] sketched a set of hypothetical enzymes that will modify a information-bearing polymer (such as DNA) so as to exactly and efficiently simulate a Turing Machine. In fact, he even analyzed the amount of energy required per computational step and argued that if the reactions are made chemically reversible and biased only slightly in the favorable direction, an arbitrarily small amount of energy per computational step can be achieved. Since then, there have been many more formal works proving that biochemical reactions that act on polymers can perform Turing-universal computation (e.g., [19–21]). In all of these studies, unlike the work presented here, there are an infinite number of distinct chemical species (polymers with different lengths and different sequences), and thus formally, an infinite number of distinct chemical reactions. These reactions, of course, can be represented finitely using an augmented notation (e.g., “cut the polymer in the middle of any ATTGCAAT subsequence”), but as such they are not finite Stochastic Chemical Reaction Networks.

A second common way of achieving Turing universality is through compartmentalization. By having a potentially unbounded number of spatially separate compartments, each compartment can implement a finite state machine and store a fixed amount of information. Communication between compartments can be achieved by diffusion of specific species, or by explicit transfer reactions. This is, for example, exploited in the Chemical Abstract Machine [22] and in Membrane Systems [23]. Note that [24], contrary to its title, only establishes that Chemical Reaction Networks appear to be able to implement feed-forward circuits (along the lines of Sect. 3), making them empirically at least P-hard.

3 Bounded Models: Boolean Logic Circuits

A natural relation to Boolean circuits leads one to expect that Stochastic Chemical Reaction Networks may well have similar computational power. For example, given a circuit built from NAND gates, we can construct a corresponding Stochastic Chemical Reaction Network by replacing each gate

$$x_k = x_i \text{ NAND } x_j$$

with the four reactions

$$\begin{aligned} A_i + A_j &\to A_i + A_j + B_k, \\ A_i + B_j &\to A_i + B_j + B_k, \\ B_i + A_j &\to B_i + A_j + B_k, \\ B_i + B_j &\to B_i + B_j + A_k. \end{aligned}$$

The presence of a single A_i molecule represents that $x_i = 0$, the presence of a single B_i molecule represents that $x_i = 1$, and the presence of neither indicates that x_i has not yet been computed. If the circuit has only feed-forward dependencies, it is easy to see that if one starts with a single A or B molecule for each input variable, then with probability 1 the correct species will be eventually produced for each output variable. In this sense, a Stochastic Chemical Reaction Network can deterministically compute the same function as the Boolean circuit, despite the uncontrollable order in which reactions occur. Note that in this particular network, the specific rate constants can affect the speed with which the computation occurs, but do not change the eventuality.

Circuits of the same general flavor as the one above can be modified to work with mass action chemical kinetics [24, 25], showing that individual Boolean logic gates can be constructed, and that they can be connected together into a circuit. This provides for efficient computation but is a nonuniform model: the number of chemical species increases with the number of gates in the circuit, and thus with the size of the problem being solved.

Contrary to the limited (finite state) computational power of Boolean circuits, individual Stochastic Chemical Reaction Networks are not limited by finite state spaces: there may potentially be an unbounded number of molecules of any given species. As even minimal finite-state machinery coupled with unbounded memory tends to allow for Turing-universal computation, one might speculate that the same should hold true for Stochastic Chemical Reaction Networks. If so, then Stochastic Chemical Reaction Networks would be capable of uniform computation, and predicting their long-term behavior would be undecidable.

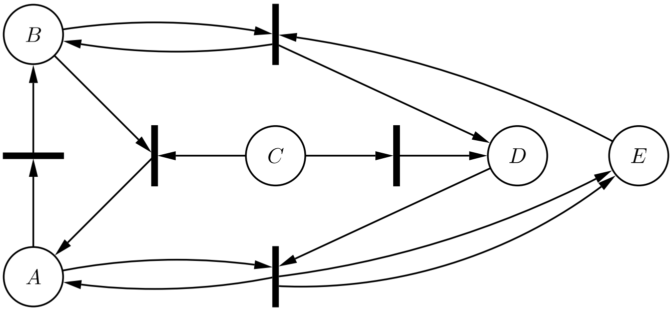
The following sections will show that this is indeed the case. Stochastic Chemical Reaction Networks are in fact much more powerful than one might think from the simple Boolean circuit approach shown above.

4 Unordered Program Models: Petri Nets and VASs

The main complicating factor when trying to "program" a Stochastic Chemical Reaction Network is that reactions occur in an uncontrollable order, making it quite a)

$$\begin{array}{rcl} A & \rightarrow & B \\ C & \rightarrow & D \\ B+C & \rightarrow & A \end{array} \qquad \begin{array}{rcl} A+D & \rightarrow & A+2E \\ B+E & \rightarrow & B+D \end{array}$$

b)



c)

$$\begin{array}{ccccccccc} & A & B & C & D & E & F & G & \\ \hline \langle & -1 & 1 & 0 & 0 & 0 & 0 & 0 & \rangle \\ \langle & 0 & 0 & -1 & 1 & 0 & 0 & 0 & \rangle \\ \langle & 1 & -1 & -1 & 0 & 0 & 0 & 0 & \rangle \\ \langle & -1 & 0 & 0 & -1 & 0 & 1 & 0 & \rangle \\ \langle & 1 & 0 & 0 & 0 & 2 & -1 & 0 & \rangle \\ \langle & 0 & -1 & 0 & 0 & -1 & 0 & 1 & \rangle \\ \langle & 0 & 1 & 0 & 1 & 0 & 0 & -1 & \rangle \end{array}$$

d)

$$\frac{3}{2} \quad \frac{7}{5} \quad \frac{2}{15} \quad \frac{13}{14} \quad \frac{242}{13} \quad \frac{17}{33} \quad \frac{21}{17}$$

Fig. 1 Four representations of the same computation. Starting with 1 A and n C's, the maximum number of D's that can be produced is 2^n. (**a**) A Stochastic Chemical Reaction Network. (**b**) A Petri net. Each circle corresponds to a *place* (a molecular species), and each black bar corresponds to a *transition* (a reaction). (**c**) A Vector Addition System. Note that dimensions F and G must be added to the Vector Addition System to capture the two reactions that are catalyzed by A and B. (**d**) A Fractran program. The numerators correspond to the reaction products, and the denominators correspond to the reactants. The first seven prime numbers are used here in correspondence to the letters A through G in the other examples. As in the previous example, F (13) and G (17) must be introduced here to avoid unreduced fractions for the catalyzed reactions

difficult to guarantee a unique outcome for a nontrivial computation. Stochastic Chemical Reaction Network computations that *are* arranged so as to guarantee a unique outcome will be called *confluent* computations.

We can find clues regarding how to program such systems, including relevant theorems, by examining the related computational models mentioned above and shown in Fig. 1. The differences between the models are minor, amounting mostly just to different interpretations or viewpoints of the same underlying fundamental process. For example, consider Petri nets [26], as shown in Fig. 1(b). In this model, a network consists of a directed bipartite graph, having connections between *places* (shown as circles) and *transitions* (shown as black bars). The state consists of a nonnegative number of *tokens* at each place, and a new state is achieved by the *firing* of a transition. When a transition fires, it consumes one token from the incident place for each incoming edge, and produces one token into the incident place for each outgoing edge (there is no difference between the two sides of the black bar). Thus, a transition is *enabled* only if there are enough tokens in the input places. In any given state, there are typically many transitions that could fire. Which one fires first is intentionally left unspecified: the theory of Petri nets addresses exactly the question of how to analyze asynchronous events. If the system uses rate constants as in (1) for each transition (in which case the model is a type of stochastic Petri net), the model is formally identical to Stochastic Chemical Reaction Networks: each place corresponds to a molecular species (the number of tokens is the number of molecules) and each transition corresponds to a reaction [27].

A closely related model, Vector Addition Systems (VASs), was developed and studied by Karp and Miller [10] for analyzing asynchronous parallel processes. Here, questions concern walks through an m dimensional integer lattice, where each step must be one of d given vectors $\mathbf{V}_\alpha \in \mathbb{Z}^m$, and each point in the walk must have no negative coordinates. Whether it is possible to walk from a point x to a point y (the *reachability* question) is in fact decidable [28]. It is also decidable whether it is possible for a walk to enter a linearly-defined subregion [29]—a special case is whether the ith component of the point ever becomes nonzero (the *producibility* question).

The correspondence between Vector Addition Systems, Stochastic Chemical Reaction Networks and Petri nets is direct. First, consider chemical reactions in which no species occurs both on the left side (as a reactant) and on the right side (as a product)—i.e., reactions that have no *instantaneous catalysts*. When such a reaction α occurs, the state of the Stochastic Chemical Reaction Network, represented as a vector, changes by addition of the vector $\mathbf{p}_\alpha - \mathbf{r}_\alpha$. Thus, the trajectory of states is a walk through $\mathbb{Z}^m$ wherein each step is any of d given vectors, subject to the inequalities requiring that the number of molecules of each species remain nonnegative, thus restricting the walk to the nonnegative orthant.

Karp and Miller's decidability results for VASs [10] directly imply the decidability of the question of whether a catalyst-free Stochastic Chemical Reaction Network can possibly produce a given target molecule (the *producability* question again). As a consequence, confluent computation by Stochastic Chemical Reaction Networks cannot be Turing universal, since questions such as whether the YES output molecule or the NO output molecule will be produced are decidable. The restriction to catalyst-free reactions is inessential here: each catalytic reaction can be replaced by two new reactions involving a new molecular species (an "intermediate state", see Fig. 1(c)), after which all reachability and producibility questions (not involving the new species) are identical for the catalyst-free and the catalyst-containing networks.

4.1 Gate Implementability

The initial path leading the authors to consider the computational power of Stochastic Chemical Reaction Networks came from a surprisingly unrelated topic. We were considering the general question of whether circuits constructed from available gate types are able to implement a desired target function. We call this the *gate implementability* question. The terms *gate* and *function* will be used interchangeably here.

It has been known since the time of Post [30] that given a set of functions of boolean values, only a finite number of tests need to be done to know whether a particular target function can or cannot be implemented by them, if function values, once produced, can be used repeatedly (in other words, if *fan-out* is available). However, in situations where values cannot be used repeatedly (as is the case, for example, in quantum computation), the implementability question becomes much less clear. Indeed, if the analogous questions are asked for circuits built of *relations*, rather than *functions*, then the ability to reuse values makes this question become decidable, whereas it is undecidable if values, once produced, can only be used once [31].

It is natural to wonder, if fan-out is *not* available, might the gate implementability question become *un*decidable, as it did for relations?

First of all, we have to be clear about what we mean by "circuits without fan-out." From a feed-forward point of view, a fan-out node in a circuit is a device with one input and two outputs, and both outputs equal the input. So, we will be generous and expand the definition of "function" to allow multiple outputs. (If we do not do this, then all circuits must be trees, and it becomes difficult to implement anything at all, since in contrast with formulas, inputs cannot be used at more than one leaf of the tree.) We will define the outputs of a feed-forward circuit to be all of the output wires which have not been fed into some other gate, and the inputs are of course all the input wires which are not produced as the output of another gate.

This gives us an implementability question for feed-forward circuits that is comparable to the implementability question for relations. As with relations, the availability of fan-out makes the question easily decidable: Simply iteratively expand the set of implementable functions, starting with the inputs and the given functions. However, without fan-out available, the situation is not quite so easy.

4.2 Gate Implementability Is Equivalent to Reachability in Stochastic Chemical Reaction Networks

In this section, we will show that any gate implementability question can in fact be reduced to a reachability question for a chemical reaction network, and vice versa. Intuitively, the idea is that the wires in the circuit correspond to molecules, the gates in the circuit correspond to reactions, the designer of the circuit corresponds to the source of randomness in the Stochastic Chemical Reaction Network, and the ability to implement a given function corresponds to the reachability question for the Stochastic Chemical Reaction Network.

The idea for the forward direction is that we consider all possible inputs to the circuit simultaneously. Since we know what we are trying to implement, we know how many inputs there are, and what the possible values for each input are, and thus we know exactly how many distinct possible states the entire circuit can be in. For example, if there are five Boolean inputs, then there are $2^5 = 32$ possible states for the circuit (one for each possible combination of values on the input wires), and every wire in the circuit can have its behavior described by a vector of length 32, giving the value of that wire in each of the 32 possible states the circuit might be in. In this example, the five inputs to the circuit would be described by the following vectors:

$$\langle 0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1\rangle,$$

$$\langle 0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1\rangle,$$

$$\langle 0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1\rangle,$$

$$\langle 0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1\rangle,$$

$$\langle 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1\rangle.$$

The vector describing an output of a gate is easily calculated from the vectors for the inputs.

The corresponding chemical reaction network will be designed to have one species for each possible vector. (In the example above, there would be 2^{32} species.) Then each gate available in the implementability question is converted into a long list of chemical reactions: For each possible combination of input vectors to the gate, we provide a chemical reaction which takes those species as reactants and produces the appropriate species (those corresponding to the correct outputs of the gate for these inputs) as products.

The starting state for the chemical reaction network is one molecule of each of the species used as inputs (in the example above, recalling that each vector is a species, the starting state would be the five listed vectors), and the target state for the reachability question is simply the corresponding set of output vector species for the target gate in the implementability question. It is clear from the design that the target state is reachable in the chemical reaction network if and only if the target gate is implementable in the implementability question.

Now we will show the other direction, that any reachability question for a chemical reaction network can be reduced to an implementability question for gates without fan-out.

The idea for this direction is to design some gates that can only be usefully combined by following exactly the reactions of the given network. The alphabet of values used by the gates will consist of one symbol for each of the chemical species, plus an extra symbol "ϵ", which we will think of as an error symbol. There will be one gate per reaction, plus one extra gate. Each reaction will be converted into a gate with as many inputs as reactants and as many outputs as products. For example, the reaction $A + 2B \rightarrow C + D$ would become a gate with 3 inputs and 2 outputs, and the computation performed by the gate is almost trivial: It outputs ϵ on every output, *unless* its inputs are $\langle A, B, B \rangle$, in which case it outputs $\langle C, D \rangle$. Other reactions are similarly converted. We also provide an extra gate with two inputs and two outputs, which is a two-wire identity gate, except that if *either* input is ϵ, then *both* outputs are ϵ. Otherwise, the first output matches the first input, and the second output matches the second input. The purpose of this gate is to allow the error symbol ϵ to spread from one wire to another, as we will see shortly.

The initial state and target state for the reachability question then become the inputs and outputs of the target gate, and again every other possible input should lead to all outputs being ϵ.

Any satisfactory solution to this implementability question clearly corresponds to a partially ordered sequence of reactions that demonstrates a positive answer to the reachability question. Conversely, any sequence of reactions reaching the target state of the reachability question can be directly converted into a circuit of gates that is *almost* guaranteed to implement the target gate. The only potential problem is that if the input given to the circuit differs just slightly from the intended input, then some of the gates will still be getting exactly the inputs that were intended, and for some circuits, it may not be the case that *all* outputs are ϵ, but rather just some subset of them. It is for this reason that we supplied the extra "error propagating" gate. If necessary, this gate can be used many times at the end of a circuit ($2n - 3$ times for a circuit with n outputs) to ensure that if any outputs are ϵ, then all outputs must be ϵ. Clearly, the availability of this gate will not otherwise affect the ability to simulate the sequence of reactions. Thus, the answer to the gate implementability question will match exactly the answer to the chemical reachability question.

5 Almost Universal: Primitive Recursive Computation

It has long been known that certain questions about whether a Petri net "might do X" are decidable, where typical values of X are, in the language of Stochastic Chemical Reaction Networks, "keep having reactions forever" or "grow without bound" or "reach a certain state" or "produce at least some given quantities of given species." [9, 10, 28]. These results carry over directly to Stochastic Chemical Reaction Networks so long as the question does not ask about the *probability* of X happening, but only about the *possibility* of it happening (i.e., only about whether the probability of X is zero vs. nonzero).

As mentioned in Sect. 4, confluent computation by Stochastic Chemical Reaction Networks can only implement decidable decision problems. Thus, for questions about the output of a Stochastic Chemical Reaction Network (given by some final quantity of the output species) to have any hope of being undecidable, the output *must* be probabilistic in nature. We will examine questions of probability in Sect. 6; here, we restrict ourselves to questions of possibility.

Although the questions of possibility listed above are known to be decidable, their complexity is sometimes not so clear. The complexity of the problem for X =

"grow without bound" is known to be doubly exponential [32], but the complexity of the problem for X = "reach a certain state" has been an open problem for decades [9].

Even though double exponential complexity sounds quite complex, the complexity of these types of problems can in fact be far greater. Some suspect that the reachability problem (i.e., X = "reach a certain state") may have complexity comparable to primitive recursive functions, which are so powerful that few natural nonprimitive recursive functions are known.

In Sect. 5.3, we present examples of problems whose complexity does exactly match the power of primitive recursive functions. Specifically, if X = "have a molecule of S_1 present when attaining the maximum possible amount of S_2", or X = "have a molecule of S_1 present after taking the longest possible (over all sequences) sequence of reactions." These questions are equivalent in power to primitive recursively defined predicates, where the number of primitive recursive functions used to recursively build up the predicate is on the order of the number of molecular species in the Stochastic Chemical Reaction Network, and the input to the predicate corresponds to the initial state of the Stochastic Chemical Reaction Network.

To show that such question are no more powerful than primitive recursive functions, in Sect. 5.2 we show that for any Stochastic Chemical Reaction Network, it is possible to define a primitive recursive function which can return the amount of S_1 that is produced by whichever sequence of reactions leads to the largest possible amount of S_2. Our proof, while far from straightforward, is much simpler than previous similar proofs (which used results on bounds for solutions to bounded versions of Hilbert's tenth problem), since it gives an explicitly primitive recursive formula bounding the size of the tree of all possible runs of the Stochastic Chemical Reaction Network. The bulk of the proof lies in defining this bounding function and proving that it indeed bounds the depth of the tree. This bound enables the definition of a primitive recursive function which analyzes the entire tree, explicitly finding the run with the largest amount of S_2 and returning the corresponding amount of S_1.

5.1 Primitive Recursive Functions

Primitive Recursive Functions were first investigated in the 1920s, starting with Skolem [33], who pointed out that many standard functions on nonnegative integers can be defined using just function composition and recursion, starting with just the successor function. This surprising fact is illustrated in Fig. 2, which shows how functions can be built up in this way, including for example a function that will tell you whether a number is prime or not.

The wide range of functions that could be defined in this way led logicians to wonder whether all mathematical functions could be defined in this way, or at least all those functions for which there exists a deterministic algorithm for calculating the value. Recall that this was long before people had ever written algorithms for

		Recursive Definition	
Function	Name	if $n = 0$	if $n = m + 1$
$S(n) = n + 1$	successor		
$A(n, a) = n + a$	addition	a	$S(A(m, a))$
$M(n, a) = n \times a$	multiplication	0	$A(a, M(m, a))$
$E(n, a) = a^n$	exponentiation	1	$M(a, E(m, a))$
$V(n) = \text{sign}(n)$	positivity	0	1
$P(n) = n - 1$	predecessor	0	m
$D(a, n) = a - n$	subtraction	a	$P(D(a, m))$
$R(n, a) = n \bmod a$	remainder	0	$\left\{\begin{array}{l} M(S(R(m, a)), \\ \quad V(D(P(a), \\ \qquad R(m, a)))) \end{array}\right.$
$C(n, a) = \prod_{i=2}^{n+1} i \bmod a$	mod product	1	$\left\{\begin{array}{l} M(C(m, a), \\ \quad R(a, S(S(m)))) \end{array}\right.$
$Z(n) = 1$ if n is prime	primality	0	$\left\{\begin{array}{l} V(M(m, \\ \quad C(P(m), S(m)))) \end{array}\right.$

Fig. 2 Examples of Primitive Recursive Functions. Starting with only the successor function, other functions can be built up one by one using a simple form of recursion. Where the function being defined is used in its own recursive definition, the rule is that it must have exactly the same arguments but with *n* replaced by *m*

electronic computers, before Gödel's famous incompleteness theorem [34] and before Turing Machines [35], in short, before people had figured out any satisfactory way of standardizing or formalizing the process of mathematical calculation. Perhaps this was the way.

It turned out that this was not the way. In 1928, Ackermann [36] showed that there is a limit to how fast a Primitive Recursive Function can grow (depending on how many other functions are used to help define it), and there turns out to exist simple deterministic algorithms for calculating functions that grow even faster than this limit, as shown in Fig. 3. Thus, the world of Primitive Recursive Functions is not large enough to encompass all mathematical calculations.

Not long after Ackermann's paper made it clear that Primitive Recursive Functions were merely a strict subset of the functions that can be calculated, Herbrand in 1931 [37] and Gödel in 1934 [38] defined General Recursive Functions, which in 1936 were argued by both Church [39] and Turing [35] to correspond exactly to the set of all functions that can possibly be calculated in any algorithmic way. This argument was accepted by most people, and is now well known as the Church–Turing thesis.

A major distinction between the General Recursive Functions and the Primitive Recursive Functions is that the latter (and also Ackermann's function) are defined for all inputs—that is to say, computation eventually halts and produces an output, no matter what the input is—whereas the former include additional functions, some of which halt only for some inputs. Figuring out which General Recursive Functions halt for which input is known as the Halting Problem, and it is formally undecidable: there is no General Recursive Function that will always correctly determine whether a given algorithm halts.

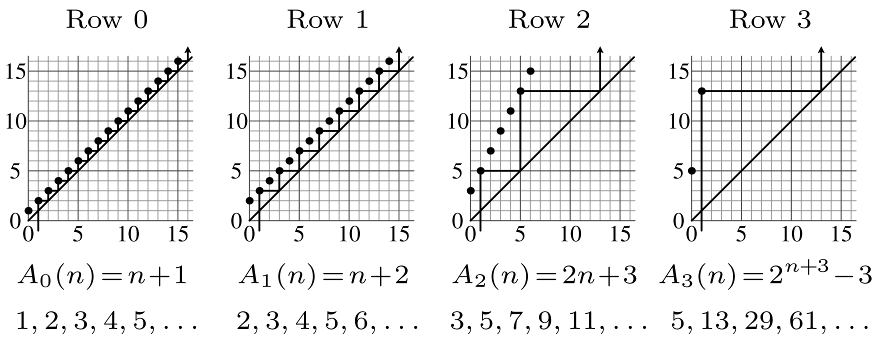


Fig. 3 An illustration of the Ackermann function. The Ackermann function $A_i(n)$ is a function of two variables, i and n. The ith row of the Ackermann function, A_i, can be constructed visually from the previous row A_{i-1} as shown: A *zig-zag line* starts going up at $x = 1$, and bounces back and forth between the function values (shown as *dots*) and the line $x = y$. The function values hit by the *zig-zag line* become the entries for the next row. The formal definition is $A_0 = S$, $A_{i+1}(0) = A_i(1)$, $A_{i+1}(m+1) = A_i(A_{i+1}(m))$. Although each row is a Primitive Recursive Function, the diagonal $f(n) = A_n(n)$ grows faster than any Primitive Recursive Function in the same sense that 2^n grows faster than any polynomial

While most people turned their attention at this point to General Recursive Functions, Rózsa Péter [40] continued to develop the theory of Primitive Recursive Functions, treating them not as a historical mistake, but as an opportunity for study. Her work makes it clear that the following definition is an equivalent way to define Primitive Recursive Functions.

Definition Primitive Recursive Functions are exactly those functions which can be computed by a Turing Machine in time bounded by *some* row of the Ackermann function.

This definition makes it evident that just about every algorithm ever used for practical calculation is in fact Primitive Recursive, since most rows of the Ackermann function grow far faster than the time required for any practical calculation.

Although Péter's work showed that many seemingly different definitions all lead to this same set of functions, the definitions were rather abstractly mathematical in nature, none of them corresponding to what we would think of today as a fundamental computational model like a Turing Machine. So, it is interesting that Primitive Recursive Functions arise here in relation to Stochastic Chemical Reaction Networks, a fundamentally reality-based model.

5.2 A Primitive Recursive Bound on the Depth of the Tree of Reachable States

Theorem 1 *Given two states $\mathcal{A}$ and $\mathcal{B}$, in order to determine whether starting from $\mathcal{A}$ a Stochastic Chemical Reaction Network can reach a state with at least as many*

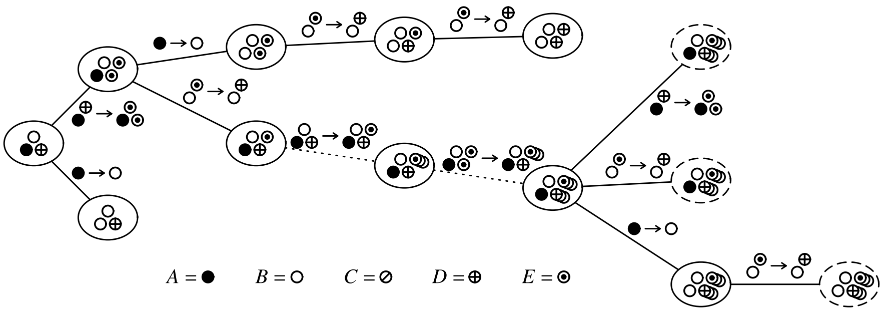


Fig. 4 The search tree for the system of Fig. 1, starting on the left with state (A, B, D). *Solid lines* represent single reactions, while *dotted lines* represent any number of further repetitions of a completed cycle that purely increases a molecular quantity, leading to the attainability of arbitrarily large quantities of that species shown, for example, as ⊙⟫. The *dashed circles* are repeats of previous states, and thus do not require further exploration even if further reactions are possible. In this example, the search tree is finite. Must this always be the case? If so, then there are no undecidable questions among questions which can be answered by scanning the full search tree. This section shows that the search tree is finite, and indeed boundable by a primitive recursive function

molecules as $\mathcal{B}$ is decidable and requires a search tree of size bounded by a primitive recursive function of the number of molecules of each species and the stoichiometric coefficients of the reactants.

Here, we present the details of our proof that the tree of possible execution paths of a Stochastic Chemical Reaction Network has depth bounded by a primitive recursive function whose "degree" is on the order of the number of species in the Stochastic Chemical Reaction Network.

For those familiar with the subject, the algorithm is nearly identical to Karp and Miller's [10], but the rest of the proof is much more direct than comparable previous proofs which relate other questions about the tree to primitive recursive functions. See also [15].

The Algorithm

In this section, we will present an algorithm for finding which species can be produced and which cannot. That is, it will find out whether any reachable states have nonzero levels of any species of interest. In fact, it will do slightly more: For any given set of molecule quantities (such as $(10A, 3B, \ldots)$), the algorithm can find out whether or not it is possible to reach any state that has at least these levels of these species.

The algorithm is simply to search through the full tree of all possible reaction sequences, using a couple of simple tricks to try to avoid getting stuck in infinite loops.

If state $\mathcal{B}$ has at least as many molecules of each species as state $\mathcal{A}$ does, then we will say that $\mathcal{B} \geq \mathcal{A}$. On the other hand, if $\mathcal{B}$ has more of some species and less of others than $\mathcal{A}$ has, we say that $\mathcal{B}$ and $\mathcal{A}$ are incomparable: $\mathcal{A} \ngeq \mathcal{B}$ and $\mathcal{B} \ngeq \mathcal{A}$.

A fundamental observation is that if the system is in state $\mathcal{A}$ at some point, and then later it is in state $\mathcal{B}$, and $\mathcal{B} \geq \mathcal{A}$, then the sequence of reactions that led from $\mathcal{A}$ to $\mathcal{B}$ may be repeated arbitrarily many times before continuing. This would appear to be a serious obstacle to exhaustively searching the space of reachable states, but in fact it will be the key to bounding the search. When this happens, we can consider two cases: $\mathcal{B} = \mathcal{A}$ or $\mathcal{B} > \mathcal{A}$.

If $\mathcal{B} = \mathcal{A}$, then this sequence of reactions leading from $\mathcal{A}$ to $\mathcal{B}$ had no effect, and may be omitted entirely. In particular, it is clear that the shortest sequence of reactions leading from the initial state of the system to any particular final state will not visit any state more than once. Thus, no possibilities will be missed if the search tree is simply pruned at any point where a previous state is repeated.

On the other hand, if $\mathcal{B} > \mathcal{A}$, that is, if $\mathcal{B}$ has strictly more of some species than the earlier state $\mathcal{A}$ had, then by repeating this sequence of reactions, an arbitrarily large amount of those species may be produced. We will call such species *freely generatable* after the sequence of reactions from $\mathcal{A}$ to $\mathcal{B}$ has occurred. If at any later point in the calculation, some potential reaction is not possible because one of the freely generatable species has run out, we can simply retroactively assume that more repeats of the sequence from $\mathcal{A}$ to $\mathcal{B}$ were performed back at the time when that species became freely generatable, and this will allow the potential reaction to proceed after all. For this reason, when a species becomes freely generatable, it may effectively be removed from the problem statement, reducing the problem to a simpler problem. So, although the search tree cannot be pruned when $\mathcal{B}$ is reached, the subtree beyond that point corresponds to searching the space of a simpler problem, in which a further repetition of the reaction sequence leading from $\mathcal{A}$ to $\mathcal{B}$ would indeed lead to pruning, since states $\mathcal{A}$ and $\mathcal{B}$ are equal in the reduced problem. The algorithm therefore specifies the quantity of a freely generatable species as ∞, a value which is considered larger than any other value, and which is unchanged by the addition or removal of molecules.

It remains to show that the search tree is always finite, and thus this algorithm will always terminate.

The Data Structure

Now, we will define a data structure whose purpose will be to help us define the bound in the next section.

At each point in the search tree, there is a (usually infinite) set $\mathbb{S}$ of all states $\mathcal{S}$ satisfying $\mathcal{S} \ngeq \mathcal{A}$ for every $\mathcal{A}$ which is an ancestor of that point in the search tree. We will call this set of states $\mathbb{S}$ the *remaining states* for that point in the search tree, because these are the states which, if reached on the next step, will not lead to pruning or simplification. Our proof will examine this set of states and use the structure of this set to provide a bound on how much deeper the search tree can be.

Fig. 5 *Left*: An example of a possible entry in list $\mathbb{L}_4$, for a system with 7 species. *Right*: All the entries that will be added to list $\mathbb{L}_3$ to replace the entry on the left, if the system arrives at state $(2, 4, 1, 3, 3, 3, 0)$. The union of the new 3-dimensional regions is precisely that portion of the old 4-dimensional region which is $\ngeq$ the new state

$$\begin{array}{ccc}
\underline{\qquad\mathbb{L}_4\qquad} & & \underline{\qquad\mathbb{L}_3\qquad} \\
 & & \vdots \\
 & & (2,0,\cdot,5,\cdot,3,\cdot) \\
\vdots & & (2,1,\cdot,5,\cdot,3,\cdot) \\
 & & (2,2,\cdot,5,\cdot,3,\cdot) \\
 & & (2,3,\cdot,5,\cdot,3,\cdot) \\
(2,\cdot,\cdot,5,\cdot,3,\cdot) & \rightsquigarrow & (2,\cdot,0,5,\cdot,3,\cdot) \\
 & & (2,\cdot,\cdot,5,0,3,\cdot) \\
\vdots & & (2,\cdot,\cdot,5,1,3,\cdot) \\
 & & (2,\cdot,\cdot,5,2,3,\cdot) \\
 & & \vdots
\end{array}$$

For any given point in the search tree, we represent the set of remaining states by lists $\mathbb{L}_i$, with each entry in list $\mathbb{L}_i$ representing an i-dimensional region of remaining states, specified by $n - i$ integers (specifying quantities of $n - i$ of the n species). The union of all regions from all lists exactly yields the set of remaining states for the given point in the search tree.

When a reaction takes the system to a new state (taking the search to a new point in the search tree), the lists are modified by eliminating each list entry which represents a region containing any state greater than or equal to the new state. Each eliminated entry is replaced by new entries in the list of next lower index. The new entries are found by considering all regions of dimension one less than the old region, lying within the old region, with a previously unspecified coordinate now specified as some particular integer k, with $0 \leq k < m$, where m is the number of molecules present, in the new state, of the species corresponding to the dimension now being specified. In general, this might lead to some redundancy, if some of the new regions lie inside other existing regions, but we will not need to worry about this.

The lists for the initial state of the system are created similarly, with the "old" region taken to be the full n-dimensional space, just a single entry in list $\mathbb{L}_n$. Thus, a system started in state $(q_1, q_2, \ldots, q_n)$, where q_i is the quantity of the ith species, will start with $\sum_i q_i$ entries in list $\mathbb{L}_{n-1}$. Similarly, whenever an entry in list $\mathbb{L}_i$ is replaced by new entries in list $\mathbb{L}_{i-1}$ due to a new state $(q_1, q_2, \ldots, q_n)$, the number of new entries will be $\sum_{i \in P} q_i$, where P is the set of species whose quantity is unspecified in the old entry.

If the ith species becomes freely generated, all list entries in all lists will have their ith entry changed to be specified as ∞, which may move some of them to the list of next lower index: Since ∞ is treated by the lists as a specified quantity, any list entry which previously did not specify the quantity of the ith species will now have one fewer unspecified quantities, and will thus move to the list of next lower index.

It remains to show that these lists eventually get exhausted as the algorithm progresses deeper into the tree. For readers familiar with the game of Chomp [41], this process is quite similar to Chomp on infinite boards.

The Bound

To each point in the search tree, with its state and its lists, we can assign a positive integer as described below. We will see that regardless of which reaction is performed at the next step, the positive integer assigned to the ensuing point in the search tree will always be less than the positive integer assigned to the current point. Since the positive integer strictly decreases with depth, it is in fact a bound on the depth.

The integer for a given state $\mathcal{A}$ and lists $\mathbb{L}_i$ is defined for a system with n species in the following nontrivial way:

$$\mathfrak{B}(\mathcal{A}, \mathbb{L}) = f_{n-1}^{|\mathbb{L}_{n-1}|}\big(f_{n-2}^{|\mathbb{L}_{n-2}|}\big(\cdots\big(f_1^{|\mathbb{L}_1|}\big(f_0^{|\mathbb{L}_0|+m\cdot r+q_{\max}}(0)\big)\big)\cdots\big)\big)$$

where r is the number of nonfreely generatable species, $q_{\max}$ is the largest number of molecules present of any of those r species, and m, a constant, is one more than the maximum coefficient appearing on the right-hand side of any reaction.

The functions f_i are defined as follows:

$$f_i(x) = f_{i-1}^{i\cdot x+m}(x),$$
$$f_0(x) = x + 1.$$

These definitions are not meant to capture intuitive notions of any meaningful functions, but rather are designed to (a) be explicitly primitive recursive, and (b) be of a form that enables the necessary proof steps below to work.

In these definitions, the exponents on the functions denote multiple applications of the function, so, for example, $f_8^3(x) = f_8(f_8(f_8(x)))$. Each f_i, as well as $\mathfrak{B}$, is a Primitive Recursive Function, since it is easy to define repeated application of a function: Given a function $g(x)$, we can define $h(n, x) = g^n(x)$ using the recursive definition $h(0, x) = x$, $h(m + 1, x) = g(h(m, x))$.

It is straightforward to show that the functions $f_i(x)$ are strictly increasing in x, and that $f_{i+1}(x) > f_i(x)$. Thus, if the exponents appearing within the definition of $\mathfrak{B}$ are in any way reduced or shifted to the right, $\mathfrak{B}$ will decrease.

This can be used to show that regardless of whether a reaction leads to a remaining state or leads to a new freely generatable species; $\mathfrak{B}$ will always decrease.

If a reaction results in one or more freely generatable species, then some parts of the exponents may shift to the right, and r will decrease. In the exponent of f_0, the decrease of r will more than make up for any increase in $q_{\max}$ (by the definition of m), so $\mathfrak{B}$ will decrease as promised.

If a reaction leads to a remaining state, then one or more list entries will be replaced by other entries. Each i-dimensional entry to be removed will be replaced by $\sum_{j\in P} q_j$ entries that are $(i - 1)$-dimensional. This number of new entries is no more than $i \cdot q_{\max}$, since P, the set of species of unspecified quantity, is of size i. So, the exponent of f_i is reduced by 1 while the exponent of f_{i-1} increases by at most $i \cdot q_{\max}$. In the formula for $\mathfrak{B}$, then an f_i gets replaced with $f_{i-1}^{i\cdot q_{\max}}$, and then this exponent is possibly reduced. But the original f_i was equivalent (by definition) to $f_{i-1}^{i\cdot x+m}$, where x is the full argument (which must be at least $q_{\max}$, since $q_{\max}$ appears in the exponent of f_0), so even just the $f_{i-1}^{i\cdot x}$ portion was bigger than the replacement, and the f_{i-1}^{m} portion more than compensates for any increase in the exponent of f_0 due to any change in $q_{\max}$. The total effect is therefore again a decrease in $\mathfrak{B}$.

Thus, we have finished showing that $\mathfrak{B}$, a primitive recursive function of the initial state, bounds the depth of the search tree. Thus, both the depth of the tree, and its total size (being at most exponential in the depth), are not only finite but bounded by a primitive recursive function of the initial state. In the next section, we will see examples which cannot be bounded by anything smaller than this.

5.3 The Max-Path Problem

We have shown that any Chemical Reaction System can be analyzed by Primitive Recursive Functions, but the reverse question is also interesting: Can any Primitive Recursive Function be calculated by a Chemical Reaction System? This question raises conceptual issues not present in the forward question, since Chemical Reaction Systems are inherently nondeterministic, it being unspecified at each step which reaction should occur next. Thus, one must choose how to define which of the possible sequences of reactions should be considered as leading to the correct (or incorrect) calculation of the function. If one chooses, say, the longest possible sequence of reactions (the deepest leaf in the search tree), or the sequence that leads to the most molecules being produced (either of all species put together, or of some particular species), then it is indeed possible to calculate any Primitive Recursive Function, where the input and output are given as numbers of molecules of certain species. These choices provide an exact equivalence in power between Chemical Reaction Systems and Primitive Recursive Functions. Admittedly, this is not a practically useful notion of calculation by a SCRN—if I have the chemicals in my lab, how do I perform an experiment that indicates the output of the computation?—but it does help clarify the boundary between decidable and undecidable questions about SCRNs.

Theorem 2 *For any primitive recursive function f, a Stochastic Chemical Reaction Network can be designed with special species S_{in}, S_{out}, and $S_{\max}$ computing f as follows. Starting with n molecules of S_{in} (and some fixed number of molecules of other species), the reachable state with the maximal amount of $S_{\max}$ will have exactly $f(n)$ molecules of S_{out}.*

We prove this theorem with a construction. We begin by presenting, for any chosen fixed integer i, a SCRN that Max-Path-computes the ith row of the Ackermann function. This simple example of Max-Path "computing" by SCRNs is enlightening in and of itself, but more importantly, it plays a crucial role in our general construction, where it is used to bound the number of steps taken by a Register Machine that computes the Primitive Recursive Function in question.

$$\begin{array}{ccc}
 & Y_0 \to X + W & \\
 & Y_1 \to X + Y_0 & \\
X + Y_2 \to Z_1 + Y_2 & Y_2 \to X + Y_1 & W + Z_1 \to Y_1 \\
X + Y_3 \to Z_2 + Y_3 & Y_3 \to X + Y_2 & W + Z_2 \to Y_2 \\
\vdots & \vdots & \vdots \\
X + Y_i \to Z_{i-1} + Y_i & Y_i \to X + Y_{i-1} & W + Z_{i-1} \to Y_{i-1}
\end{array}$$

Fig. 6 A Chemical Reaction System for nondeterministically computing entries for the first i rows of the Ackermann function using $2i + 2$ species. However, as shown in this paper, no Chemical Reaction System with a finite number of species is able to compute *all* rows of the Ackermann function. To compute an entry in the ith row, $A_i(n)$, start this Chemical Reaction System in the state (Y_i, nX). Then the maximum number of molecules of X that can be produced is exactly $A_i(n)$, achievable by always choosing the first possible reaction from the list

SCRNs for Rows of the Ackermann Function

Figure 1(a) gives a SCRN that computes 2^n. This example can be generalized to compute any chosen row of the Ackermann function. Since the Ackermann function grows faster than any primitive recursive function, the full Ackermann function cannot be Max-Path computed by any single SCRN; using a different SCRN for each row of the function is the best we could hope to do.

We prove that the construction works by proving the two sides: First, we prove that starting in state (Y_i, nX) we *can* produce $A_i(n)$ X's. Second, we prove that no more than $A_i(n)$ X's can be produced.

We prove the first part by induction on row index i. Our inductive assumption will be that from (Y_{i-1}, nX) we can get to $(W, A_{i-1}(n)X)$. (The base case is easy to confirm.) Now starting with (Y_i, nX), we first convert all X's to Z_{i-1}'s by reactions in the first column. Then through a reaction in the second column we reach (Y_{i-1}, X, nZ_{i-1}), and the inductive assumption allows us to reach $(W, A_{i-1}(1)X, nZ_{i-1})$. Now we repeatedly use the first possible reaction in the third column, producing (Y_{i-1}, same X, one fewer Z_{i-1}), followed by the inductive assumption, producing (W, A_{i-1}(previous X), same Z_{i-1}), until we can no longer use that reaction in the third column. At this point, we have produced

$$\big(W, \underbrace{A_{i-1}\big(\cdots\big(A_{i-1}(}_{n+1 \text{ times}}1)\big)\big)X\big) = \big(W, A_i(n)X\big).$$

This shows that it is indeed possible to produce $A_i(n)$ X's.

Now, we argue that no more than $A_i(n)$ X's can be produced from (Y_i, nX). The proof consists of showing that the expression

$$T_i\big(T_{i-1}\big(\cdots\big(T_2\big(A_1^{\#Y_1}\big(A_0^{\#Y_0}(\#X)\big)\big)\big)\big)\big) \quad \text{where } T_i(m) = A_{i-1}^{\#Z_{i-1}}\big(A_i^{\#Y_i}(m)\big)$$

does not increase no matter which reaction is performed, assuming there is a exactly one of the Y's or W present (an invariant enforced by our system). Since the initial value of this expression is $A_i(n)$ when starting in (Y_i, nX), we would then know that no more than $A_i(n)$ X's can be produced.

The following two lemmas are useful.

Lemma 1 $A_i(A_j(m)) \geq A_j(A_i(m))$ *for* $i > j$.

Proof If $i > j$, then $A_j(A_i(m)) \leq A_{i-1}(A_i(m)) = A_i(m+1) \leq A_i(A_j(m))$. □

Lemma 2 $A_i(m) \geq A_{i-1}^2(m)$.

Proof First, we expand $A_i(n) = A_{i-1}(A_{i-1}(\cdots(1)))$ where the composition occurs $n+1$ times. Except in edge cases, the lemma is then equivalent to showing that $A_{i-1}(\cdots(1)) \geq n$ where the composition occurs $n-1$ times. This inequality holds because applying the Ackermann function increases the argument by at least one. □

Now, we will use these lemmas to show that each of the three types of reactions (in the three columns) does not increase our expression.

Consider the reaction $X + Y_i \to Z_{i-1} + Y_i$. The reaction takes subexpression $T_i(T_{i-1}(\cdots(\#X))) = A_{i-1}^{\#Z_{i-1}}(A_i(T_{i-1}(\cdots(\#X))))$ to subexpression $A_{i-1}^{\#Z_{i-1}+1}(A_i(T_{i-1}(\cdots(\#X-1))))$. The start subexpression is equal to

$$A_{i-1}^{\#Z_{i-1}}\big(A_i\big(T_{i-1}\big(A_0\big(\cdots(\#X-1)\big)\big)\big)\big) \geq A_{i-1}^{\#Z_{i-1}}\big(A_i\big(A_0\big(T_{i-1}\big(\cdots(\#X-1)\big)\big)\big)\big)$$

using the first lemma. Since $A_{i-1}(A_i(m-1)) = A_i(m)$, this expression equals the end subexpression.

Now, consider the reaction $Y_i \to X + Y_{i-1}$. It takes the subexpression $A_i(A_{i-2}^{\#Z_{i-2}}(T_{i-2}(\cdots(\#X))))$ to the subexpression

$$A_{i-2}^{\#Z_{i-2}}\big(A_{i-1}\big(T_{i-2}\big(\cdots(\#X+1)\big)\big)\big) \leq A_{i-1}^2\big(A_{i-2}^{\#Z_{i-2}}\big(T_{i-2}\big(\cdots(\#X)\big)\big)\big)$$

by applications of the first lemma. This is not more than the original subexpression by the second lemma.

Lastly, consider the reaction $W + Z_i \to Y_i$. This reaction takes subexpression $A_i^{\#Z_i}(A_{i-1}^{\#Z_{i-1}}(T_{i-1}(\cdots(\#X))))$ to $A_i^{\#Z_i-1}(A_{i-1}^{\#Z_{i-1}}(A_i(T_{i-1}(\cdots(\#X)))))$, which is not greater than the original by applying the first lemma.

SCRNs for Primitive Recursive Functions

Now, we show that given any primitive recursive function f, a Stochastic Chemical Reaction Network can be designed so that the state with the maximal amount of S_2 will have exactly $f(n)$ molecules of S_1, where n is given as input by being the number of molecules of an input species S_3 when the system is started. We sketch the proof here.

Any primitive recursive function can be computed by a Register Machine[2] in time bounded by some row of the Ackermann function (see Sect. 5.1). The required

[2] See Sect. 6 for a description of Register Machines and Broken Register Machines, and how SCRNs can be designed to simulate Broken Register Machines.

row can be determined by a structural examination of the primitive recursive function. Our Stochastic Chemical Reaction Network is designed to first compute an upper bound B on the running time needed to compute f by computing the appropriate row of the Ackermann function as in the previous section.

The Stochastic Chemical Reaction Network then simulates a Broken Register Machine (that is, a Register Machine whose decrement instructions may fail nondeterministically even when the register is not empty) for B steps, which we know is more than enough time for the Register Machine program to finish. After each of the B steps (with the `halt` instruction changed to a `nop` (no operation) instruction so that B steps can indeed occur), the Stochastic Chemical Reaction Network passes control to a "subroutine" which doubles the amount of S_2 (actually, all it can do is allow the amount of S_2 to at most double, but that is good enough). In addition, every successful decrement of a register produces an extra molecule of S_2. Thus, S_2 winds up being a large integer whose binary digits are a record of the times at which decrement instructions successfully decremented a register. This means that any run with the largest possible amount of S_2 must have always succeeded at decrementing whenever possible. In other words, it emulated the Register Machine in the correct, nonbroken way. Thus, we can be sure that in this run, S_1 has been computed correctly. Since the bulk of the time is consumed by doubling S_2, the correct run is also the longest possible sequence of reactions for the Stochastic Chemical Reaction Network, and the same remains true if we append a "clean up" routine to the end of the computation, that clears away the large quantity of S_2.

Thus primitive recursive functions are in perfect correspondence with questions of the form "How many molecules of S_1 will there be if a Stochastic Chemical Reaction Network produces the maximal amount of S_2?" or "How many molecules of S_1 will there be if the Stochastic Chemical Reaction Network takes the longest possible sequence of reactions?" So, although questions of possibility in Stochastic Chemical Reaction Networks are decidable, we have shown here that in some ways they have the full power of primitive recursive functions.

6 Ordered Program Models: Register Machines and Fractran

Because of the above and other decidability results, Petri nets, Stochastic Chemical Reaction Networks, and VASs are typically conceptually grouped with nonuniform models such as Boolean circuits, as was mentioned in Sect. 3. However, when provided with rate constants and evaluated in a probabilistic context, these models are, in fact, capable of uniform computation as well.

Bennett [18] proposed a method for simulating a TM that uses a DNA-like information carrying polymer as the equivalent of a TM tape, with an attached chemical group representing the head position and head state.[3] Reactions then occur on

[3] Recall that a TM consists of an infinite tape, and a head which can be in some finite number of internal states pointing to a specified position on the tape and capable of reading and writing from

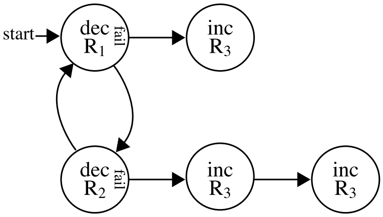


Fig. 7 A register machine comparing the value of register R_1 to R_2. If $R_1 \le R_2$, then it outputs 1 in register R_3. If $R_1 > R_2$ then it outputs 2 in register R_3. The start state is indicated with "start" and the halting states are those without outgoing arrows

this polymer that mimic the operation of the TM. The SCRN corresponding to this system has a different species for each polymer sequence, length, and the "head" chemical group and location. A single molecule then represents a single TM (tape and attached head), and reactions transform this molecule from one species to another. Thus, infinitely many species and infinitely many reactions are needed to represent Bennett's biomolecular TM simulation as a SCRN (although augmented combinatorial formalisms, which go beyond SCRNs, can represent Bennett's chemical TMs and other Turing-universal polymer-based chemical machines; see, for example, [21]).

Taking a different approach of storing and processing information, we show that SCRNs with a *finite* set of species and chemical reactions are Turing universal in probability—they can execute any computer program for any length of time, and produce the correct output with high probability. Thus, to increase the complexity of the computation performed by SCRNs, it is not necessary to add new reactions or species (as is the case when simulating circuits or using arbitrarily complex polymers). Our method, building on [16] as described in [14], involves showing that Register Machines (RMs) can be simulated by SCRNs for any length of time with little probability of error. Since it is known that any computer program can be compiled to a RM [13, 42], we can conclude that any computer program can be effectively compiled to a SCRN. Also since there exist specific RMs known to be Turing-universal (i.e., capable of simulating *any* computer program), we can conclude that there is a *Turing-universal* SCRN that can simulate any computer program with high probability.

Register Machines are a simplified, idealized abstraction of how computers work, with a CPU manipulating memory. Minsky showed in the 60s that Register Machines are capable of universal computation. A Register Machine is a machine that has a fixed number of *registers*, each of which can hold an arbitrary nonnegative integer. In addition to the registers, it has a fixed *program* which consists of a set of instructions. Every instruction is either an increment instruction, a decrement instruction, or a halt instruction. The increment and decrement instructions specify

and to the tape. Reading a bit of the tape allows the head to transition to different internal states and move left or right depending on the read bit; whether and which symbol is written depends of the state of the head.

which register is to be incremented or decremented, and they also specify which instruction should be executed next, after the increment or decrement. Decrement instructions, however, might not succeed with their intended decrement—if the register is 0, it cannot be decremented. In this case, the decrement instruction is said to *fail*, and each decrement instruction specifies an alternate next instruction to go to in the case that the decrement fails. The current *state* of a Register Machine is given by the values of the registers, along with which instruction is the next one to execute. A simple example of an RM comparing two integers is shown in Fig. 7. Register Machines are nice because of their simplicity, which makes it easy for other systems to simulate them.

One variant of Register Machines which in our experience is sometimes useful is what we call Broken Register Machines. These are the same as Register Machines except that decrement instructions are allowed to fail (nondeterministically) even if the register is nonzero. (If the register is zero, the instruction is of course forced to fail as before.) It is possible to show that Broken Register Machines turn out to be equivalent to Petri nets and VASs (and thus to Stochastic Chemical Reaction Networks as well), although the equivalence is not quite as direct as for the other systems. The nature of the equivalence between Broken Register Machines and Stochastic Chemical Reaction Networks, combined with the fact that Broken Register Machines only need to decide between two options at a time, enables one to show that in fact only two priority levels are necessary for a Stochastic Chemical Reaction Network to be universal.

Another model that turns out to be related is a lesser known model called Fractran [11], shown by Conway to be Turing universal. A Fractran program consists of an ordered list of rational numbers (see Fig. 1(d)). Execution is deterministic: starting with a positive integer n as input, we find the first fraction on the list that produces an integer when multiplied by n, and this product becomes the new number n'. This process is iterated forever unless it halts due to no fraction resulting in an integer. Conway showed that any Register Machine program can be converted directly into a Fractran program: representing every integer in fully factored form, $n = p_1^{a_1} \cdots p_m^{a_m}$, where p_i is the ith prime, the exponents $a_1 \dots a_k$ store the contents of the k registers, while other distinct primes p_h are each present iff the Register Machine is in state h. The denominator of each Fractran fraction conditions execution on being in state h and—if the operation is to decrement the register—on having a nonempty register. The numerator provides for increments and sets the new state. Since Register Machines are Turing-universal (although since they only allow increment and decrement operations, thus storing all state in unary, they entail exponential slowdowns compared to more reasonable computational models); it follows that Fractran is also universal.

Examination of Conway's construction illustrates the relation to VASs, Petri nets, and Stochastic Chemical Reaction Networks. Considering the integer n as the vector of exponents in its prime factorization, multiplication by a fraction corresponds to subtracting the exponents in the denominator and adding the exponents in the numerator, subject to the condition that no negative exponents are generated. This corresponds exactly to a Vector Addition System. Equivalently, each fraction can be interpreted as a chemical reaction: each species is represented by a unique prime number, and the denominator specifies the reactants and their stoichiometry, while the numerator specifies the products. (Catalytic reactions would correspond to nonreduced fractions, and can be avoided as shown in Fig. 1.) The determinism—and hence universal computational power—inherent in Fractran execution corresponds to there being a strict priority in which the various possible transitions are applied.

6.1 Computation in Stochastic Chemical Reaction Networks

If it were possible to prioritize the reactions in a Stochastic Chemical Reaction Network, then by analogy to the ordered fractions in Fractran, this would establish the Turing-universality of Stochastic Chemical Reaction Networks. (This result is also well known in the field of Petri nets, and our analysis of Register Machines shows that in fact only two distinct priority levels are necessary.)

By giving higher-priority reactions vastly faster rate constants k_α, we can approximate a priority list: almost surely, of all reactions for which all reactants are present in sufficient number, a reaction with a much faster rate will occur first. However, "almost surely" turns out not to be good enough for a couple of reasons. First, there is a nonzero probability of the slow reaction happening at each step, and thus probability of successful output falls exponentially with the number of steps. Second, the number of molecules of a given species can potentially exceed any bound, so the ordering of actual rates $\rho_\alpha(\mathcal{A})$ may eventually be different from the specified ordering of rate constants k_α. Especially in light of the decidability results mentioned above, it is not surprising that this naive approach to achieving Turing universality with Stochastic Chemical Reaction Networks fails.

If there were some way to increase rate constants over time, this could solve these problems, but of course, rate constants cannot change. Another way to promote one reaction over another would be to give the preferred reaction some extra time to occur before the competing reaction has a chance to occur. This approach turns out to be workable, and it is not too hard to set up some reactions that produce a signal after some delay, where the delay depends on a particular concentration. We refer to such a set of reactions as a *clock*. An important technical point is that since the entire computation will consist of an unknown number of steps, the probability of error at any given step must be decreasing so that the sum of all the probabilities can remain small regardless of how long the computation winds up taking. To address this issue, the clock can at each step increase the concentration that controls its delay, so that the delays are progressively longer, and thus the probabilities of error are progressively smaller. Fortunately, it turns out that a simple polynomial slowdown in overall computation time is all that is required for making the total probability of error (over the entire course of the arbitrarily long computation) be small.

In the following, we give a construction for simulating Register Machines with Stochastic Chemical Reaction Networks with only a polynomial slowdown, and we prove that successful output will occur with fixed probability $1 - \epsilon$ independent of the input and computation time. An initial number of "precision molecules" can be added to reach any desired level of ϵ. Thus, tolerating a fixed, but arbitrarily low, probability that computation will result in an error, Stochastic Chemical Reaction Networks become Turing universal. In consequence, the probabilistic variants of the reachability and producibility questions are undecidable.

The simulation is relatively simple to understand, but its performance is limited by the fact that it is simulating a Register Machine, which is exponentially slower than a Turing Machine (in the space used by the Turing Machine), due to its unary representation of information. Can Stochastic Chemical Reaction Networks do better than this? It turns out that they can. In Sect. 7, we discuss a more sophisticated algorithm that allows Stochastic Chemical Reaction Networks to directly polynomially simulate a Turing Machine.

Probability in SCRNs Is Undecidable

Theorem 3 *For all* $0 \leq \epsilon < 1/2$, *the following problem is undecidable*: *given a Stochastic Chemical Reaction Network* C, *a species* S, *and a starting state* $\mathcal{A}$, *determine*, *to within* ϵ, *the probability that* C *starting from* $\mathcal{A}$ *will produce at least one molecule of* S.

To prove this theorem, we will show how Stochastic Chemical Reaction Networks are capable of simulating Register Machines. First, we define the correspondence between instantaneous descriptions of Register Machines and states of Stochastic Chemical Reaction Networks that our construction attains. Then we show that determining whether a Register Machine ever reaches a particular instantaneous description is equivalent to ascertaining whether our Stochastic Chemical Reaction Network enters a set of states with sufficiently high probability.

Definition 1 An *instantaneous description ID* of a Register Machine M with t registers is a vector $(a, c_1, \ldots, c_t)$ where a is a state of M and $c_i \in \mathbb{N}$ represents the value of register i.

Definition 2 The reachability relation $ID \overset{M*}{\rightarrow} ID'$ is defined naturally. Namely, it is satisfied if M eventually reaches ID' starting from ID.

Definition 3 For two states $\mathcal{A}$ and $\mathcal{B}$ of a Stochastic Chemical Reaction Network C, we write $\mathcal{A} \overset{C}{\rightarrow} \mathcal{B}$ if there is a reaction that takes the system from $\mathcal{A}$ to $\mathcal{B}$. Let $\overset{C*}{\rightarrow}$ be the reflexive transitive closure of $\overset{C}{\rightarrow}$.

Instantaneous descriptions of a Register Machine map to sets of states of our Stochastic Chemical Reaction Network as follows.

Definition 4 For an instantaneous description $ID = (a, c_1, \ldots, c_t)$ of a Register Machine M let $\mathcal{M}(ID, n)$ be the state of a Stochastic Chemical Reaction Network that contains exactly:

- n molecules of species A,
- c_i molecules of R_i $\forall 1 \le i \le t$,
- 1 molecule of S_a,
- and 1 molecule of T, B, B', and B'' each.

Definition 5 Our Stochastic Chemical Reaction Networks will be said to *ϵ-follow* a Register Machine M if there is some n_0 such that for all instantaneous descriptions ID and ID' of M we have

(a) $ID \overset{M*}{\rightarrow} ID' \iff \Pr[\mathcal{M}(ID, n_0) \overset{C*}{\rightarrow} \mathcal{M}(ID', n) \text{ for some } n] > 1 - \epsilon$,
(b) $ID \overset{M*}{\nrightarrow} ID' \iff \Pr[\mathcal{M}(ID, n_0) \overset{C*}{\rightarrow} \mathcal{M}(ID', n) \text{ for some } n] < \epsilon$.

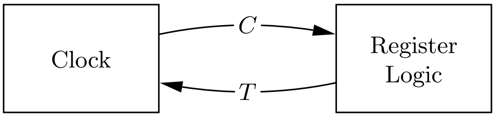
Theorem 4 *For any Register Machine M, and any $\epsilon > 0$, there is a Stochastic Chemical Reaction Network C that ϵ-follows M.*

In fact, slight modifications of our construction can show that all questions about whether a Stochastic Chemical Reaction Network "might do X" mentioned Sect. 5 becomes uncomputable in the probabilistic setting ("does X with probability $>\epsilon$").

Proof We construct a Stochastic Chemical Reaction Network to simulate the Register Machine, consisting of two components: a clock module and a register logic module (Fig. 8). The communication between the modules is established through two species, T and C, of which at most a single molecule is present. Whenever the clock releases the C, the register logic module can complete a step of the register machine (with the exception of the actual decrement of a decrement instruction), converting the C into a T in the process. The clock module then takes the T and, after a delay, releases another C to repeat the process. The delay imposed by the clock module makes it exceedingly likely that any decrement waiting to happen will occur before the next C is released. This effectively enforces the reaction order that is necessary for correct computation.

The register logic module has a single molecule of species S_a for every state a of the register machine. The number of molecules of species R_i stores the value of the register i. If the current register machine state a is an increment state, once the clock module releases the C, then the reaction $S_a + C \to S_b + R_i + T$ increments the ith register and transitions to the next state b. If the current state is a decrement state and the register i being read is empty, then the reaction $S_a + R_i \to S'_a$ is not possible, and once the clock module releases the C, the reaction $S_a + C \to S_c + T$ takes place and transitions to the state c indicating that the register is empty. If the register i is not empty (i.e., there is at least one molecule of R_i in solution), then the intent is that the reaction $S_a + R_i \to S'_a$ should decrement the register and capture S_a before the clock module next releases a C. (Otherwise, the reaction $S_a + C \to S_c + T$ could occur first, erroneously sending the register logic module into the state c, which is only supposed to happen if the register is empty.)

Thus, the only possible error that can occur in the register logic module is if $S_a + C \to S_c + T$ occurs before $S_a + R_i \to S'_a$ in a decrement step, when register


(a) The Clock and Register Logic Modules

(b) Clock

$$\begin{aligned}
&T + B \to T' + B\\
&T' + A \to T + A\\
&T' + B' \to T'' + B'\\
&T'' + A \to T' + A\\
&T'' + B'' \to C + B'' + A
\end{aligned}$$

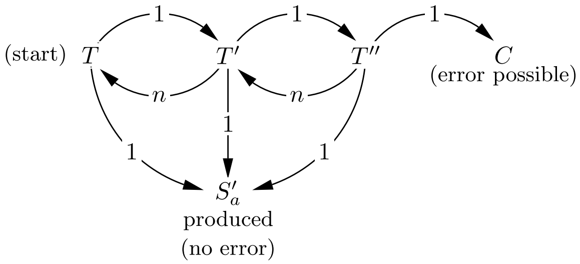
(c) Register Logic

In state a increment register i and go to state b	$\Rightarrow$	$S_a + C \;\to\; S_b + R_i + T$
In state a decrement register i and go to state b, or if the register is empty go to state c	$\Rightarrow$	$S_a + R_i \;\to\; S'_a$ $S'_a + C \;\to\; S_b + T$ $S_a + C \;\to\; S_c + T$

Fig. 8 Simulating a register machine. (**a**) The communication between the clock and the register logic modules is through single molecules of species C and T. (**b**) The clock module is responsible for producing a C molecule once every so often. The clock module is designed so that the length of time between receiving a T and producing a C slowly increases throughout the computation, thus slowing down the register logic module to help it avoid error. Specifically, the more A's there are, the longer the delay. The clock starts out with n_0 A's and one each of B, B', and B'' and T. Every clock cycle not only produces a C, but increases the number of A's by one. Thus, at the beginning of the kth cycle, there are $n = k + n_0$ molecules of A. The clock's operation is further analyzed in Fig. 9. (**c**) The register logic module simulates the register machine state transitions. The register logic module starts out with quantities of molecules of R_i indicating the starting value of register i, and a single molecule of species S_a where a is the start state of the register machine. Note that at all times the entire system contains at most a single molecule of any species other than the A and R_i species. All rate constants are 1 (The construction will work with any rate constants)

i is not empty. By delaying the release of the C, the clock module ensures that the probability of this happening is low. The delay increases from step to step sufficiently to guarantee that the union bound taken over all steps of the probability of error does not exceed ϵ.

Let us analyze the probability of error quantitatively. Suppose the current step is a decrement step and that the decremented register has value 1. This is the worst case scenario since if the register holds value greater than 1, the rate of the reaction $S_a + R_i \to S'_a$ is correspondingly faster, and if the step is an increment step or the register is zero, then no error can occur. Figure 9 illustrates the state diagram of the relevant process. All of the reactions in our Stochastic Chemical Reaction Network



$$\frac{d}{dt}\begin{bmatrix} s \\ s' \\ s'' \\ p \\ q \end{bmatrix} = \begin{bmatrix} -2 & n & 0 & 0 & 0 \\ 1 & -2-n & n & 0 & 0 \\ 0 & 1 & -2-n & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} s \\ s' \\ s'' \\ p \\ q \end{bmatrix}$$

Fig. 9 The state diagram for a single decrement operation when there are n A's and the register to be decremented holds the value 1, and the corresponding system of differential equations governing the instantaneous probabilities of being in a given state. The *numbers on the arrows* are the transition rates. The instantaneous probability of being in state T is s, in state T' is s', and in state T'' is s''. The instantaneous probability of being in the error-possible state is p and the probability of being in the no-error state is q

have the same rate constant of 1. Thus, all reactions with exactly one molecule of each reactant species in solution have the same reaction rate of 1. There are two reactions for which this single molecule condition is not true: $T' + A \to T + A$ and $T'' + A \to T' + A$, since there are many A's in solution. If there are n A's in solution, each of these two reactions has rate n. Now, we will bound the probability that the clock produces the C before the $S_a + R_i \to S'_a$ reaction occurs, which is a bound on the probability of error. The top 4 states in the diagram (Fig. 9) represent the 4 possible states of the clock: we either have a T, T', T'', or a C. A new cycle starts when the register logic module produces the T and this is the start state of the diagram. No matter what state the clock is in, the reaction $S_a + R_i \to S'_a$ can occur at rate 1 in the register logic module. Once this happens, no error is possible. On the diagram this is indicated by the bottom state (no error) which is a sink. On the other hand, if a C is produced first then an error is possible. This is indicated by the sink state C (error possible).

We compute the absorption probability of the error-possible state by solving the corresponding flow problem. Solving the system of differential equations in Fig. 9 for $\frac{dp}{dt}$ under the condition that $\frac{ds}{dt} = -1$, $\frac{ds'}{dt} = \frac{ds''}{dt} = 0$, we find that the absorption probability of the error-possible state is $p = \frac{1}{(n+2)^2+4}$. Thus, the probability of error for a step with n A's is bounded by $p = \frac{1}{(n+2)^2+4}$. In order to be sure that the probability that no error occurs during *any* point in the computation is larger than

$1 - \epsilon$, recall that n increases by one at each step, so we need

$$\sum_{n=n_0}^{\infty} \frac{1}{(n+2)^2+4} < \epsilon.$$

The terms in the above inequality are inversely quadratic in n, so if $n_0 = 1$ then the sum is finite (in fact, it is roughly 0.3354). This means that for any ϵ, we can choose an appropriate n_0, the initial number of A's, to make the above inequality true. □

How fast is the Register Machine simulation? Since each consecutive step is potentially delayed longer and longer, we need to be careful that the simulation is not slowed down too much. Indeed, it can be shown that the expected time to perform t steps of the Register Machine simulation requires at most $O(t^4)$ SCRN time [14].

6.2 Eliminating Dependency on the Number of Molecules Disables Universal Computation

If the rates of the possible reactions do not depend on the number of molecules then it can be shown that the system is incapable of universal computation. In particular, it will be predictable in the sense that the probability that at least one molecule of a given species is eventually produced can be computed to arbitrary precision. This precludes the previous output method of using an indicator species whose production with high or low probability indicates the outcome of the computation. Further, any other method of output that can be converted to this form of output cannot be universal either. This includes, for example, Stochastic Chemical Reaction Networks that enter a specific state with high or low probability to indicate the output. Specifically, the model we are considering here is the following: Suppose we are given a Stochastic Chemical Reaction Network with given constant rates for all the reactions, and an initial set of molecules. Then at each step, based solely on the reaction rates, a reaction is chosen. This reaction then occurs if the reactants for it are present. Such steps continue indefinitely.

The difference between this model and the standard stochastic one is that in the standard model, the reaction rate is obtained by combining a rate constant with the current concentrations as described in Sect. 2.1 (see (1)), while here for all reactions α and states $\mathcal{A}$, $\rho_\alpha(\mathcal{A}) = k_\alpha$ if all the reactants of α are present in $\mathcal{A}$ and 0 otherwise.

Theorem 5 *Let $\mathbb{S}$ be the infinite set of all states with at least one molecule of the indicator species present. Suppose for all reactions α and states $\mathcal{A}$, $\rho_\alpha(\mathcal{A}) = k_\alpha$ if all the reactants of α are present in $\mathcal{A}$ and* 0 *otherwise. Then there is an algorithm that given* $0 < \epsilon$ *and any starting state $\mathcal{A}$, computes* $\Pr[\mathcal{A} \overset{C*}{\to} \mathcal{B}$ *for some* $\mathcal{B} \in \mathbb{S}]$ *within ϵ.*

Let $\tilde{\mathbb{S}}$ be the (probably infinite) set of states from which no state in $\mathbb{S}$ is reachable, and let $\mathbb{R}$ be the set of states outside $\mathbb{S}$ from which it is possible to reach $\mathbb{S}$. (Note that given any state, the question of whether it is possible to reach some state in $\mathbb{S}$ is computable, as shown in Sect. 5.2.) Note also that there is a bound b such that for any state $\mathcal{A} \in \mathbb{R}$, the length of the shortest sequence of reactions leading from $\mathcal{A}$ into $\mathbb{S}$ is at most b. This means that there is some constant p_0 such that for any state $r \in \mathbb{R}$, the probability of entering $\mathbb{S}$ within b steps is at least p_0. Thus, the probability of remaining in $\mathbb{R}$ must decay at least exponentially.

This implies that the probability that the system will eventually enter $\mathbb{S}$ or $\tilde{\mathbb{S}}$ is 1, and so simply by computing the probabilities of the state tree for $\mathbb{R}$ far enough, one can compute the probability of entering $\mathbb{S}$ to arbitrary precision.

7 Efficiency of Computation by Stochastic Chemical Reaction Networks

Section 6 showed that universal computation (in probability) can be performed by SCRNs, but our construction inherits the ridiculous inefficiency of Register Machines, which in general require exponential time to simulate Turing machine computations. Is it possible to use the power of chemistry to perform computations more quickly and efficiently?

Trivial ways to speed up a chemical computer involve changing environmental conditions such as increasing the temperature or the effective concentration (molecular count per unit volume). In order to discuss the "intrinsic speed" of the computer we are proposing, we fix the temperature, as well as the maximum concentration (recall that the volume scales dynamically with the molecular count in our model, see Sect. 2.1). Then the performance of the chemical computer will be gauged asymptotically as the size of the "tape" as well as the number of simulation steps increases. With improved chemical programming, it turns out that compared to the abstract Turing Machine, its chemical implementation incurs only a polynomial slowdown. The volume required, however, inevitably grows exponentially with the size of the tape of the Turing machine being simulated. This is impossible to avoid since fixing the number of species there is simply no way to store information in a form other than unary.

7.1 Stochastic Chemical Reaction Networks Can Efficiently Simulate Turing Machines

Theorem 6 *For any $0 < \epsilon < 1/2$ and any Turing Machine M, we can make a Stochastic Chemical Reaction Network that, starting with n molecules of species S_{in} (and some number of molecules of other species, dependent on ϵ but not n), will with high probability ($> 1 - \epsilon$) result in fast (expected time polynomial in the running time of $M(n)$) and accurate (eventually produces S_{halt} iff $M(n)$ halts) computation.*

Of course, by having different output species, the same output method can be used to indicate a 0/1 output or in fact an integer output encoded in unary.

The overall idea to achieve this fast Turing Machine simulation is to adopt the Register Machine simulation, but allow more sophisticated operations on the registers [14, 17]. If in addition to incrementing and decrementing a register, we could double or halve the value of any register and check the remainder in a constant number of clock cycles of the Register Machine simulation, then we could simulate a Turing Machine in linear time. To do this, we can represent the accessed portion of the Turing Machine head tape as two integers whose binary representation encodes the tape to the left and to the right of the head respectively, such that their least significant bits represent the contents of the tape cells immediately to the left and right of the head. Since reading and removing the least significant bit corresponds to halving and checking for remainder, and writing a new least significant bit corresponds to doubling and possibly adding one, a single Turing Machine step can be performed in a small constant number of these enhanced Register Machine cycles. With registers represented in unary molecular counts, halving would correspond to a reaction scheme that takes two molecules of one species and produces one molecule of another species, while doubling would correspond to a reaction scheme that takes one molecule of one species and produces two molecules of another species. Conversion of all molecules must take place quickly and exactly—if a single molecule is not converted, the least significant bit(s) will be in error. Unfortunately, we will see that halving a register quickly is rather difficult, but fortunately we will be able to avoid the halving operation.

In the following section, we provide a construction similar to (but not identical to) that in of Ref. [14] and give an informal explanation of how it works.

The Exploding Computer

To perform computation quickly using molecular counts, we have a number of challenges. The primary difficulty is that if every molecule matters for a decision, then the presence or absence of a single molecule (for example, the parity of a register) must be communicated to all other molecules in the system that are affected by the decision. But in our model of well-mixed chemistry, communication happens only by chance collisions between molecules—and rare species will therefore interact rarely.

The main technique that allows large numbers of molecules to be processed quickly is for the state changes to occur via explosive chain reactions. These "explosions" do not necessarily increase the number of molecules; they might simply change the molecules from one form to another. Each explosion starts as an exponentially growing chain reaction, until the amount of reactive material starts to get used up, at which point it finishes with exponential decay of the reactive material. Thus, an exponential amount of reactive material can be processed in a given amount of time, as shown in Fig. 10. By changing the number of product molecules in the reaction, the explosion scheme can be easily transformed into a means to quickly and exactly double the number molecules present.

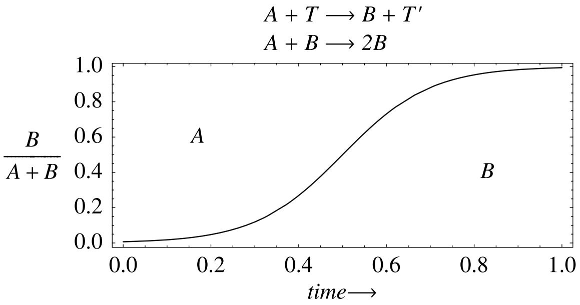


Fig. 10 The time course of a reactant-limited chain reaction explosion, shown as a conversion from a species A to a species B, initiated by a trigger T. If at the beginning, a fraction p of all molecules in the system are A molecules, then the number of converted molecules grows like e^{kpt}, where t is time and k is the rate constant of the reaction catalyzed by B. For the first half of the chain reaction, at least $p/2$ of the molecules are A, and so the expected time for the first half to complete is under $(2/kp)\log|A|/2$. For the second half of the chain reaction, over $p/2$ of the molecules are B, so each molecule of A gets transformed at a rate above $kp/2$, so the quantity of A decreases faster than $e^{kpt/2}$, and the expected time for the second half to complete is under $(2/kp)\log|A|/2$. Thus, the total time needed for the explosion to finish is on the order of $\log|A|/p$

The naive implementation of having a halving reaction akin to $2M \to M'$ is slow for the same reasons as shown in Fig. 11.

If we are to avoid having to halve the value of a register, we must have an architecture for computation that only requires doubling when reading and writing bits to and from memory. To do this, we use the digits of the memory integer as a queue of binary digits. We can read and remove the most significant digits (as we will show), we can shift the digits over (by doubling, or multiplication by a constant), and we can write new low order digits by simply producing a few extra molecules. Thus freshly written digits get exponentially amplified step by step until they are the largest contribution to the overall magnitude, at which point the system is able to detect their value.

Before proceeding, we should make sure that these operations are sufficient for efficient simulation of Turing Machines. To see this, here is how to convert a Turing Machine into a program that uses only queues. First, consider a Turing Machine that uses a fixed amount of space on a binary tape. This finite tape is encoded in the queue using three bits per cell, one bit for the cell's value, an another bit to indicate the cell that the Turing Machine head is reading, and the third bit to indicate the first and last cells. Note that after one time step, the tape will be changed only in the three cells around where the head is reading: the center cell might have a new value, and either of the adjacent cells might need to be marked to indicate that this is where the Turing machine is now reading. To implement a single time step of the Turing Machine, the new queue program will make a pass through the whole queue, keeping the most recent three cells memorized at every step. Each step consists of spitting out the correct new value for the oldest of the three cells and then reading in one more cell. The queue program knows when it has completed a pass, thanks to the third bit in each cell. Thus, to simulate Turing Machines that use arbitrary amounts

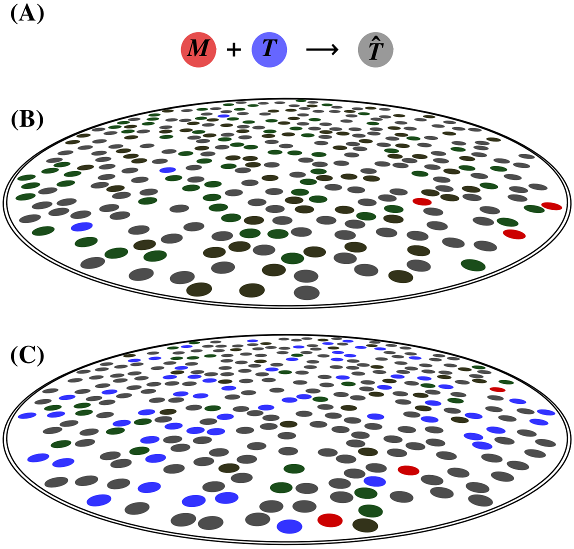


Fig. 11 (**A**) To read the most significant digit of M, we compare M (*red*) to a known threshold quantity T (*blue*). This is done by a simple reaction, $M + T \rightarrow \hat{T}$. The goal is that after all the M and T molecules have reacted to form $\hat{T}$, only the species in excess, either M or T, will remain. However, the time it takes for this reaction to complete will depend on the amounts of M and T. (**B**) If M and T are present in nearly equal quantities, then toward the end of the reaction only a few molecules of M and a few molecules of T will remain, and we will need to wait a very long time to be confident that these last few molecules have found and reacted with each other, especially when the volume is large. (**C**) If either M or T is significantly in excess of the other, with the excess being at least some constant fraction of the entire volume, then toward the end of the reaction, as one of the species approaches extinction, its rate of decay will remain exponential, and the reaction will fully finish quickly, regardless of volume

of tape, the queue program can simply output some blank cells at the beginning and end of each pass. Overall, the queue program is slower than the original Turing Machine, but only by a linear factor—if the original machine took $O(t^{16})$ steps, the queue program will take $O(t^{17})$ steps. (Slightly more efficient implementations are possible [14, 43].)

With this queue architecture, the challenge of detecting a single molecule is avoided; all we need is a scheme that allows the system to be able to read the high order memory digit quickly and accurately. This can be achieved by storing integers in the memory using a Cantor-set structure. To be able to read the most significant digit of the memory integer, we need to compare the memory integer to a threshold value, and as shown in Fig. 11 it is important that the memory integer not be too close to the threshold value. That is, the magnitude of the memory integer, regardless of the contents of the memory, should be separated from the threshold value it is being compared to by a gap that is at least some fixed fraction of the threshold value itself, so that the comparison will always complete in logarithmic time. The way we will satisfy this requirement is by using numbers which, in base 3, use only the digits 1 and 2. These numbers have a Cantor-set structure. Thus, the highest possible $k+1$ digit number starting with 1, namely $2 \cdot 3^k - 1 = 1222\ldots2_3$, and the lowest possible $k+1$ digit number starting with 2, namely $2.5 \cdot 3^k - 0.5 = 2111\ldots1_3$, are separated by an interval that is proportional in size to the numbers themselves, making the leading digit easily detectable.

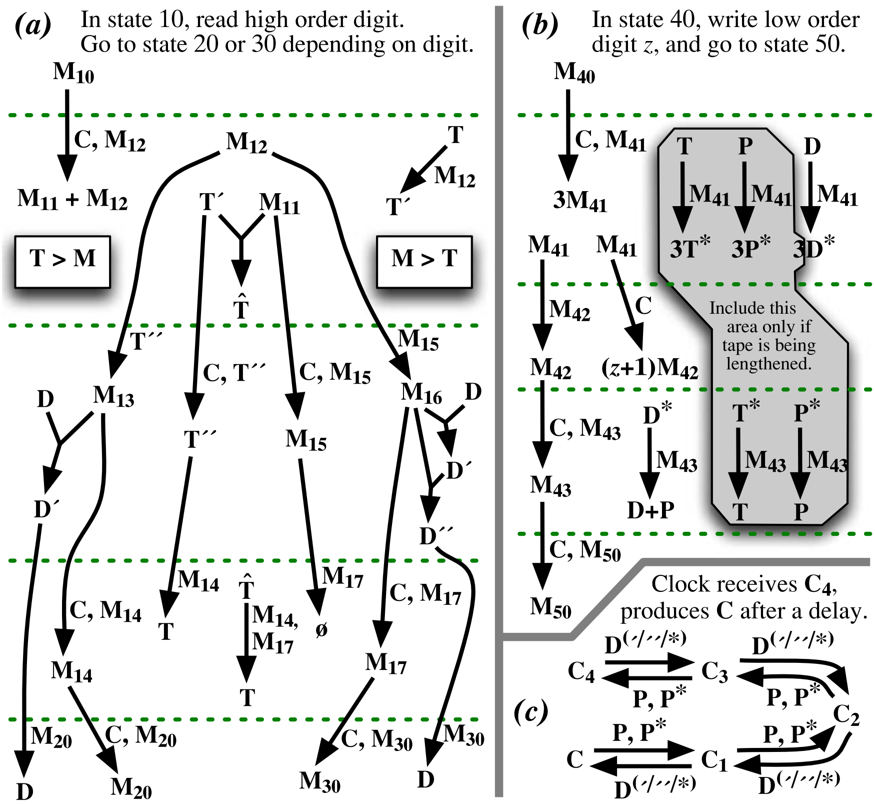
The system can write a low order digit into the memory by simply having just a single molecule present of the species responsible for writing this digit.

We have discussed the representation of the queue (i.e., encoded Turing Machine tape) as the molecular counts of a register species—but how do we represent which step of the program is currently being executed? Since the program contains a finite number of states, it is possible to assign a distinct molecular species for each program state. In fact, we combine both representations into one: if the queue program is in state 10 with the integer M in the queue, then we will have M copies of the molecular species $\mathbf{M}_{10}$. The molecular count encodes the queue, and the molecular species encodes the program step being executed. Thus, to push a "1" onto the bottom of the queue and transition to state 20, we perform an "explosion" that converts the M copies of $\mathbf{M}_{10}$ into $2M$ copies of $\mathbf{M}_{20}$ and then produce one more $\mathbf{M}_{20}$. Effectively, the chemical system is designed as a sequence of conditional explosive reactions, each of which changes the "color" of the solution.

As in the Register Machine simulation of the previous section, the system can have a very low chance of ever making a mistake, even though there is some chance of error at every step, as a result of having the speed of the system regulated by a clock that slows down over time. Since each step is more likely to succeed when it is given more time to complete its work, the slowing clock makes successive steps more and more likely to succeed. Intuitively, if it makes it through the first several steps without error, later steps become progressively easier to do correctly, so it's quite likely to just go forever without making any mistakes. Mathematically, the probability of error decreases at each step fast enough that the sum of all the probabilities of error converges to a small value. This value can be made arbitrarily small, regardless of the computation to be done by the Turing machine, simply by starting the system with a greater number of precision molecules (**P** in Fig. 12).

The molecular species and reactions for the simulation are shown in Fig. 12. Four clock cycles are required for each step so that the various suboperations do not interfere with each other. At each step, the clock molecule **C** triggers an explosive chain reaction, and the output of that chain reaction is used to catalyze all the other reactions occurring at that step (with the exception of comparisons and subtractions, which have no catalysts).

When reading a bit of memory, the reactions compare the memory M with the threshold T, as discussed in Fig. 11. After the first clock cycle, which performs the comparison, only half of the remaining reactions will occur, according to the result of the comparison. If $T > M$, then the leading digit of M was "1," and only the reactions on the left side will occur. If $M > T$, then the leading digit of M was "2," and only the reactions on the right side will occur. During the second clock cycle, D is subtracted from M either once or twice, so as to zero out the leading digit (which was just read). During the third cycle, the threshold T is restored, and the fourth cycle cleans up D and M.



(a) In state 10, read high order digit. Go to state 20 or 30 depending on digit.

(b) In state 40, write low order digit *z*, and go to state 50.

Clock receives $\mathbf{C_4}$, produces $\mathbf{C}$ after a delay.

(c)

Fig. 12 Reactions for a chemical system emulating a Turing Machine that has been converted into a queue program. The *horizontal dashed lines* represent clock cycles. This example uses the convention that the program states of the queue program are multiples of 10, while substates required for the chemical implementation modify the rightmost digit. Species listed to the side of an arrow are catalysts for the reaction: At least one of them must be present for the reaction to occur. In a slight abuse of this notation, when the clock signal **C** is used as a "catalyst," it is actually being converted to $\mathbf{C}_4$. (So, it is not really being used catalytically—but this notation makes the diagram cleaner.) $\mathbf{M}_i$ molecules store the memory integer and encode the program state i. **T** molecules (of various sorts) store the comparison threshold. **D** molecules store a single '1' in the most significant digit. There is more **D** than **T**. **P** molecules store the current precision of the system. **C** is the clock signal. There is exactly one **C** molecule at any time. (**a**) Reading a bit of memory. (**b**) Writing a bit of memory. (**c**) Operation of the clock. Any **D** species (**D**, **D**′, **D**″, or **D***) can serve as a catalyst for the conversion of the **C** in each step

Every read operation must be followed by a write operation, so that the tape does not shrink. Extra write operations are allowed, so the tape can grow, but states corresponding to such extra operations must include the reactions in the gray region. The first clock cycle multiplies M by 3, and if the tape is growing, then T, P, and D also get multiplied by 3. The second clock cycle writes the new digit of M. The third cycle cleans up D, T, and P, and it also adds D to P. This way, the precision P is always a multiple of D, and the multiple grows by one with each write operation, so the precision of the simulation increases at every step. The fourth cycle cleans up M.

The clock molecule is used to trigger the advance from one stage of the reaction to the next. Therefore, when $\mathbf{C}$ is used as a "catalyst," it is actually transformed into a $\mathbf{C}_4$, so that it cannot trigger the advance to the following stage until some time has passed. Effectively, the clock slowly lets the $\mathbf{C}_4$ become a $\mathbf{C}$ again. To become a $\mathbf{C}$, it has to work its way down, but since P is greater than D by a growing factor, the process of the $\mathbf{C}_4$ becoming a $\mathbf{C}$ becomes slower and slower as time goes on. This slowing of the clock is what makes the whole system become more and more reliable over time.

A detailed construction based on the same principles is presented in [14], with an analysis of its error probability and running time. A more efficient construction can be implemented based on the work of Angluin et al. in the distributed computing literature [17] (see [14, 44]). Simulating t_{TM} steps of a Turing Machine using s_{TM} space can be done in $\mathrm{polylog}(m) \cdot t_{\mathrm{TM}}$ time where $m = O(2^{s_{\mathrm{TM}}})$ is the total maximum molecular count encountered.

7.2 Turing Machines Can Efficiently Simulate Robust Stochastic Chemical Reaction Networks

We have seen that enforcing reaction order, even probabilistically, is enough to achieve Turing-universality. However, our simulation of Turing Machines (and Register Machines) by Stochastic Chemical Reaction Networks, uses reaction propensities rather weakly: while it was essential that one reaction propensity is higher than another, and increases over time, the exact value of reaction propensities are not used in the computation. Intuitively, we can say that a Stochastic Chemical Reaction Network behaves "robustly" if its behavior does not depend crucially upon getting the reaction propensities exactly right. Formal definitions can be found in [44], as well as the proof that the Turing Machine embedding based on [17] is robust.

Such robust chemical systems form an interesting class, whose computational power can be almost exactly captured, bounding above and below the maximum amount of computation such systems can perform in a unit of time, compared to a Turing Machine. Although on the order of m reactions can occur per unit time, where m is the total number of molecules present, the actual amount of computation is at most $\mathrm{polylog}(m)$ Turing Machine steps.

While fast Turing Machine embeddings in robust Stochastic Chemical Reaction Networks show a lower bound on their computational power, how can we show that they are not capable of performing more computation per unit time? The main idea of the argument is that robust chemical systems are easy to simulate by a Turing Machine. Intuitively, since robust chemical systems are robust to perturbations in reaction rates, they permit some sloppiness when trying to predict their behavior. Then, since it is widely believed that there is no universal way of speeding up Turing

Machines, it should not be possible to speed up arbitrary Turing Machines by embedding them in an chemical system and simulating the system. With some caveats related to real-number arithmetic, for robust systems, the problem of estimating the probability of being in a given state at a given time t can be solved in $\text{polylog}(m) \cdot t$ computation time on a Turing Machine, where m is the maximum molecular count encountered. This implies that, loosely stated, for robust Stochastic Chemical Reaction Networks, it is neither possible to embed more than $\text{polylog}(m)$ computation time per chemical unit time, nor is it possible to simulate the Stochastic Chemical Reaction Network using less than $\text{polylog}(m)$ computation time per chemical unit time [44].

It should be emphasized that the correspondence between Turing Machines and Stochastic Chemical Reaction Networks is surprisingly tight. One can simulate the other with surprisingly little loss of efficiency (especially for programs using little memory where $\text{polylog}(m)$ for $m = O(2^{s_{\text{TM}}})$ is small compared to t_{TM}).

8 Concluding Remarks

The power of different systems to do computation can vary greatly. It has previously been assumed that systems such as genetic regulatory networks and chemical reaction networks are much weaker than the gold standard computational systems such as Turing Machines. On the other hand, we have become accustomed to proofs that even some of the simplest systems are capable of universal computation [45, 46], meaning that they are in some senses equivalent in power to Turing Machines, and thus predicting their eventual behavior is impossible even in theory. Chemical reaction networks have been shown to be universal when combined with polymer memory [18] or membrane-separated compartmental memory [22], but researchers have previously assumed that on their own, a finite number of species in a well-mixed medium can only perform bounded computations [22, 24].

In contrast with this historical intuition, here we have shown that in fact such "plain" chemical reaction networks can indeed perform unbounded computation, using the concentration (number of molecules) of each species as the storage medium. We went on to pinpoint both the power and the weakness of chemical reaction network computation by showing that it is just as fast as Turing Machine computation, but that it requires exponentially more space.

This universality of chemical reaction networks turns out to derive from their probabilistic nature. If the possible reactions in a chemical system could be prioritized, so that the next reaction at each step is always the one with highest priority, then universal behavior is easily attainable (along the lines of [12]), but of course chemistry does not behave in this way. However, since the reaction rates in a chemical system are influenced by the concentrations, they are somewhat under the control of the system itself, and as we have shown, this weak form of prioritization turns out to be enough to let the system perform universal computation with high probability of success.

If we require that the chemical system be guaranteed to give the right answer without fail, then the system is effectively deprived of the opportunity to use its reaction rates, since they only influence what is likely to happen, not what is guaranteed to happen. Indeed, in this situation, the system is incapable of universal computation. Thus, the stochastic reaction rate foundation turns out to be the source of the computational power of chemical reaction networks.

Open questions, along the lines of the results we have given, include:

(1) Are continuous Stochastic Chemical Reaction Networks (using mass action kinetics) Turing universal?[4]
(2) Can one have a universal Stochastic Chemical Reaction Network which has constant probabilities (that do not depend on concentrations) for all reactions except one, with the remaining reaction having a decaying probability that depends on time (but not on concentrations)?
(3) Can Stochastic Chemical Reaction Networks that have reversible reactions be universal?
(4) What is the power if one wishes to guarantee that all paths in a Stochastic Chemical Reaction Network lead to same result (confluent computation)? Are we limited to Boolean logic circuits, or can we do some sort of uniform computation?
(5) Can a more efficient Turing Machine simulation be embedded in a nonrobust Stochastic Chemical Reaction Network than a robust one?

Acknowledgements The research was supported in part by the "Alpha Project" at the Center for Genomic Experimentation and Computation, an NIH Center of Excellence (grant number P50 HG02370), as well as NSF Grants No. 0523761 and 0728703 to EW and NIMH Training Grant MH19138-15.

References

1. Gillespie DT (1977) Exact stochastic simulation of coupled chemical reactions. J Phys Chem 81:2340–2361
2. Arkin AP, Ross J, McAdams HH (1998) Stochastic kinetic analysis of a developmental pathway bifurcation in phage-l Escherichia coli. Genetics 149:1633–1648
3. Gibson M, Bruck J (2000) Efficient exact stochastic simulation of chemical systems with many species and many channels. J Phys Chem A 104:1876–1889
4. Guptasarma P (1995) Does replication-induced transcription regulate synthesis of the myriad low copy number proteins of Escherichia coli? Bioessays 17:987–997
5. Levin B (1999) Genes VII. Oxford University Press, Oxford
6. McAdams HH, Arkin AP (1997) Stochastic mechanisms in gene expression. Proc Natl Acad Sci 94:814–819
7. Elowitz MB, Levine AJ, Siggia ED, Swain PS (2002) Stochastic gene expression in a single cell. Science 297:1183–1185
8. Suel GM, Garcia-Ojalvo J, Liberman LM, Elowitz MB (2006) An excitable gene regulatory circuit induces transient cellular differentiation. Nature 440:545–550
9. Esparza J, Nielsen M (1994) Decidability issues for Petri nets—a survey. J Inf Process Cybern 3:143–160

[4] Eric Stansifer (personal communication) seems to have answered this question in the affirmative.

10. Karp RM, Miller RE (1969) Parallel program schemata. J Comput Syst Sci 3(4):147–195
11. Conway JH (1972) Unpredictable iterations. In: Proceedings of the 1972 number theory conference. University of Colorado, Boulder, pp 49–52
12. Conway JH (1987) Fractran: a simple universal programming language for arithmetic. Springer, New York, chap 2, pp 4–26
13. Minsky M (1967) Computation: finite and infinite machines. Prentice–Hall, New Jersey
14. Soloveichik D, Cook M, Winfree E, Bruck J (2008) Computation with finite stochastic chemical reaction networks. Nat Comput. doi:10.1007/s11047-008-9067-y
15. Zavattaro G, Cardelli L (2008) Termination problems in chemical kinetics. In: van Breugel F, Chechik M (eds) CONCUR. Lecture notes in computer science, vol 5201. Springer, Berlin, pp 477–491
16. Liekens AML, Fernando CT (2006) Turing complete catalytic particle computers. In: Proceedings of unconventional computing conference, York
17. Angluin D, Aspnes J, Eisenstat D (2006) Fast computation by population protocols with a leader. Technical Report YALEU/DCS/TR-1358, Yale University Department of Computer Science. Extended abstract to appear, DISC 2006
18. Bennett CH (1982) The thermodynamics of computation—a review. Int J Theor Phys 21(12):905–939
19. Păun G (1995) On the power of the splicing operation. Int J Comput Math 59:27–35
20. Kurtz SA, Mahaney SR, Royer JS, Simon J (1997) Biological computing. In: Hemaspaandra LA, Selman AL (eds) Complexity theory retrospective II. Springer, Berlin, pp 179–195
21. Cardelli L, Zavattaro G (2008) On the computational power of biochemistry. In: Horimoto K, Regensburger G, Rosenkranz M, Yoshida H (eds) AB. Lecture notes in computer science, vol 5147. Springer, Berlin, pp 65–80
22. Berry G, Boudol G (1990) The chemical abstract machine. In Proceedings of the 17th ACM SIGPLAN–SIGACT annual symposium on principles of programming languages, pp 81–94
23. Păun G, Rozenberg G (2002) A guide to membrane computing. Theor Comput Sci 287:73–100
24. Magnasco MO (1997) Chemical kinetics is Turing universal. Phys Rev Lett 78:1190–1193
25. Hjelmfelt A, Weinberger ED, Ross J (1991) Chemical implementation of neural networks and Turing machines. Proc Natl Acad Sci 88:10983–10987
26. Petri CA (1962) Kommunikation mit Automaten. Technical Report Schriften des IMM No 2. Institut für Instrumentelle Mathematik, Bonn
27. Goss PJE, Peccoud J (1998) Quantitative modeling of stochastic systems in molecular biology by using stochastic Petri nets. Proc Natl Acad Sci USA 95:6750–6755
28. Mayr EW (1981) Persistence of vector replacement systems is decidable. Acta Inform 15:309–318
29. Sacerdote GS, Tenney RL (1977) The decidability of the reachability problem for vector addition systems (preliminary version). In: 9th annual symposium on theory of computing, Boulder, pp 61–76
30. Post EL (1941) On the two-valued iterative systems of mathematical logic. Princeton University Press, New Jersey
31. Cook M (2005) Networks of relations. PhD thesis, California Institute of Technology
32. Rosier LE, Yen H-C (1986) A multiparameter analysis of the boundedness problem for vector addition systems. J Comput Syst Sci 32:105–135
33. Skolem T (1923) Begründung der elementaren Arithmetik durch die rekurrierende Denkweise ohne anwendung scheinbarer Veränderlichen mit unendlichem Ausdehnungsbereich. Videnskapsselskapets Skrifter. 1. Matematisk-naturvidenskabelig Klasse, 6
34. Gödel K (1931) Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. Monatschefte Math Phys 38:173–198
35. Turing A (1936–1937) On computable numbers, with and application to the Entscheidungsproblem. Proc Lond Math Soc 42(2):230–265
36. Ackermann W (1928) Zum hilbertschen Aufbau der reellen Zahlen. Math Ann 99:118–133
37. Herbrand J (1931) Sur la non-contradiction de l'arithmétique. J Reine Angew Math 166:1–8

38. Gödel K (1934) On undecidable propositions of formal mathematical systems. In: Davis M (ed) The undecidable. Springer, Berlin, pp 39–74. Lecture notes taken by Kleene and Rosser at Princeton
39. Church A (1936) An unsolvable problem of elementary number theory. Am J Math 58:345–363
40. Péter R (1951) Rekursive funktionen. Akadémiai Kiadó, Budapest
41. Gale D (1974) A curious nim-type game. Am Math Mon 81:876–879
42. Minsky ML (1961) Recursive unsolvability of Post's problem of 'tag' and other topics in theory of Turing machines. Ann Math 74:437–455
43. Neary T, Woods D (2005) A small fast universal Turing machine. Technical Report NUIM-CS-2005-TR-12, Dept. of Computer Science, NUI Maynooth
44. Soloveichik D (2008) Robust stochastic chemical reaction networks and bounded tau-leaping. arXiv:0803.1030v1
45. Cook M (2004) Universality in elementary cellular automata. Complex Syst 15:1–40
46. Cook M, Rothemund PWK (2004) Self-assembled circuit patterns. In: Winfree E (ed) DNA computing 9, vol 2943. Springer, Berlin, pp 91–107

Log-gain Principles for Metabolic P Systems

Vincenzo Manca

Abstract Metabolic P systems, shortly MP systems, are a special class of P systems, introduced for expressing biological metabolism. Their dynamics are computed by *metabolic algorithms* which transform populations of objects according to a *mass partition* principle, based on suitable generalizations of chemical laws. In this paper, the basic principles of MP systems are formulated for introducing the *Log-gain* principles, and it is shown how to use them for constructing MP models from experimental data of given metabolic processes.

1 Introduction

Metabolism is the basis of the biomolecular processes of life. In its abstract and simplest setting, a metabolic system is constituted by a *reactor* containing a population of (bio)molecules of some given types, and communicating with an environment from/to which it gets/expels matter and/or energy. In this reactor, some reactions are active which transform molecules into other kinds of molecules, according to some stoichiometric patterns. These reactions transform matter by satisfying some chemical principles which can be formulated in very general terms: (i) molecules participate to reactions according to fixed ratios expressed by integer coefficients, (ii) matter cannot be created or destroyed, namely reactions transform matter of one kind into matter of other kinds, and (iii) each reaction needs or releases some energy, in order to be performed. A state of such a metabolic system is essentially described by the amount of molecules of each type which are inside it. This means that, if $x_1, x_2, \dots, x_n$ are respectively the quantities of the different types of molecules, and if we observe the system at steps $0, 1, 2, \dots,$ then its evolution turns out to be a function from natural numbers (the steps, separated by a suitable time interval) to vectors of dimension n (the amounts of substances, expressed, as rational numbers, in terms of some population unit).

V. Manca (✉)
Department of Computer Science, Strada Le Grazie, 15-37134 Verona, Italy
e-mail: vincenzo.manca@univr.it

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_28, © Springer-Verlag Berlin Heidelberg 2009

The classical approach of dynamical systems describes this kind of evolution by means of Ordinary Differential Equations (ODE) of the following kind:

$$\begin{aligned}
\frac{dx_1}{dt} &= f_1(x_1, x_2, \dots, x_n),\\
\frac{dx_2}{dt} &= f_2(x_1, x_2, \dots, x_n),\\
&\vdots\\
\frac{dx_n}{dt} &= f_n(x_1, x_2, \dots, x_n)
\end{aligned} \tag{1}$$

they are called *autonomous* ODE because the time variable t does not occur explicitly as argument of the functions at the right member (see [46] for classical analyses of biological metabolism). According to Peano–Picard's theorem, if the functions $f_1, f_2, \dots, f_n$ are of class $\mathbf{C}^1$ (continuously differentiable functions), then the relative Cauchy's problem is solvable in a suitable interval: there exist n functions $x_1 = g_1(t)$, $x_2 = g_2(t), \dots, x_n = g_n(t)$ that are solutions of the system and satisfy some given initial conditions [24]. The principle underlying ODE models is the *mass action principle*, according to which the infinitesimal variation of a substance produced by a reaction, is proportional to the product of the reactant masses instantaneously available. One of the main problems of differential models is the exact determination of the differential equations which govern the phenomenon under investigation. In fact, the infinitesimal nature of quantities involved in the equations require a knowledge of the microscopic molecular kinetic, and when the system is very complex and reactions are not completely understood, the numerical data are very difficult or even impossible to be evaluated. The same problem occurs even in stochastic simulations of (bio)chemical reactions, which are based on Gillespie's approach [21]. Therefore, it would be very important to formulate the dynamical description of systems in a way that overcomes this intrinsic limitation.

Some discrete models of (bio)chemical processes were introduced in [26, 35], where logical formulae, on strings inside membranes, expressed even complex dynamics. Though, this framework resulted too complex for expressing real biomolecular processes. P systems were then introduced in [37] as a new computation model, inspired by biology [10, 38, 47], into the mainstream of the formal language theory. They are essentially based on *multi-sets* rewriting and *membranes*. Applications of this perspective to biological modeling were developed in [3, 18, 20, 41–45]. A new discrete perspective in dynamical systems, based on P systems, was introduced in [1] and extended in [34, 39, 40]. A general discrete mathematical theory of reaction systems was developed in a series of papers [12–14] where their properties are mainly studied in the perspective of computation models. In this paper, we elaborate a new approach, introduced in [33] and then widely developed, on different lines, in [5–7, 11, 15–17, 27–32], which is focused on the notion of *Metabolic* P systems, shortly MP systems. In particular, we show the possibility of deducing an MP model, for a given metabolic process, from a suitable macroscopic observation of its behavior along a certain number of steps. MP systems are a special type of P systems which were proven to effectively model the dynamics of several biological processes, among them: the Belousov–Zhabotinsky reaction (Brusselator), the Lotka–Volterra dynamics, the SIR (Susceptible-Infected-Recovered epidemic) [4, 5], the leukocyte selective recruitment in the immunological response [5], the Protein Kinase C activation [7], circadian rhythms [15], mitotic cycles [19, 32], and a Pseudomonas quorum sensing process [9].[1]

2 The Mass Partition Principle

The new perspective introduced by MP systems can be synthesized by a new principle which replaces the mass action principle. We call it the *mass partition principle*. According to it, the system is observed along a discrete sequence of steps, and at each step, all the matter of a kind of substance, consumed in the time interval between two consecutive steps, is partitioned among all the reactions which need it to generate their products. If we are able to determine the amount of reactants that each reaction takes in that step, according to the stoichiometry of the reactions, which we assume to know, we can perfectly establish the amount of substances consumed and produced between two steps, therefore, whole the dynamics can be discovered. In this sense, another reading of MP systems could be just “Mass Partition” systems.

MP system reactions act on object populations, rather than on single objects (as P system rules do). Moreover, their dynamics is deterministic at population level, whereas nothing can be said about the dynamical evolution of single objects. This situation resembles what happens in the macroscopic gas laws, which specify deterministic relationships among pressure, volume and temperature measures, but do not cope with the mechanical behavior of single molecules.

As an instance, consider a metabolic system with four kinds of substances a, b, c, d and the reactions, (2) where u_1, u_2, u_3 at the right hand, are the number of elements (with respect to a population unit) which are consumed/produced by the reactions, for each occurrence of the involved substances, when the system goes from a state to the next one of its evolution:

$$\begin{aligned} r_1 &: aa \rightarrow bc \quad (u_1), \\ r_2 &: b \rightarrow ad \quad (u_2), \\ r_3 &: ad \rightarrow ac \quad (u_3). \end{aligned} \tag{2}$$

For example, the rule r_1 says that $2u_1$ molecules of substance a (a occurs two times as a reactant of r_1) are consumed while u_1 molecules of b and u_1 molecules of c

[1] The package Meta-P-lab, developed in Java within the research group on *Models of Natural Computing* led by the author, at the Department of Computer Science of the University of Verona, Italy [8], provides computational tools for MP systems. The last release is available at the site *mplab.sci.univr.it*.

are produced. This formulation of reactions is an implicit assumption of Avogadro's chemical principle, asserting that in a reaction the same number of molecules is involved for each substance occurrence in the reaction stoichiometry [31].

Let us assume to know the quantities $u_1[i], u_2[i], u_3[i]$, called *flux units* of r_1, r_2, r_3 respectively, giving the substance molar amounts consumed/produced by the reactions between two instants i and $i+1$. The reactant/product substance variations due to reaction r_j are multiples of u_j. Therefore, if we indicate by $a[i], b[i], c[i], d[i]$ the amount of substances a, b, c, d at the instant i, in the previous example, the following equations are easily derived by the form of reactions (2):

$$\begin{aligned} a[i+1]-a[i] &= -2u_1[i]+u_2[i], \\ b[i+1]-b[i] &= u_1[i]-u_2[i], \\ c[i+1]-c[i] &= u_1[i]+u_3[i], \\ d[i+1]-d[i] &= u_2[i]-u_3[i]. \end{aligned} \tag{3}$$

By looking at the reactions, we see that a is consumed by the first reaction and is produced by the second one, therefore, if the first reaction moves u_1 molecules and the second one moves u_2 molecules, the global variation of a is just of $-2u_1+u_2$ elements (the third reaction consumes, but at same time produces u_3 elements of a). Analogous arguments will provide the other equations. We can express synthetically the system of equations (3) in terms of usual matrix product:

$$X[i+1]-X[i] = \mathbb{A} \times U[i]$$

where

$$\begin{aligned} X[i] &= \big(a[i], b[i], c[i], d[i]\big), \\ U[i] &= \big(u_1[i], u_2[i], u_3[i]\big) \end{aligned}$$

and $\mathbb{A}$ is the following matrix:

$$\mathbb{A} = (\mathbb{A}_{x,r} \mid x \in X,\ r \in R) = \begin{pmatrix} -2 & 1 & 0 \\ 1 & -1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & -1 \end{pmatrix}.$$

This example shows that if we know the flux units at an instant i, then we are able to calculate the state of the system after a time interval τ. This will be the main point of our further investigation.

Let us consider a set X of substances and a set of R of reactions seen, as pairs of strings, and represented in the arrow notation according to which any reaction is identified by a rewriting rule $\alpha_r \to \beta_r$ with α_r, β_r strings over X (α_r represents the reactants of r, while β_r represents the products of r). As usual, for a string γ and a symbol x we denote by $|\gamma|_x$ the number of occurrences of the symbol x in the string γ, while $|\gamma|$ is the length of γ. Then the general definition of stoichiometric matrix can be given.

Definition 1 The *stoichiometric matrix* $\mathbb{A}$ of a set R of reactions over a set X of substances is $\mathbb{A} = (\mathbb{A}_{x,r} \mid x \in X,\ \ r \in R)$ where $\mathbb{A}_{x,r} = |\beta_r|_x - |\alpha_r|_x$. The set of reactions having the substance x as a reactant is $R_\alpha(x) = \{r \in R \mid |\alpha_r|_x > 0\}$ and the set of rules consuming or producing x is $R(x) = \{r \in R \mid \mathbb{A}_{x,r} \neq 0\}$. Two reactions r_1, r_2 *compete* for $x \in X$ if $r_1, r_2 \in R_\alpha(x)$ for some substance $x \in X$.

3 Metabolic P Systems

MP systems are essentially deterministic P systems where the transition to the next state (after some specified interval of time) is calculated according to a *mass partition strategy*, that is, the available matter of each substance is partitioned among all reactions which need to consume it. The policy of matter partition is regulated at each instant by *flux regulation maps*, or simply *flux maps*.

The notion of MP system we introduce here generalizes that one given originally in [29], and developed in [27–32], which was based on *reaction maps*. In fact, the log-gain theory we present in this paper can be naturally formulated in terms of *flux maps* rather than *reaction maps*. We distinguish the two kinds of MP systems as MPR and MPF systems (MP systems with reaction maps *versus* MP systems with flux maps), which we will show to be equivalent.

A discrete dynamical system is specified by a set of states and by a discrete dynamics on them, that is, by a function from the set $\mathbb{N}$ of natural numbers to the states of the system [25, 34]. In this context, the natural numbers which are argument of dynamics are called *instants* or *steps*.

Definition 2 (*MPF System*) An MP system with flux regulation maps, shortly an MPF system, is a discrete dynamical system specified by a construct

$$M = (X, R, V, Q, \Phi, \nu, \mu, \tau, q_0, \delta)$$

where X, R, V are finite disjoint sets, and the following conditions hold, with $n, m, k \in \mathbb{N}$:

- $X = \{x_1, x_2, \ldots, x_n\}$ is the set of *substances* (the types of molecules);
- $R = \{r_1, r_2, \ldots, r_m\}$ is the set of *reactions* over X, that is, pairs (in arrow notation) of type $\alpha \to \beta$ with α, β strings over the alphabet X;
- $V = \{v_1, v_2, \ldots, v_k\}$ is the set of *parameters* (such as pressure, temperature, volume, pH, ...) equipped by a set $\{h_v : \mathbb{N} \to \mathbb{R} \mid v \in V\}$ of *parameter evolution functions*;
- Q is the set of *states*, seen as functions $q : X \cup V \to \mathbb{R}$ from substances and parameters to real numbers. We denote by $q_{|X}$ the restriction of q to substances, and by $q_{|V}$ its restriction to parameters.

- $\Phi = \{\varphi_r \mid r \in R\}$ is a set of *flux maps*, where the function $\varphi_r : Q \to \mathbb{R}$ states the amount (moles) which is consumed/produced, in the state q, for every occurrence of a reactant/product of r. We set by $U(q) = (\varphi_r(q) \mid r \in R)$ the *flux vector* at state q;
- ν is a natural number which specifies the number of molecules of a (conventional) mole of M, as its *population unit*;
- μ is a function which assigns, to each $x \in X$, the *mass* $\mu(x)$ of a mole of x (with respect to some measure unit);
- τ is the *temporal interval* between two consecutive observation steps;
- $q_0 \in Q$ is the *initial state*;
- $\delta : \mathbb{N} \to Q$ is the dynamics of the system, given by: $\delta(0) = q_0$ and

$$\begin{aligned} \delta(i+1)_{|X} &= \mathbb{A} \times U\big(\delta(i)\big) + \delta(i)_{|X}, \\ \delta(i+1)_{|V} &= \big(h_v(i+1) \mid v \in V\big) \end{aligned} \tag{4}$$

  where $\mathbb{A}$ is the stoichiometric matrix of R over X, and ×, + are the usual matrix product and vector sum.

The algorithm of Table 1 computes (and provides as outputs) N steps in the evolution of a given metabolic system M starting from an initial state (the vector

Table 1 The computation of N steps of an MP system M

```
Metabolic-Algorithm($M$, $N$)
begin
For $i = 0, \ldots, N$ do
For each $x \in X$ do
    begin
    $y := x[i]$;
    For each $r \in R(x)$ do
        begin
        $u := \varphi_r(\delta(i))$;
        $z := u \cdot \mathbb{A}_{x,r}$;
        $y := y + z$;
        end
    $x[i+1] := y$ ;
    write x[i+1];
    end
For each $v \in V$ do
    begin
    $v[i+1] := h_v(i+1)$;
    write $v[i+1]$;
    end
end
```

notation $\delta(i) = (x_1[i], x_2[i], \ldots, x_n[i], v_1[i], v_2[i], \ldots, v_m[i])$ is used). The values of ν, μ, τ of Definition 2 have no direct influence in the mathematical description of the dynamics of an MP system. Nevertheless, they are essential for the physical interpretation of the dynamics with respect to a specific mass/time measure scale.

Given a real metabolic system which can be observed in its evolution, then almost all the elements occurring in the definition of MP system: substances, reactions, parameters, parameter evolutions …can be, in principle, deduced by a macroscopic observation of the system. The only component which cannot be directly measured by a macroscopic observation is the set Φ of flux functions. In fact, they depend on the internal microscopic processes on which molecules are involved. This means that the key point, for defining an MP system modeling a metabolic system, is the determination of this set of functions. This problem corresponds to the problem of evaluating the kinetic rate constants in differential models of metabolic processes. In Sect. 6, we will show that there is a method for deducing a set of functions approximating Φ, by means of a suitable procedure of observation and of an algebraic elaboration of the data collected during the observation.

Reaction maps were introduced in [29]. In MP systems with reaction maps, shortly MPR systems, flux units are not given by flux maps, but they are calculated by means of maps $f_r : Q \to \mathbb{R}$, where $r \in R$, called *reaction maps*, and by means of *inertia* functions $\psi_x : Q \to \mathbb{R}$, where $x \in X$. The reaction map f_r provides a value $f_r(q)$, called the *reactivity* of the rule r in the state q, while $\psi_x(q)$ determines the (molar) quantity of substance x that is not consumed in the state q. Flux units are calculated according to *partition ratios* of substance reactivities. In fact, at any step, the amount of each substance x is distributed among all the rules competing for it, proportionally to the reactivities of reactions in that step, but including as competing also the reaction $x \to x$ having as reactivity the inertia of x.

Definition 3 (*MPR System*) An MP system with reaction maps, shortly an MPR system, is a discrete dynamical system specified by a construct

$$M = (X, R, V, Q, F, \psi, \nu, \mu, \tau, q_0, \delta)$$

where all the components different from F, ψ, and δ are as in Definition 2. The set $F = \{f_r : Q \to \mathbb{R} \mid r \in R\}$ is constituted by functions, called reaction maps, ψ is a function $\psi : X \times Q \to \mathbb{R}$ which provides for each substance and state, the *inertia* of the substance in the state, that is, its quantity (number of moles) not available to be consumed by the reactions. The dynamics δ is that of the MPF system $M = (X, R, V, Q, \Phi, \nu, \mu, \tau, q_0, \delta)$ where the flux functions are given by the following equations:

$$w_{r,x}(q) = \frac{f_r(q)}{\psi_x(q) + \sum_{r' \in R_\alpha(x)} f_{r'}(q)}, \tag{5}$$

$$\varphi_r(q) = \begin{cases} f_r(q) & \text{if } \alpha_r = \lambda; \\ \min\{\frac{w_{r,y}(q) \cdot q(y)}{|\alpha_r|_x} \mid y \in \alpha_r\} & \text{otherwise.} \end{cases} \tag{6}$$

Equation (5) provides the partition of x which r is allowed to consume, while (6) computes the reaction unit of r, as the minimum of substance quantities available to all reactants of r.

Definition 4 Two MP systems are equivalent when they have the same sets X (substances) and V (parameters, with the same related evolution functions), the same $\nu, \mu, q_{0|X}, \tau$ (of Definition 2), and the same dynamics.

From Definition 3, it follows that for any MPR an equivalent MPF exists, now we show that also the converse relation holds, that is, for any MPF an equivalent MPR exists.

Definition 5 An MP system is *monic* if $|\alpha_r| \leq 1$ for every $r \in R$.

Lemma 6 *For any MPF system there exists a monic MPF system which is dynamically equivalent to it.*

Proof If in a rule, for example $r : ab \to c$, many substances occur in the left side, then we can split it into two rules: $r_1 : a \to c$, $r_2 : c \to \lambda$ by requiring that the fluxes of the two rules are equal. We can proceed in this way for all the rules which are not monic. Of course, the monic MP system which we get in this manner is equivalent to the original one. □

Theorem 7 *For any MPF system, there exists an equivalent MPR system.*

Proof According to the previous lemma, we can assume that the MPF system M we consider is monic. Then we define an MPR system M' where for any rule r, we set the value of its reaction map f'_r equal to the corresponding flux map of M, that is, for any state q, $f'_r(q) = \varphi_r(q)$, and at the same time, we add, for any substance x, an inertia function ψ_x defined as $\psi_x(q) = q(x) - \sum_{r \in R_\alpha(x)} f'_r(q)$. With these positions, by Definition 3, for any rule $r \in R$ having $\alpha_r = \lambda$, we have a flux map φ'_r in M' that is trivially equal to φ_r. Otherwise, for a rule r with $\alpha_r \neq \lambda$, the following equations hold for any $y \in \alpha_r$:

$$w'_{r,x}(q) = \frac{f'_r(q)}{\psi_x(q) + \sum_{r' \in R_\alpha(x)} f'_{r'}(q)}, \tag{7}$$

$$\varphi'_r(q) = \min\{w'_{r,y}(q) \cdot q(y) \mid y \in \alpha_r\} \tag{8}$$

but for any rule $r \in R$ and for any $y \in \alpha_r$, from (7) we have

$$w'_{r,y}(q) \cdot q(y) = \frac{f'_r(q) \cdot q(y)}{(q(y) - \sum_{r' \in R_\alpha(y)} f'_r(q)) + \sum_{r' \in R_\alpha(y)} f'_{r'}(q)} = f'_r(q) \tag{9}$$

therefore, as we assumed M to be monic, this value coincides with the minimum requested by the equation (8). Finally, by the position $f'_r(q) = \varphi_r(q)$, it results, from (8) and (9), that $\varphi'_r(q) = \varphi_r(q)$. In conclusion, the fluxes of the MPR system M' are the same of the MPF system M which we started from, whence the two systems result to be equivalent. □

4 MP Models and Differential Models

It was shown in [17] that a special correspondence can be established between MPR systems and differential models, such that, under suitable hypotheses and within some definite approximation, the MPR non-cooperative systems provide the same evolutions computed by numerical integration of the corresponding ODE. In fact, MPR systems transform naturally into ODE systems according to the *mass action* principle, on which differential models are based on. According to the mass action, the amount of products generated by a reaction is proportional to the product of quantities of its substrates (considered with their multiplicity). This idea is formalized by the following definition, where x' is the derivative of x with respect to time. In this section, all the MP systems we consider are MPR systems.

Definition 8 (*MPR–ODE Transform*) Let $M = (X, R, \ldots F, \ldots)$ be an MPR systems (F the set of reaction maps). The following system of equations $ODE(M)$, for $x \in X$, where $\Pi(\alpha_r)$ is the product of all the quantities of reactants of reaction r, is called the *ODE*-transformed of M:

$$x' = \sum_{r \in R} \mathbb{A}_{x,r} f_r(q) \Pi(\alpha_r).$$

Let us briefly recall from [17] the main results concerning the relationship between reactivity MPR systems and ordinary differential equations (here the term "monic" corresponds to the attribute "non-cooperative" of [17]).

Theorem 9 *For any MPR system M, there exists a monic MPR system M' having the same ODE transform of M.*

Theorem 10 *Given an ODE system E, an MPR system can be found having E as its ODE-transform.*

Definition 11 (*Uniformly Inerzial MPR System*) For some positive $\psi \in \mathbb{R}$, an MPR system is ψ-uniformly inertial if all its substances have as inertia a constant value ψ.

Theorem 12 *Given a non-cooperative, ψ-uniformly inertial MPR system M, if $ODE(M)$ satisfies the differential conditions of univocal solvability, then the evolution of M converges, as $\psi \to \infty$, to the solution of $ODE(M)$ having the initial conditions of M.*

4.1 An MP Model of Mitotic Oscillator

One of the most interesting modeling results for reactivity MP systems was the translation of Golbeter's differential model of mitotic oscillations, formalized by (10) referring to Table 2 [22, 23].

The fundamental mechanism of mitotic oscillations concerns the fluctuation in the activation state of a protein, produced by the *cdc*2 gene in fission yeast or by homologous genes in other eukaryotes. The simplest form of this mechanism is found in early amphibian embryos. Here (see Fig. 1), cyclin is synthesized at a constant rate v_i and triggers the transformation of inactive (M^+) into active (M) *cdc*2 protein, which provides the formation of a complex known as M-phase promoting factor (*MPF*) which triggers mitosis, but at same time M, by a chain of events that we do not mention, elicits the transformation of a protease, from state X^+ to X, and X degrades cyclin, producing the inactivation of $cdc2$ that brings the cell back to the initial conditions in which a new division cycle can take place.

The following ODE is the differential model of the dynamics described in Fig. 1, where c, m, x are the percentages of C, M, X, respectively (while $1 - m$, $1 - x$ are

Table 2 Numerical values of Goldbeter's model (expressed in suitable measure units based on micro-mole and second)

$K_1 = 0.005$	$K_2 = 0.005$	$K_3 = 0.005$	$K_4 = 0.005$
$V_{M1} = 3$	$V_{M3} = 1$	$V_2 = 1.5$	$V_4 = 0.5$
$v_i = 0.025$	$v_d = 0.25$	$K_c = 0.5$	$K_d = 0.02$
$k_d = 0.01$	$c[0] = 0.01$	$x[0] = 0.01$	$m[0] = 0.01$

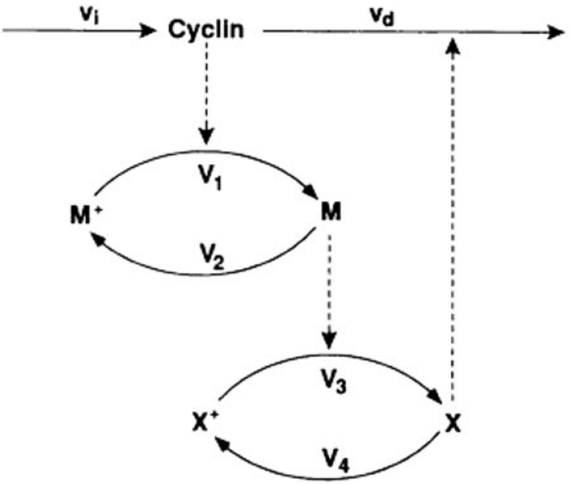


Fig. 1 The model provided by A. Goldbeter from [22]

the percentages of M^+, X^+, respectively):

$$\begin{aligned} \frac{dc}{dt} &= v_i - v_d x \frac{c}{K_d + c} - K_d c, \\ \frac{dm}{dt} &= V_1 \frac{(1-m)}{K_1 + (1-m)} - V_2 \frac{m}{K_2 + m}, \\ \frac{dx}{dt} &= V_3 \frac{(1-x)}{K_3 + (1-x)} - V_4 \frac{x}{K_4 + x}. \end{aligned} \tag{10}$$

Figure 2 is a solutions of these equations obtained by numerical integration for some value of parameters given in [22] and reported in Table 2.

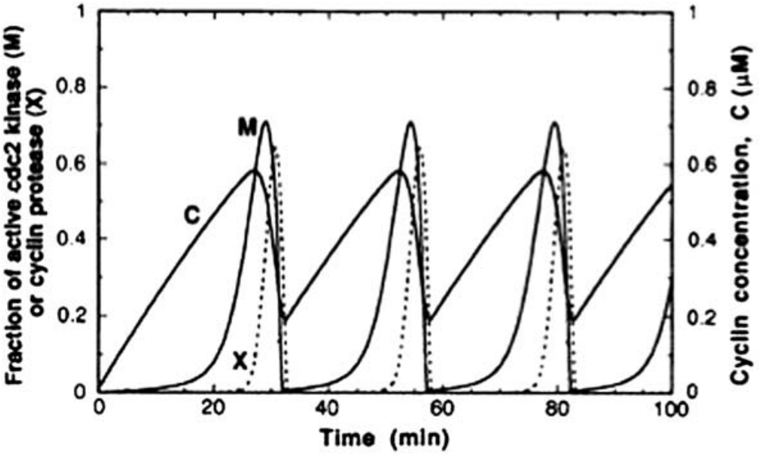


Fig. 2 A numerical solution of the set of differential equations (10) implementing the model provided by A. Goldbeter from [22]

Table 3 An MP model of the differential system (10)

$r_1 : A \to AC$	$f_1 = \frac{v_i}{a}$
$r_2 : C \to X$	$f_2 = \frac{v_d X}{K_d + c}$
$r_3 : X \to \lambda$	$f_3 = \frac{v_d c}{K_d + c}$
$r_4 : C \to \lambda$	$f_4 = k_d$
$r_5 : C \to MC$	$f_5 = \frac{V_{M1} M^+}{(K_c + c)(K_1 + (1-m))}$
$r_6 : M^+ \to \lambda$	$f_6 = \frac{V_{M1} C}{(K_c + C)(K_1 + (1-m))}$
$r_7 : M \to M^+$	$f_7 = \frac{V_2}{K_2 + m}$
$r_8 : X^+ \to XM$	$f_8 = \frac{V_{M3} m}{K_3 + (1-x)}$
$r_9 : M \to \lambda$	$f_9 = \frac{V_{M3} \cdot (1-x)}{K_3 + (1-x)}$
$r_{10} : X \to X^+$	$f_{10} = \frac{V_4}{K_4 + x}$

An MP system corresponding to Goldbeter's model is given in Table 3. It was obtained by means of the general procedure described in [17] and provides (for suitable specific parameters of the system) a dynamics equal to that of Fig. 2.

5 Metabolic Log-gain Principles

Given a real system that we can observe for a certain number of steps, is it possible to determine an MP system which could predict, within an acceptable approximation, the future behavior of the given system? We will show how this task could be achieved. In fact, in some cases, we can determine, in a systematic way, an MP system which is an adequate model of some observed metabolic dynamics.

In order to discover the reaction fluxes at each step, we introduce some *log-gain principles*. In fact, it seems to be natural that a proportion should exist between the relative variation of substances and the relative variation of flux units. The relative variation of a substance x is defined as the ratio $\varDelta(x)/x$. In differential notation (with respect to the time variable), this ratio is related to $\frac{dx}{dt}/x$, and from elementary calculus we know that it is the same as $\frac{d(\lg x)}{dt}$. This equation explains the term "log-gain" for expressing relative variations [46]. We use a discrete notion of log-gain for stating the following principle. In the following, given a dynamics δ of an MP system, we use a simplified notations, for $i \in \mathbb{N}$, $r \in R$, and $w \in X \cup V$ (notation of Definition 2):

$$u_r[i] = \varphi_r\big(\delta(i)\big),$$
$$w[i] = \big(\delta(i)\big)(w).$$

Principle 13 (*Log-gain*) For $i \in \mathbb{N}$ and $r \in R$, let us call

$$Lg\big(u_r[i]\big) = \big(u_r[i+1] - u_r[i]\big)/u_r[i]$$

the log-gain of the flux unit u_r at the step i, and analogously,

$$Lg\big(w[i]\big) = \big(w[i+1] - w[i]\big)/w[i]$$

the log-gain of the substance or parameter w at step i. There exists a subset T_r of $X \cup V$ of elements called (log-gain) *tuners* of r such that: $Lg(u_r[i])$ is a linear combination, in a unique way, of the tuners of r:

$$Lg\big(u_r[i]\big) = \sum_{w \in T_r} p_{r,w} Lg\big(w[i]\big). \tag{11}$$

Log-gain principle extends a very important rule, well known in theoretical biology as the *allometric principle*. According to it, a specific ratio holds between the relative variations of two related biological parameters (e.g. mass of an organism and its superficial area). This principle seems to be a general property of living

Table 4 Sirius' reactions and reaction maps, where $k_1 = 4, k_2 = 0.02, k_3 = 4, k_4 = 0.02, k_5 = 4, a[0] = 100, b[0] = 100, c[0] = 0$, and $k = 100$ is the inertia of each substance

$r_1:$	$a \to aa$	$f_1 = k_1$
$r_2:$	$a \to b$	$f_2 = k_2 \cdot c$
$r_3:$	$b \to \lambda$	$f_3 = k_3$
$r_4:$	$a \to c$	$f_4 = k_4 \cdot b$
$r_5:$	$c \to \lambda$	$f_5 = k_5$

organisms which allows them to keep basic equilibria underlying their internal organization. As it is reported in [2], many empirical laws on metabolism are instances of *allometry* and also the abundance of power laws in biological systems is related to this principle. Therefore, it is not surprising that log-gain mechanism is the basis for deducing the right metabolic algorithm of MP systems which fit the observations of given metabolic processes.

Now, starting from this first formulation of log-gain principle, we will present a method for constructing an MP system which reproduces, with an acceptable approximation, the dynamics of a given system that we *observe* for a sufficient number of steps. In this context "observe" means to know, with a sufficient precision, the (molar) quantities of all different kinds of molecules, and the values of parameters, for a sequence of steps. Let us denote these quantities, according to our simplified notation, with the sequence, for $i = 0, \dots, t$, of vectors:

$$\delta(i) = \big(x_1[i], x_2[i], \dots, x_n[i], v_1[i], v_2[i], \dots, v_k[i]\big).$$

Moreover, we assume to known the reactions, the parameter evolutions, and the tuners of reactions (the time unit and the molar unit are fixed). We want to predict the substance vectors $X[i]$ for steps $i > t$. We solve the problem when we discover the flux maps Φ. By using a simple metabolic systems, called Sirius, we present a procedure, based on a log-gain principle. This procedure will allow us to determine some vectors of flux units $U[i]$ which provide an MP dynamics fitting with given observation data.

Consider the metabolic system Sirius given in Table 4. This system is interesting because it provides a simple metabolic oscillator with no parameter. Its differential formulation, according to the correspondence stated in [17], is given by equations (12), its dynamics generated by an MPR system is given in Fig. 3:

$$\begin{aligned}\frac{da}{dt} &= k_1 a - k_2 ca - k_4 ba,\\ \frac{db}{dt} &= k_2 ac - k_3 b,\\ \frac{dc}{dt} &= k_4 ab - k_5 c.\end{aligned} \tag{12}$$

Consider two consecutive steps i and $i + 1$ of its evolution. Let $a[i], b[i], c[i]$ be the quantities of a, b, c, respectively (expressed in moles), at step i. Substance a is produced by the rules r_1 and it is consumed by the rules r_2, r_4, and analogously

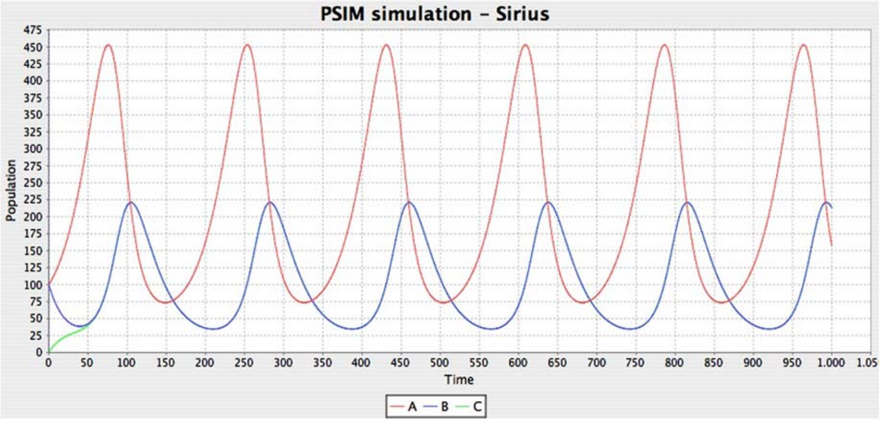


Fig. 3 Sirius' oscillator dynamics

for the other substances. Therefore, by the stoichiometric matrix of the reactions, we get the following system of equations we call $SD[i]$ (*Substance Differences at step* i). We assume that this system has maximal rank, that is, in it no equation is a linear combination of other equations:

$$\begin{aligned}
a[i+1]-a[i] &= u_1[i]-u_2[i]-u_4[i],\\
b[i+1]-b[i] &= u_2[i]-u_3[i],\\
c[i+1]-c[i] &= u_4[i]-u_5[i].
\end{aligned} \tag{13}$$

Let the values $u_1[i], u_2[i], u_3[i], u_4[i], u_5[i]$ be the flux units at the steps i, then the log-gain module provides the following system of equations $LG[i]$:

$$\begin{aligned}
Lg\bigl(u_1[i]\bigr) &= p_1 Lg\bigl(a[i]\bigr),\\
Lg\bigl(u_2[i]\bigr) &= p_2 Lg\bigl(a[i]\bigr)+p_3 Lg\bigl(c[i]\bigr),\\
Lg\bigl(u_3[i]\bigr) &= p_4 Lg\bigl(b[i]\bigr),\\
Lg\bigl(u_4[i]\bigr) &= p_5 Lg\bigl(a[i]\bigr)+p_6 Lg\bigl(b[i]\bigr),\\
Lg\bigl(u_5[i]\bigr) &= p_7 Lg\bigl(c[i]\bigr).
\end{aligned} \tag{14}$$

Putting together the systems (14) and (13), at steps i and $i+1$, respectively, we get the system that we indicate by $LG[i]+SD[i+1]$ and we call *observation log-gain module* at step i.

Now, consider the (observation) log-gain module at step 0 and assume to know the vector $U[0]$ (a method for finding it is outlined in [31]). Moreover, assume to know the tuners of the reactions, and set to 1 the coefficients of the tuners, but add a value p_r to the log-gain linear combination of rule r, which we call *log-gain offset* of rule r, and which includes the error we introduce by reducing to 1 the coefficients of the tuner log-gains. As an instance, Sirius' log-gain module has 8 equations and 12 variables, but after setting to 1 the log-gain coefficients and adding the log-gain offsets, we get a system of 8 equations and 10 variables. In general, an *offset log-gain module* is determined according to the following principle.

Principle 14 (*Offset Log-gain*) There exists a subset T_r of $X \cup V$ of elements called (log-gain) *tuners* of r and a unique value p_r, called offset log-gain, such that

$$Lg\big(u_r[i]\big) = \sum_{w \in T_r} Lg\big(w[i]\big) + p_r. \tag{15}$$

The offset log-gain module, we call $OLG[i]$, has $n + m$ equations and $2m$ unknown variables. Now, we show that we can reduce the number of variables by obtaining a square linear system. In fact, if we consider the stoichiometric module $SD[i+1]$, we realize that the sum of the offsets of reactions consuming or producing a given substance x is constrained to be equal to a fixed value. For example, in the case of Sirius, we have the following $SD[i+1]$ system:

$$\begin{aligned} a[i+2] - a[i+1] &= u_1[i+1] - u_2[i+1] - u_4[i+1], \\ b[i+2] - b[i+1] &= u_2[i+1] - u_3[i+1], \\ c[i+2] - c[i+1] &= u_4[i+1] - u_5[i+1] \end{aligned} \tag{16}$$

that can be rewritten, in terms of (offset) log-gains, in the following manner:

$$\begin{aligned} &a[i+1]Lg\big(a[i+1]\big) - u_1[i] - u_2[i] - u_4[i] \\ &\quad = u_1[i]Lg\big(u_1[i]\big) - u_2[i]Lg\big(u_2[i]\big) - u_4[i]Lg\big(u_4[i]\big). \\ &b[i+1]Lg\big(b[i+1]\big) - u_2[i] - u_3[i] = u_2[i]Lg\big(u_2[i]\big) - u_3[i]Lg\big(u_3[i]\big), \\ &c[i+1]Lg\big(c[i+1]\big) - u_4[i] - u_5[i] = u_4[i]Lg\big(u_4[i]\big) - u_5[i]Lg\big(u_5[i]\big) \end{aligned} \tag{17}$$

now, let us distinguish in the log-gain offset linear combination the non-offset part, by setting

$$Lg_{u_r}[i] = \sum_{w \in T_r} Lg\big(w[i]\big) \tag{18}$$

with this notation the previous system can easily be put in the following form:

$$\begin{aligned} &a[i+1]Lg\big(a[i+1]\big) - u_1[i] + u_2[i] + u_4[i] \\ &\quad = u_1[i]Lg_{u_1}[i] - u_2[i]Lg_{u_2}[i] - u_4[i]Lg_{u_4}[i] + p_1 + p_2 + p_4, \\ &b[i+1]Lg\big(b[i+1]\big) - u_2[i] + u_3[i] = u_2[i]Lg_{u_2}[i] - u_3[i]Lg_{u_3}[i] + p_2 + p_3, \\ &c[i+1]Lg\big(c[i+1]\big) - u_4[i] + u_5[i] = u_4[i]Lg_{u_4}[i] - u_5[i]Lg_{u_5}[i] + p_4 + p_5 \end{aligned}$$

that is, for suitable linear operators $K_1, K_2, K_3, H_1, H_2, H_3$:

$$\begin{aligned} &K_1\big(Lg_{u_1}[i], Lg_{u_2}[i], Lg_{u_4}[i]\big) = H_1(p_1 + p_2 + p_4), \\ &K_2\big(Lg_{u_2}[i], Lg_{(u_3}[i]\big) = H_2(p_2 + p_3), \\ &K_3\big(Lg_{u_4}[i], Lg_{u_5}[i]\big) = H_3(p_4 + p_5). \end{aligned} \tag{19}$$

From system (19), it becomes apparent that if we impose to the log-gain offsets of rule r_2 and r_4 to be equal to 0, then the log-gain gaps covered by their log-gain offsets can be covered by the offset of the rule r_1. Analogously, by the other equations we can choose, as log-gain offsets different from 0, the offsets of rules r_3 and r_5, respectively. In conclusion, we reduce only to three the number of log-gain offsets occurring in the offset log-gain module, reaching finally a system with a number of variables equal to the number of equations.

The previous argument can be easily generalized to any offset log-gain module, by stating the following lemma.

Lemma 15 *In any log-gain module $OLG[i]$, for every substance x, the sum of all the offsets of rules in $R(x)$ is equal to a value depending on the log-gains of tuners of rules of $R(x)$.*

On the basis of this lemma, we can obtain a further elaboration of the log-gain principle for reducing the number of offsets of log-gain module exactly to the number of substances.

Definition 16 (*Offset Log-gain Covering Property*) Given an MP system M of substances $X = \{x_1, \dots, x_n\}$, then a set $R_0 = \{r_{j_1}, \dots, r_{j_n}\}$ of reactions has the offset log-gain covering property if $r_i \in R(x_i)$ and, for every $1 < i \leq n$.

Consider a case of two substances x_1, x_2 such that $R(x_1) = \{r_1, r_2, r_3, r_4\}$ and $R(x_2) = \{r_2, r_3\}$. In this case, if we choose r_1 from $R(x_1)$ and r_3 from $R(x_2) - R(x_2)$, then we get a set $R_0 = \{r_1, r_3\}$ having the offset log-gain covering property.

The offset log-gain covering property allows us to write down a system of equations for computing, at each step, the reaction fluxes of a system where states are known by observation.

Principle 17 (*Offset Log-gain Adjustment*) Let M be an MP system with a set of reactions $R = \{r_1, r_2, \dots, r_m\}$ and a set of substances $X = \{x_1, x_2, \dots, x_n\}$. Let R_0 be a subset of reactions having the offset log-gain covering property. The following system $OLGA[i]$ is associated to any step i of the dynamics of M, where $r \in R$, T_r are the tuners of reaction r, $x \in X$ and the log-gain offset $p_r[i] = 0$ if $r \notin R_0$:

$$\begin{aligned} &Lg\big(u_r[i]\big) = \sum_{w \in T_r} Lg\big(w[i]\big) + p_r[i], \\ &x[i]Lg\big(x[i+1]\big) = (\mathbb{A}_{x,r_1}, \mathbb{A}_{x,r_2}, \dots, \mathbb{A}_{x,r_m}) \times U[i+1]. \end{aligned} \tag{20}$$

Theorem 18 *The log-gain principles implies that* $OLGA[i]$ *of an MP system, for any step* $i \in \mathbb{N}$, $i > 0$, *has one and only one solution.*

Proof (informal) The variations of substances, at step i, determine the substance log-gains, and, according to the log-gain principles, they determine univocally the log-gain of the flux units at step i. Consequently, the vector $U[i+1]$ is determined, and finally according to (4), the new state of the system is completely determined. □

Corollary 19 *Given an MP system* M, *A set* R_0 *of reactions of* M *having the offset log-gain covering property can be always found.*

When many different subsets of reactions can be found which have the offset log-gain covering property, it is reasonable to assume that the choice which provides the offset vectors with minimal norm (in $\mathbb{R}^n$) represents the *OLGA* system which yields the better approximations to the real fluxes of a given observed dynamics.

However, criteria and methods are under investigation for finding offset log-gain coverings which could provide optimal results. Moreover, what is stated in the previous theorem and corollary should be grounded on a formal basis, by introducing an algebraic formulation of the offset log-gain covering property. More specifically, we intend to provide suitable conditions ensuring that the OLGA matrix corresponding to a covering has a non-null determinant. These questions will be topics of further elaborations of the log-gain theory for MP systems.

In the case of Sirius, the following $OLGA[i]$ system was obtained, where flux units and substance quantities at step i are assumed to be known, while unknown variables, in the left side of equations, are the components of flux unit vector $U[i+1]$ and of the offset vector $O[i+1] = (p_1[i+1], p_3[i+1], p_5[i+1])$.

$$\begin{aligned}
&u_1[i+1] - p_1[i+1]u_1[i] = u_1[i]Lg\big(a[i]\big) + u_1[i],\\
&u_2[i+1] = u_2[i]Lg(a[i]) + u_2[i]Lg\big(c[i]\big) + u_2[i],\\
&u_3[i+1] - p_3[i+1]u_3[i] = u_3[i]Lg\big(b[i]\big) + u_3[i],\\
&u_4[i+1] = u_4[i]Lg\big(a[i]\big) + u_4[i]Lg\big(b[i]\big) + u_4[i],\\
&u_5[i+1] - p_5[i+1]u_5[i] = u_5[i]Lg\big(c[i]\big) + u_5[i],\\
&u_1[i+1] - u_2[i+1] - u_4[i+1] = a[i+2] - a[i+1],\\
&u_2[i+1] - u_3[i+1] = b[i+2] - b[i+1],\\
&u_4[i+1] - u_5[i+1] = c[i+2] - c[i+1].
\end{aligned} \tag{21}$$

If we solve the $OLGA[i]$ of Sirius for $i = 0$ ($U[0]$ is assumed to be known, and $X[i] = (a[i], b[i], c[i])$, for $i = 1, \dots, t$, are given by observation), then we get $U[1]$. If we apply the same procedure again for $i = 1, 3, \dots$, we get the vectors $U[2]$, $O[2]$, $U[3]$, $O[3], \dots, U[t]$, $O[t]$. Now, assume that reaction fluxes depend on the quantities a, b, c with some polynomial dependence of a given degree, say a third degree, then we can use standard approximation tools for finding the functional dependence of the vector U with respect to the substance quantities a, b, c. The resulting polynomials approximate the regulation functions Φ we are searching for, and our task is completed.

General methods are under investigation which could systematically search for polynomials (depending on substance quantities and parameter values) which also fit with the flux units determined by *OLGA* modules solutions. In some numerical experiments [36], flux polynomials were found with a good approximation to the observed dynamics. Our procedure, based on the log-gain principles, assumes the knowledge of $U[0]$. Actually, there are several possibilities under investigation and one of them is based on a suitable iteration of the log-gain module [31]. However, we discovered experimentally a very interesting fact, which deserves a more subtle theoretical investigation. Consider the system $SD[i]$ of 3 equations and 5 variables $(u_1, u_2, u_3, u_4, u_5)$, choose one of its infinite solutions (imposing some additional very natural constraints) and identify it with $U[0]$. In many cases, we found that independently from the chosen value of $U[0]$, after a small number of steps, say 3 steps, our procedure will generate with a great approximation, the same sequence of flux vectors. This means that the data collected in the observation steps are sufficient to determine the functions which on the basis of substance quantities, regulate the dynamics of the system. The method was tested for Sirius and other systems (Lotka–Volterra, Brusselator, Mitotic Oscillator). Numerical elaborations were performed by standard MATLAB® operators (*backslash* operator for square matrix left division or in the least squares sense solution) and interpolation was performed by polynomials of third degree. Specific observation strategies were adopted, by using about one hundred steps. In almost all cases, the observed dynamics were correctly reconstructed [36]. This means that the flux functions, deduced according to the meta-metabolic algorithm, provide MP systems with the same dynamics of the observed systems. Of course, in our experiments we discovered what we already knew, because the systems we observed were artificial systems, but this does not diminishes the validity of our method, as it showed a good level of coincidence between observation and deduced regulation. Therefore, in the case of natural systems, from suitable observations, we could discover, with good approximation, the underlying regulation maps, and consequently, reliable computational models of their dynamics can be found. However, applications of meta-metabolic algorithm to more complex dynamics as well as deeper theoretical analyses of the simulation results will be topics for further research.

6 Conclusions

MP systems proved to be relevant in the analysis of dynamics of metabolic processes. Their structure clearly distinguishes a reaction level and a regulation level. We showed that an essential component of the regulation level can be deduced by applying the log-gain theory to data that can be collected from observations of the systems. The search of efficient and systematic methods for defining MP systems from experimental data is of crucial importance for systematic applications of MP systems to complex dynamics. The log-gain method can deduce, in a given metabolic system, a time series of (approximate) flux unit vectors $U[i]$ (i ranging in time instants), from a time series of observed states. This method is based on the solution of one linear system (of $n+m$ equations and $n+m$ variables) for each value of i. Two crucial tasks remain to be performed for a complete discovery of the underlying MP dynamics which explains an observed dynamics: (i) a systematic reliable determination of the initial vector $U[0]$ which is the basis of our iterative method (results in this direction are in [31]), and (ii) a systematic way for deducing the flux regulation maps (depending on the state) from the time series of flux vectors. Our future research will focus on these problems in order to show the applicability of our method to real complex cases. At this end, three main research lines are under investigation: (i) extending the theoretical aspects, (ii) performing suitable biological experiments, and (iii) developing computational tools for modeling biological phenomena.

References

1. Bernardini F, Manca V (2003) Dynamical aspects of P systems. Biosystems 70:85–93
2. von Bertalanffy L (1967) General systems theory: foundations, developments, applications. George Braziller Inc, New York
3. Besozzi D, Ciobanu G (2005) A P system description of the sodium-potassium pump. In: Mauri G, Păun G, Pérez-Jiménez MJ, Rozenberg G, Salomaa A (eds) Membrane computing, WMC 2004. Lecture notes in computer science, vol 3365. Springer, Berlin, pp 210–223
4. Bianco L (2007) Membrane models of biological systems. PhD thesis, University of Verona
5. Bianco L, Fontana F, Franco G, Manca V (2006) P systems for biological dynamics. In: Ciobanu G, Păun G, Pérez-Jiménez MJ (eds) Applications of membrane computing. Springer, Berlin, pp 81–126
6. Bianco L, Fontana F, Manca V (2005) Reaction-driven membrane systems. In: Wang L, Chen K, Ong Y-S (eds) Advances in natural computation. Lecture notes in computer science, vol 3611. Springer, Berlin, pp 1155–1158
7. Bianco L, Fontana F, Manca V (2006) P systems with reaction maps. Int J Found Comput Sci 17(1):27–48
8. Bianco L, Manca V, Marchetti L, Petterlini M (2007) Psim: a simulator for biomolecular dynamics based on P systems. In: IEEE congress on evolutionary computation (CEC 2007), pp. 883–887
9. Bianco L, Pescini D, Siepmann P, Krasnogor N, Romero-Campero FJ, Gheorghe M (2007) Towards a P systems pseudomonas quorum sensing model. In: Membrane computing, WMC 2006. Lecture notes in computer science, vol 4361. Springer, Berlin, pp 197–214
10. Ciobanu G, Păun G, Pérez-Jiménez MJ (eds) (2006) Applications of membrane computing. Springer, Berlin
11. Castellini A, Franco G, Manca V (2007) Toward a representation of hybrid functional Petri nets by MP systems. In: Suzuki, Y., Păun, G., Adamatzky, A., Hagiya, M., Umeo, H. (eds) Natural computing, 2nd international workshop on natural computing, IWNC University of Nagoya, Japan, pp 28–37
12. Ehrenfeucht A, Rozenberg G (2004) Basic notions of reaction systems. In: Lecture notes in computer science, vol 3340. Springer, Berlin, pp 27–29
13. Ehrenfeucht A, Rozenberg G (2006) Reaction systems. Fund Inform 76:1–18
14. Ehrenfeucht A, Rozenberg G (2007) Events and modules in reaction systems. Theor Comput Sci 376:3–16

15. Fontana F, Bianco L, Manca V (2005) P systems and the modeling of biochemical oscillations. In: Freund R, Păun G, Rozenberg G, Salomaa A (eds) Membrane computing, WMC 2005. Lecture notes in computer science, vol 3850. Springer, Berlin, pp 199–208
16. Fontana F, Manca V (2008) Predator-prey dynamics in P systems ruled by metabolic algorithm. Biosystems 91(3):545–557
17. Fontana F, Manca V (2007) Discrete solutions to differential equations by metabolic P systems. Theor Comput Sci 372:165–182
18. Franco G, Manca V (2004) A membrane system for the leukocyte selective recruitment. In: Alhazov A, Martín-Vide C, Păun G (eds) Membrane computing, WMC 2003. Lecture notes in computer science, vol 2933. Springer, Berlin, pp 180–189
19. Franco G, Guzzi PH, Mazza T, Manca V (2007) Mitotic oscillators as MP graphs. In: Hoogeboom HJ, Păun G, Rozenberg G, Salomaa A (eds) Membrane computing, WMC 2006. Lecture notes in computer science, vol 4361. Springer, Berlin, pp 382–394
20. Franco G, Jonoska N, Osborn B, Plaas A (2008) Knee joint injury and repair modeled by membrane systems. Biosystems 91(3):473–488
21. Gillespie DT (1977) Exact stochastic simulation of coupled chemical reactions. J Phys Chem 81(25):2340–2361
22. Goldbeter A (1991) A minimal cascade model for the mitotic oscillator involving cyclin and cdc2 kinase. Proc Natl Acad Sci 88(20):9107–9111
23. Goldbeter A (2004) Biochemical oscillations and cellular rhythms. Cambridge University Press, New York
24. Jost J (2005) Dynamical systems. Springer, Berlin
25. Kurka P (2003) Topological and symbolic dynamics. Société Mathématique de France, Paris
26. Manca V (1998) String rewriting and metabolism: a logical perspective. In: Păun G (ed) Computing with bio-molecules. Springer, Berlin, pp 36–60
27. Manca V (2006) Topics and problems in metabolic P systems. In: Păun G, Pérez-Jiménez MJ (eds) Membrane computing, BWMC4. Fenix Editora, Sevilla, pp 173–183
28. Manca V (2006) MP systems approaches to biochemical dynamics: Biological rhythms and oscillations. In: Hoogeboom HJ, Păun G, Rozenberg G, Salomaa A (eds) Membrane computing, WMC 2006. Lecture notes in computer science, vol 4361. Springer, Berlin, pp 86–99
29. Manca V (2007) Metabolic P systems for biochemical dynamics. Prog Nat Sci 17(4):384–391
30. Manca V (2008) Discrete simulations of biochemical dynamics. In: Garzon MH (ed) DNA 13. Lecture notes in computer science, vol 4848. Springer, Berlin, pp 231–235
31. Manca V (2008) The metabolic algorithm: principles and applications. Theor Comput Sci 404:142–157
32. Manca V, Bianco L (2008) Biological networks in metabolic P systems. Biosystems 91(3):489–498
33. Manca V, Bianco L, Fontana F (2005) Evolutions and oscillations of P systems: Applications to biological phenomena. In: Mauri G, Păun G, Pérez-Jiménez MJ, Rozenberg G, Salomaa A (eds) Membrane computing, WMC 2004. Lecture notes in computer science, vol 3365. Springer, Berlin, pp 63–84
34. Manca V, Franco G, Scollo G (2005) State transition dynamics: basic concepts and molecular computing perspectives. In: Gheorghe M (ed) Molecular computational models: unconventional approachers. Idea Group Inc, UK, Chap. 2, pp 32–55
35. Manca V, Martino DM (1999) From string rewriting to logical metabolic systems. In: Păun G, Salomaa A (eds) Grammatical models of multi-agent systems. Gordon and Breach, London
36. Pagliarini R (2007) Esperimenti per la determinazione computazionale dei parametri regolativi nei P sistemi metabolici. Master thesis, University of Verona
37. Păun G (2000) Computing with membranes. J Comput Syst Sci 61(1):108–143
38. Păun G (2002) Membrane computing. An introduction. Springer, Berlin
39. Scollo G, Franco G, Manca V (2006) A relational view of recurrence and attractors in state transition dynamics. In: Schmidt R (ed) Relations and 25 kleene algebra in computer science. Lecture notes in computer science, vol 4136. Springer, Berlin, pp 358–372
40. Scollo G, Franco G, Manca V (2008) Relational state transition dynamics. J Log Algebr Program 76:130–144

41. Pérez–Jiménez MJ, Romero-Campero FJ (2005) A study of the robustness of the EGFR signalling cascade using continuous membrane systems. In: Mira J, Alvarez JR (eds) Mechanisms, symbols, and models underlying cognition, IWINAC 2005. Lecture notes in computer science, vol 3561. Springer, Berlin, pp 268–278
42. Pescini D, Besozzi D, Mauri G, Zandron C (2006) Dynamical probabilistic P systems. Int J Found Comput Sci 17(1):183–204
43. Suzuki Y, Fujiwara Y, Tanaka H, Takabayashi J (2001) Artificial life applications of a class of P systems: abstract rewriting systems on multisets. In: Calude CS, Păun G, Rozenberg G, Salomaa A (eds) Multiset processing, mathematical, computer science, and molecular computing points of view. Lecture notes in computer science, vol 2235. Springer, Berlin, pp 299–346
44. Suzuki Y, Tanaka H (2002) A symbolic chemical system based on an abstract rewriting system and its behavior pattern. J Artif Life Robot 6:129–132
45. Suzuki Y, Tanaka H (2006) Modelling p53 signaling pathways by using multiset processing. In: Ciobanu G, Păun G, Pérez-Jiménez MJ (eds) Applications of membrane computing. Springer, Berlin, pp 203–214
46. Voit EO (2000) Computational analysis of biochemical systems. Cambridge University Press, Cambridge
47. The P systems web page. http://ppage.psystems.eu

Hybrid Method for Simulating Small-Number Molecular Systems

Kazufumi Mizunuma and Masami Hagiya

Abstract Computational devices such as the toggle switch or the oscillator have recently been used in artificial or biological cells in which the number of molecular species is very small. To simulate their behavior, the stochastic simulation algorithm by Gillespie and the "τ-leap" method, also proposed by Gillespie to reduce simulation time, are widely used. In this paper, we discuss groups of cells that interact with the environment by exchanging molecules through their membranes. The stochastic simulation algorithm or even the "τ-leap" method requires a large amount of computation time because all the cells in the group and the environment need to be stochastically simulated. In this paper, we propose a hybrid simulation method in which molecular species in the environment are treated based on their concentration, and their time evolution is obtained by solving ordinary differential equations. The behavior of the cell group is then estimated by stochastically simulating some sampled cells. If the state of cells influences the environment, the whole simulation process is iterated until the time evolution of the environment becomes invariant. If the simulation ends after a few iterations, then the overall simulation time greatly decreases.

1 Introduction

In recent years, many attempts have been made to numerically simulate the behavior of a cell or a cell group by using system dynamics (systems biology) and computational devices such as the toggle switch or the oscillator, either in a natural or artificial cell (synthetic biology) [2, 5, 7]. To simulate biochemical reactions in such systems, models based on deterministic ordinary differential equations (ODEs) are generally applied [10]. In cells, however, due to the stochastic behavior of a small-population species, accurately predicting their fluctuations is not possible using such methods. The result of this is that a large amount of computational time is inevitably required for stochastic and accurate simulations. Thus, an obvious need exists for an effective method with higher speed and a minimal loss of accuracy.

Bistability is a fundamental behavior of a biological system that has been studied through numerous experiments and simulations [5]. Such a system has two steady

M. Hagiya (✉)
Graduate School of Information Science and Technology, University of Tokyo, Tokyo, Japan
e-mail: hagiya@is.s.u-tokyo.ac.jp

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_29, © Springer-Verlag Berlin Heidelberg 2009

states, and a small change in some molecular species in the population leads the system from one state to the other. Biological examples of bistable systems include the λ phage lysis-lysogeny switch [6], genetic toggle switches, including those found in quorum sensing [10], the lactose operon repressor system, cellular signal transduction pathways, and various systems of cell-cycle control. The key to studying such systems is to know the regulatory mechanisms that result in switching between states. Many experiments have indicated that noise plays an important role in the behavior of bistable systems [5, 8]. To date, stochastic modeling has shown that the effect of noise extensively affects the transition of states. Among the efforts in that direction, Tian et al. employed the Poisson τ-leap method in place of ODEs to simulate the behavior of some molecules in cells, and showed the effect of noise on state transition [10]. Their work also indicated gains in simulation speed obtained by the Poisson τ-leap method compared to Gillespie's original stochastic simulation algorithm (SSA). However, because of the large population of cells, stochastic modeling still requires a great deal of time to simulate the interaction between a cell and the environment. In particular, considering that each cell affects the environment, all the cells should be simulated together.

In this paper, we present a hybrid method to reduce the time for simulating bistable biological systems that interact with the environment, including the quorum sensing systems discussed by Tian et al.

We first deal with a case in which the interaction between a cell and the environment does not depend on the state of a cell. We made an approximation of the discrete numbers of molecules by assuming a continuous concentration. Because numerous cells exist in such a system, the random effects of cells are averaged. As a result, we can partition the simulation into two parts that of individual cells and that of their environment, and use the Poisson τ-leap method [4] for the former and the deterministic ODE method for the latter. In particular, each cell can be simulated separately with the environmental concentration simulated in the latter part.

If the state of each cell has an effect on its environment, accurate simulations require great computational time because all the cells should be simulated together. However, we can approximate the behavior of the environment using the results of simulating a small number of cells as a sample, i.e., by using the ratio of the cells that made a state transition and the distribution of the time of state transition. Using the result of these samples, we can obtain the continuous concentration of the environment as a function of time. With this result, the simulation of sample cells can be repeated, taking the new concentration of the environment into account. In principle, this process can be iterated until the environmental concentration converges. If the concentration converges after a small number of iterations, effective gains are achieved in simulation speed with a small sacrifice in simulation accuracy. In summary, our method can reproduce an approximate behavior of a cell group and the environment with a short simulation time.

A concrete example to which our method has been applied is the toggle switch analyzed by Tian et al. [10]. We modified by it introducing the dependency of acyl-homoserine lactone (AHL) production on cell states. We used the same experimental parameters as in Tian et al. [10].

The rest of this paper is organized as follows. We give a brief review of stochastic methods and simulations of toggle switches in Sect. 2, and present our methods with some examples in Sect. 3. Section 4 concludes the paper.

2 Background

In this section, we provide some background of the system used in our simulation: the stochastic simulation methods and simulations of toggle switches in a cell or a cell group.

2.1 Gillespie's Stochastic Simulation Algorithm

2.1.1 Original SSA

In this paper, we study the evolution of molecular numbers in an idealized cell, i.e., a well-stirred chemical reaction system. Gillespie proposed a method for the time evolution simulation of such a system [3]. This system has a fixed volume $\varOmega$ and contains N molecular species $\{S_1, \dots, S_N\}$, which interact chemically inside it at a constant temperature through reaction channels. We then describe the number of species S_i as $X_i(t)$ at time t. Note that our goal is to describe the time evolution of $\mathbf{X}(t) = [X_1(t), \dots, X_N(t)]^T = \mathbf{x}$ when we start the simulation with some given initial state $\mathbf{X}(t_0) = \mathbf{x}_0$.

For each reaction channel R_j $(j = 1, \dots, M)$, we define a probability density function $a_j(\mathbf{x})$ in a given state $\mathbf{X}(t)$, which is called *propensity function* and used with the form of $a_j(\mathbf{x})dt$ to represent the probability that one reaction R_j will occur somewhere inside $\varOmega$ in an infinitesimal time interval $[t, t + dt)$. We also define a state change vector ν_j whose element ν_{ij} represents the change in the number of species S_i due to the occurring reaction R_j. Hence, with these two functions, we can characterize the reaction channels.

During a stochastic simulation, we can determine the time and index of the next reaction. The stochastic simulation algorithm is the procedure for generating these randomly in accordance with the current values of the propensity functions and then recalculating the next state of the system.

In the so-called direct method, we use two independent random numbers r_1 and r_2, a sample of uniform distribution in the unit interval, and then take the time of the next reaction to be the current time plus μ, where

$$\mu = \frac{1}{a_0(\mathbf{x})} \ln\left(\frac{1}{r_1}\right).$$

Here, $a_0(\mathbf{x}) = \sum_{k=1}^{M} a_k(\mathbf{x})$, and the index of the next reaction is determined by the inequality, which the value of j satisfies

$$\sum_{k=1}^{j-1} a_k(\mathbf{x}) < r_2 a_0 \leq \sum_{k=1}^{j} a_k(\mathbf{x}).$$

Finally, the system is updated by

$$\mathbf{x}(t+\mu) = \mathbf{x}(t) + \nu_j.$$

We adopted the result of this method [3, 9] as reference to measure the exactness of the Poisson τ-leap method described in the next section.

2.1.2 τ-Leap Method

Our simulation model adopts the Poisson τ-leap method to achieve computational efficiency of the stochastic algorithm [4]. This method was also proposed by Gillespie for reducing the simulation time with only a small loss of accuracy.

The assumption of this method is almost the same as that of his original SSA: a well-stirred biochemical reaction system that contains N molecular species $\{S_1, \ldots, S_N\}$ inside a volume Ω, reaction channels $\{R_1, \ldots, R_M\}$ at a constant temperature, propensity function $a_j(\mathbf{x})$, and the element ν_j, which represents the change in the number of species due to reaction R_j. However, the evolution method of the system is different from the original SSA. Because step sizes in the original method tend to be very small, it requires a large simulation time. To overcome this drawback, we assume in the Poisson τ-leap method that a reasonably large number of reactions are firing in a well- adjusted time interval $[t, t+\tau)$. Moreover, between the time τ, the propensity function $a_j(\mathbf{x})$ is assumed to not change. In addition, the number of reactions in the channel R_j is sampled from a Poisson random variable $P(a_j(\mathbf{x})\tau)$ with the mean $a_j(\mathbf{x})\tau$. Finally, after generating a sample value for each reaction channel, the system is updated by

$$\mathbf{x}(t+\tau) = \mathbf{x}(t) + \sum_{j=1}^{M} \nu_j P\big(a_j(\mathbf{x})\tau\big).$$

By using this method, we can introduce a stochastic aspect into a simulation, in contrast to the ODE method [10].

2.2 Simulations of Toggle Switches

2.2.1 Toggle Switch with SOS Pathway

The toggle switch, the behavior of which Tian et al. simulated, consists of two genes, *lacI* and λcI, which encode the transcriptional regulator proteins LacR and λCI, respectively [2, 7, 10]. Because expression of one gene tends to repress the other, this system has two distinct bistable states. A state with low expression of $lacI$ when λcI has high a expression level and a state that is the exact converse. Transitions between these steady states can be induced by a signal that temporarily moves the system to an intermediate state.

Although a deterministic model has been proposed that realizes the existence of bistability in a genetic toggle switch, it does not agree with experimental results concerning the population distribution of cells in each state.

Based on this deterministic model, Tian et al. introduced a stochastic aspect into the model and analyzed the results using cells that show different genetic switching behaviors under the same experimental condition [10]. The system is updated by the generated Poisson samples given by

$$u(t+\tau) = u(t) + P\left[\varepsilon\left(\alpha_1 + \frac{\beta_1 K_1^3}{K_1^3 + v(t)^3}\right)\tau\right] - P\left[\left(d_1 + \frac{\gamma s}{1+s}\right)u(t)\tau\right],$$

$$v(t+\tau) = v(t) + P\left[\varepsilon\left(\alpha_2 + \frac{\beta_2 K_2^3}{K_2^3 + u(t)^3}\right)\tau\right] - P\big[d_2 v(t)\tau\big],$$

where u and v are the molecular numbers of λCI and LacR, respectively. In this system, α_1 is the basal synthesis rate of λCI and $\alpha_1 + \beta_1$ is its maximal rate. In addition, $\alpha_2 + \beta_2$ are the equivalent parameters for LacR expression. The term ε denotes the copy number of the toggle switch plasmid and s is related to the degradation of λCI when DNA in cells is damaged and the SOS pathway becomes active. Finally, d_2 is the constant degradation rate of LacR. To investigate the fluctuational behavior of the system, we reproduced the simulation by Tian et al. according to their parameters [10]. We adopted $\frac{1}{100}$ min as the time τ. The reaction data for this simulation are $\alpha_1 = \alpha_2 = 0.2$ μM·min^{-1}, $\beta_1 = \beta_2 = 4$ μM·min^{-1}, $\varepsilon = 1$, $d_1 = d_2 = 1$ min^{-1}, $K_1 = K_2 = 1$ μM, and $\gamma = 1$ min^{-1}. We estimated that ≈500 molecules per cell in *E. coli* corresponds to a concentration of 1 μM. The initial molecular numbers are $u(0) = 2,125$ and $v(0) = 125$ [7]. The degradation rate of λCI is set to $d_1 = 1$ when $t \in [0, 60]$ and $t \geq 960$, but $d_1 + \gamma s/(1+s)$ in $t \in [60, 960]$ because it is necessary to shift the system from the steady state to an intermediate state with large degradation of λCI [10].

The results of these simulations are shown in Fig. 1. Panel A gives the result of a deterministic simulation of successful switching ($s = 2.2$) and Panel B shows one of unsuccessful switching ($s = 1.7$). The different behaviors of the genetic switch under the same condition ($s = 1.7$) are shown in Panel C (successful switching) and Panel D (unsuccessful switching).

Panel A and B demonstrate that genetic switching will occur only if the number of λCI is below a threshold value attributable to its degradation.

Indeed, the deterministic model indicates that no switching takes place with the parameter $s = 1.7$, but switching does occur with the parameter $s = 2.2$. Tian et al. described that the boundary value of s is approximately 2.0 [10]. However, we observed slightly different results in the simulations using the stochastic model. Figure 1 gives the results of two simulations: a successful transition in Panel C and an

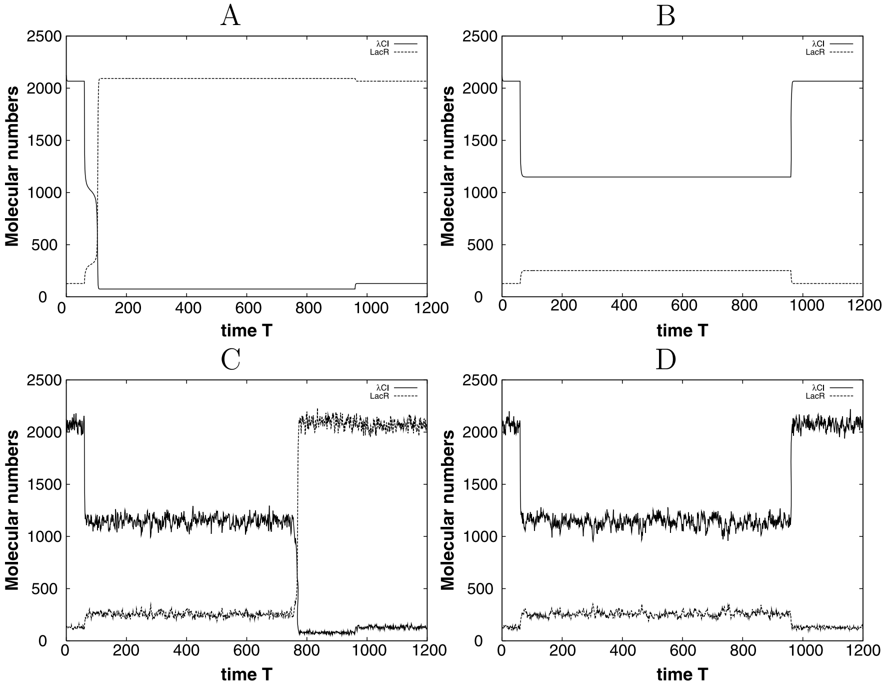


Fig. 1 Simulations of the toggle switch interfaced with the SOS signaling pathway

unsuccessful transition in Panel D based on the same degradation parameter $s = 1.7$. Because the decrease in λCI leads the system from the steady state with a high λCI expression level to an intermediate unstable state, the intrinsic noise brought about by introducing the Poisson random variables into ODEs determines if switching occurs.

Figure 2 shows the number of switched cells in stochastic simulations (using Gillespie's original method and the Poisson τ-leap method) based on different parameters s. Furthermore, the figure shows that to simulate stochastic models, we are able to reliably use the Poisson τ-leap method instead of the exact SSA method and reduce the computational time.

2.2.2 Toggle Switch with Quorum-Sensing Signaling Pathway

The toggle switch was also combined with a quorum-sensing signaling pathway, giving a different example of a stochastic simulation. In the following example of a quorum-sensing signaling pathway, the signal protein AHL determines the activity of each cell based on variation in the cell population densities of the culture.

The artificial network system Kobayashi et al. constructed consists of a sensor plasmid and the toggle switch plasmid [7]. As the signal molecule AHL diffuses between the culture and cells, variation in cell population densities influences the

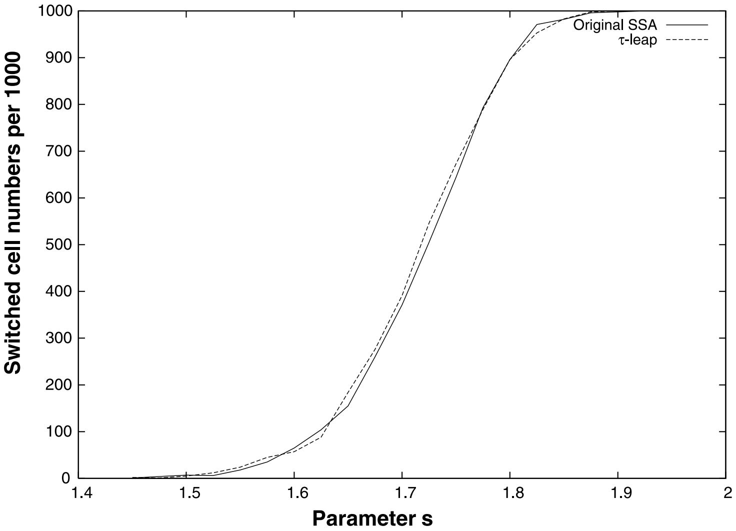


Fig. 2 Number of switched cells in stochastic simulations based on different degradation parameters s

sensor plasmid in each cell. In the toggle switch plasmid, the expression of LacR is negatively regulated by λCI, and that of λCI is negatively regulated by the total LacR from the sensor plasmid, resulting in the toggle switch behavior. The system is updated by the generated Poisson samples given by

$$x_{1i}(t+\tau) = x_{1i}(t) + P[b_1\tau] + P[\mu_1 k_1 \bar{x}_e \tau] - P[\mu_1 x_{1i}\tau] - P[x_{1i}d_1\tau],$$

$$x_{2i}(t+\tau) = x_{2i}(t) + P\left[\varepsilon_1\left(b_2 + \frac{\beta_1 x_{1i}^2}{K_1^2 + x_{1i}^2}\right)\tau\right] - P[x_{2i}d_2\tau],$$

$$x_{3i}(t+\tau) = x_{3i}(t) + P\left[\varepsilon_2\left(b_3 + \frac{\beta_2 K_2^3}{K_2^3 + x_{4i}^3}\right)\tau\right] - P[x_{3i}d_2\tau],$$

$$x_{4i}(t+\tau) = x_{4i}(t) + P\left[\varepsilon_2\left(b_4 + \frac{\beta_3 K_3^3}{K_3^3 + (x_{2i} + x_{3i})^3}\right)\tau\right] - P[x_{4i}d_3\tau],$$

$$x_e(t+\tau) = x_e(t) + \sum_{i=1}^{N}\big(P[\mu_2 k_2 x_{1i}\tau] - P[\mu_2 k_2 \bar{x}_e \tau]\big) - P[x_e d_e \tau],$$

where x_{1i} and x_e are the number of AHL molecules in the ith cell and in the culture, respectively; x_{2i} and x_{3i} are the numbers of LacR, which are from the sensor plasmid and the toggle switch plasmid, respectively; and x_{4i} is the number of λCI. In addition, ε_1 and ε_2 are the copy numbers of the sensor plasmid and the toggle switch plasmid, respectively. μ_1 and μ_2 represent the diffusion rates of AHL across the cell membrane, where $\mu_2 = \frac{k_1}{k_2}\mu_1$. $x_{ji} = x_{ji}(t)$ are numbers of molecules; $\bar{x}_e$ and $\bar{x}_{1i}$ are

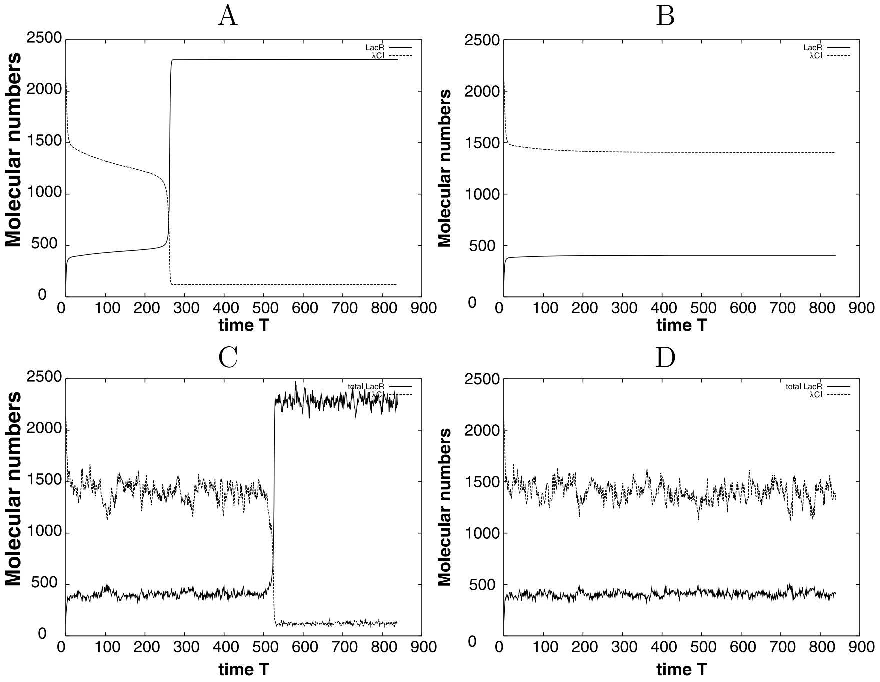


Fig. 3 Simulations of the toggle switch interfaced with the quorum-sensing signaling pathway

concentrations of x_e and x_1, respectively; and k_1 (=500) and k_2 ($= k_1 \cdot V_{\text{ext}}/V_c$) are factors for exchanging concentrations to molecular numbers in each cell and in the culture, respectively. (V_{ext} represents the total extracellular volume, and V_c is the volume of a cell) [1, 10]. Note that only AHL molecules are freely diffusible across the membrane.

We simulated the system when $t \in [0, 840]$ according to the data of Tian et al. under the initial condition $(x_{1i}, x_{2i}, x_{3i}, x_{4i}, x_e)\mid_{t=0} = (0, 0, 125, 2125, 0)$. The reaction data are $\varepsilon_1 = 0.5$ and $\varepsilon_2 = 1$; $b_1 = 0.45$ μM·min^{-1} and $b_2 = b_3 = b_4 = 0.2$ μM· min^{-1}; $\beta_1 = 0.97$ μM·min^{-1} and $\beta_2 = \beta_3 = 4$ μM·min^{-1}; $d_1 = d_2 = d_3 = 1$ min^{-1} and $d_e = 0.01$ min^{-1}; $K_1 = 0.11$ μM and $K_2 = K_3 = 1$ μM; $\mu_1 = 2$ min^{-1}; and the volume factor $V_{\text{ext}}/V_c = 3.5 \times 10^6$ [10], and the cell number $N = 3080$ or 10000 (only with ODEs).

Results of the simulations are shown in Fig. 3. Panel A gives the result of a deterministic simulation of successful switching ($N = 10000$) and Panel B shows that of unsuccessful switching ($N = 3080$). The different behaviors of a genetic switch under the same condition ($N = 3080$) are shown in Panel C (successful switching) and Panel D (unsuccessful switching). Here, although these two cells occur under the exact same condition, by introducing stochastic parameters, we could realize individualistic behaviors of cells in the same culture.

As in the toggle switch consisting of a single cell, by comparing Panel A with Panel B, we see that genetic switching happens only if the number of λCI is below

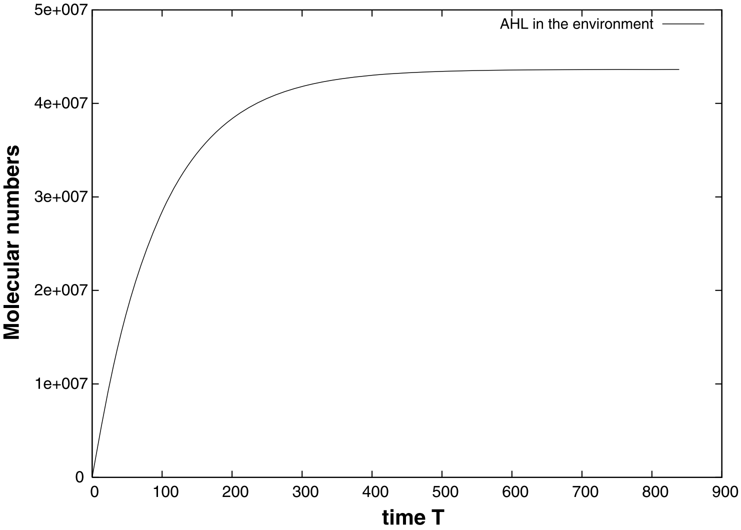


Fig. 4 Number of AHL molecules in the extracellular culture

a threshold value. In the deterministic model, no switching occurs with the number of cells below 9000, but in the stochastic model, as shown by Tian et al., almost all cells switch at around that number [10]. Figure 3 gives two simulations for a successful transition in Panel C, and an unsuccessful transition in Panel D based on the same environment in the culture. Because the decrease in λCI similarly leads each cell from the steady state with a high λCI expression level to an intermediate unstable state, the intrinsic noise brought about by introducing the Poisson random valuables into ODEs makes the difference as to whether switching occurs.

The number of AHL molecules in the extracellular environment is shown in Fig. 4. As previously described, this system is updated by the stochastic method. Because the behavior of the genetic switch in each cell is determined by the same environment, the most important factor for the switching population distribution is considered the number of AHL molecules in the environment, i.e., x_e. These results indicate that if we can achieve this simulation curve, we will be able to reproduce the behavior of each cell. Note that the curve is very smooth because the production of AHL from several cells is averaged.

3 Hybrid Simulation

In this section, we present two approaches to hybrid simulations for reducing the computation time of a stochastic simulation. We evaluated their accuracy and efficiency using the previous quorum-sensing model.

3.1 Methods

3.1.1 Basic Approach

First, we investigated a basic approach to hybrid simulations by using a toggle switch with a quorum-sensing signaling pathway described in the previous section. The toggle switch exists in each cell and is influenced by AHL, which diffuses in from the environment. Because AHL is assumed to be the only molecular species that can pass across cell membranes, the biological state of the cells is determined by the extracellular molecular number of AHL. In other words, if simulating the number of AHL molecules in a short time, we can easily predict the behavior of each cell.

Here, we assume that the number of AHL molecules in the environment (extracellular space in the culture) changes continuously, not randomly. The model is then given by

$$
\begin{aligned}
x_{1i}(t+\tau) &= x_{1i}(t) + P[b_1\tau] + P[\mu_1 k_1 \bar{x}_e \tau] - P[\mu_1 x_{1i}\tau] - P[x_{1i} d_1 \tau],\\
x_{2i}(t+\tau) &= x_{2i}(t) + P\left[\varepsilon_1\left(b_2 + \frac{\beta_1 x_{1i}^2}{K_1^2 + x_{1i}^2}\right)\tau\right] - P[x_{2i} d_2 \tau],\\
x_{3i}(t+\tau) &= x_{3i}(t) + P\left[\varepsilon_2\left(b_3 + \frac{\beta_2 K_2^3}{K_2^3 + x_{4i}^3}\right)\tau\right] - P[x_{3i} d_2 \tau],\\
x_{4i}(t+\tau) &= x_{4i}(t) + P\left[\varepsilon_2\left(b_4 + \frac{\beta_3 K_3^3}{K_3^3 + (x_{2i} + x_{3i})^3}\right)\tau\right] - P[x_{4i} d_3 \tau],\\
\frac{d\bar{x}_1}{dt} &= b_1 + \mu_1 \bar{x}_e - \mu_1 \bar{x}_1 - d_1 \bar{x}_1,\\
\frac{d\bar{x}_e}{dt} &= \mu_2 N(\bar{x}_1 - \bar{x}_e) - d_e \bar{x}_e,
\end{aligned}
$$

where the reaction parameters are the same as those of Tian et al. [10]. Note that the equation for $\bar{x}_e$ is changed to a continuous differential equation, and the equation for $\bar{x}_1$ is added. This variable $\bar{x}_1$ represents the average concentration of AHL in a cell.

Note that this approach is based on the observation in Sect. 2.2. Indeed, the number of diffused AHL molecules from each cell is randomly determined by a sample of the Poisson distribution. However, because large numbers of cells are living in the culture, all diffused AHL molecules are summed and the time evolution of their gross number is averaged.

3.1.2 Iterative Approach

Next, we extended this basic approach and applied the iterative method, which we also applied to the toggle switch with a quorum-sensing pathway. In the model Tian et al. simulated [10], AHL in the extracellular culture one-sidedly influenced the toggle switch to change its state. We hence assume that the state of the toggle switch reversely influences the environment. In short, we changed the production of AHL in a cell to be influenced by the state of the toggle switch; if a state transition occurs, with the degradation of λCI, the production rate of AHL (b_1) also decreases. Under such condition, we cannot use our basic approach as the behavior of each cell is different regardless of whether its toggle switch changes its state.

Except for the equation for $\bar{x}_1$, the new model is given by

$$\begin{aligned}
x_{1i}(t+\tau) &= x_{1i}(t) + P\left[\left(b + \frac{\beta_4 x_{4i}^3}{K_4^3 + x_{4i}^3}\right)\tau\right] + P[\mu_1 k_1 \bar{x}_e \tau] - P[\mu_1 x_{1i} \tau] \\
&\quad - P[x_{1i} d_1 \tau], \\
x_{2i}(t+\tau) &= x_{2i}(t) + P\left[\varepsilon_1\left(b_2 + \frac{\beta_1 x_{1i}^2}{K_1^2 + x_{1i}^2}\right)\tau\right] - P[x_{2i} d_2 \tau], \\
x_{3i}(t+\tau) &= x_{3i}(t) + P\left[\varepsilon_2\left(b_3 + \frac{\beta_2 K_2^3}{K_2^3 + x_{4i}^3}\right)\tau\right] - P[x_{3i} d_2 \tau], \\
x_{4i}(t+\tau) &= x_{4i}(t) + P\left[\varepsilon_2\left(b_4 + \frac{\beta_3 K_3^3}{K_3^3 + (x_{2i} + x_{3i})^3}\right)\tau\right] - P[x_{4i} d_3 \tau], \\
\frac{d\bar{x}_e}{dt} &= \mu_2 N(\bar{x}_1 - \bar{x}_e) - d_e \bar{x}_e,
\end{aligned}$$

where the reaction parameters are almost the same as those of Tian et al. [10], except that some new parameters are introduced: $b = 0.4$ is the minimum synthesis rate of AHL in a cell, $b + \beta_4$ is its basal rate with $\beta_4 = 0.07$, and $K_4 = 1$ µM.

For simulating the behavior of the system, we used the following approach. First, we introduced the deterministic ODE into the system, which calculates the average concentration of AHL in a cell given by

$$\frac{d\bar{x}_1}{dt} = \big(b + \beta_4\big(1 - \rho(t)\big)\big) + \mu_1 \bar{x}_e - \mu_1 \bar{x}_1 - d_1 \bar{x}_1,$$

where $\rho(t)$ is the ratio of cells whose toggle switch has undergone a state transition by the time t. Here, we approximated the synthesis rate $b + \frac{\beta_4 x_{4i}^3}{K_4^3 + x_{4i}^3}$ of AHL in a cell by $b + \beta_4$ in the original state of the toggle switch ($x_{4i} \gg K_4$) and by b after a state transition ($x_{4i} \ll K_4$). Initially, $\rho(t)$ was set to be constantly 0. Second, we simulated the system for a specific number of times. We adopted 100 as the number of times to get the time-population distribution $\rho(t)$ of the state transition of the toggle switch. The ratio $\rho(t)$ is approximated by $\frac{M(t)}{100}$, where $M(t)$ is the number of cells that changed the state of the toggle switch by t. Next, we performed a simulation that reflected the influence of the observed time-population distribution of the toggle switch. From this, we attained another time-population distribution of the toggle switch based on the previous simulation. We repeated this process until the time-population distribution no longer changed.

3.2 Results

3.2.1 Basic Approach

The results of the simulation are shown in Fig. 5. Two curves are shown: one according to the method of Tian et al. and the other based on our method.

Because the number of AHL molecules that diffuse from each cell is averaged for numerous cells ($N = 3080$), the curve based on our model is almost on top of the curve using the original method. From the perspective of computation time, we simulated a single cell and computed the time evolution of the extracellular AHL molecules in 6.6 seconds, whereas the original model took about 4 hours. The time-population distribution of cells can be approximately computed by stochastically simulating a cell for a reasonable number of times, for example, 100, less than the $N = 3080$ required for the time evolution of AHL.

3.2.2 Iterative Approach

The results of the simulation are shown in Fig. 6. Eight curves are shown: one by the original method and the others by our method.

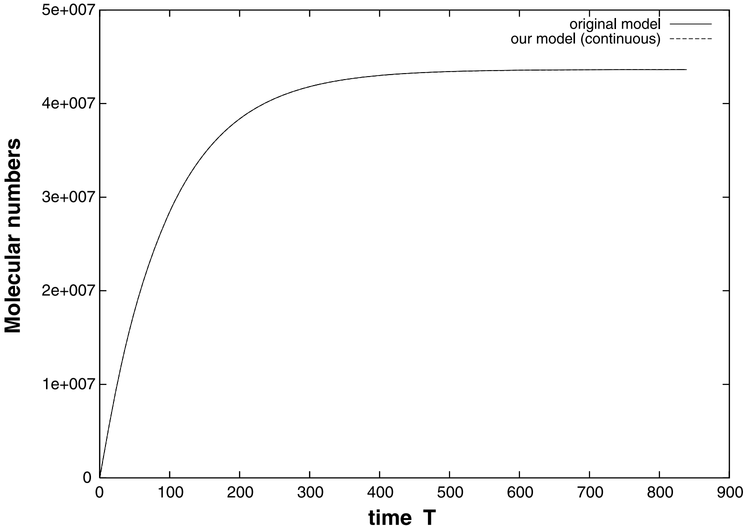


Fig. 5 Number of AHL molecules in the extracellular culture: original model and our basic model

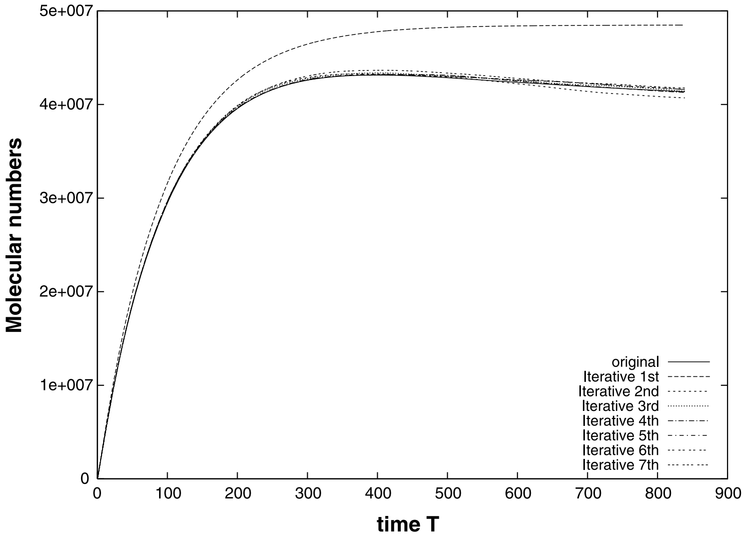


Fig. 6 Number of AHL molecules in the extracellular culture: original model and our iterative model (1st–7th)

As shown in the figure, we achieved almost the same curve as the original method after three or four iterations. The first curve was obtained with $M(t) = 0$; in other words, we assumed that all cells did not change the state of the toggle switch. The second curve, which is shown at the bottom, was lower than the original one because the computed number of AHL molecules was larger, and the transition of the toggle switch occurred in more cells in the first iteration. In the third and fourth iterations, the curves become almost stable, and with more iterations, the curves converge to the original one. From the viewpoint of computation time, we achieved the time evolution of the extracellular AHL molecules in 12.0 minutes on each iteration, whereas the original model took about 4 hours.

All simulation programs were written in Java. The platform we ran these programs on was a 1.3 GHz Pentium M PC with 0.99 GB memory.

4 Conclusions

In this study, we attempted to simulate the stochastic behavior of certain molecules in cells. In systems such as cells, because some species have a smaller number of copies, their behavior is not sufficiently averaged. However, in an experimental culture, for example, species that are large in number and those that are small in number are often mixed although their fluctuational behaviors are different. While we dealt with two different situations, our approaches were based on the same principle. We divided our simulation into two parts, one part in each cell and the other part in the environment, and then simulated them with the stochastic method and the continuous method. Our results demonstrated the superior potential of our technique.

Note that this study examined only the models based on Tian et al. Due to the severity of the parameters, we were unable to test many models. In the future, we plan to examine more models and parameters, and thus reveal the conditions by which we can simulate these models using our approach.

References

1. Garcia-Ojalvo J, Elowitz MB, Strogatz SH (2004) Modeling a synthetic multicellular clock: repressilators coupled by quorum sensing. Proc Natl Acad Sci USA
2. Gardner TS, Cantor CR, Collins JJ (2000) Construction of a genetic toggle switch in escherichia coli. Nature
3. Gillespie DT (1977) Exact stochastic simulation of coupled chemical reactions. J Phys Chem
4. Gillespie DT (2001) Approximate accelerated stochastic simulation of chemically reacting systems. J Phys Chem
5. Hasty J, McMillen D, Isaacs F, Collins JJ (2001) Intrinsic noise in gene regulatory networks. Nat Rev Genet
6. Isaacs FJ, Hasty J, Cantor CR, Collins JJ (2001) Prediction and measurement of an autoregulatory genetic module. Proc Natl Acad Sci USA
7. Kobayashi H, Kærn M, Araki M, Chung K, Gardner TS, Cantor CR, Collins JJ (2004) Programmable cells: interfacing natural and engineered gene networks. Proc Natl Acad Sci USA
8. Thattai M, van Oudenaaden A (2001) Intrinsic noise in gene regulatory networks. Proc Natl Acad Sci USA
9. Tian T, Burrage K (2004) Binomial leap methods for simulating stochastic chemical kinetics. J Phys Chem
10. Tian T, Burrage K (2004) Stochastic models for regulatory networks of the genetic toggle switch. Proc Natl Acad Sci USA

Part IX Broader Perspective

On Involutions Arising from Graphs

Jurriaan Hage and Tero Harju

Abstract We investigate various aspects of involutions of groups, i.e. anti-automorphisms of order at most two. The emphasis is on finite Abelian groups. We count the number of involutions for the cyclic groups, and consider the problem for direct products of groups. We also give a characterization for the set of skewed squares of finitely generated Abelian groups with identity as the involution. The present paper is motivated by our research into switching classes of combinatorial graphs where the edges have skew gains.

1 Introduction

Involutions occur naturally in biological processes especially in the base inversions of DNA. The material in this paper is mainly motivated not so much by the biological applications, but rather by a quest for techniques which enable analysis of networks of processors which can be modeled by actions of groups on finite combinatorial graphs. Such an action is presented by Seidel switching (see [10, 11] or [3, 5, 12]), and in a more general setting by switching in the so-called skew gain graphs (see [2, 4, 6]).

In the context of Seidel switching (of ordinary graphs), the confronted involution corresponds to the inversion of the underlying group. In the general setting of skew gain graphs, more flexibility is gained by having general involutions in the underlying group that acts on the graphs.

In the following, we consider involutions in groups without the graphs which employ them. In Sect. 3, we concentrate on cyclic groups, and then in Sect. 4, we consider involutions of direct products of finite groups. In Sect. 5, we consider a problem for infinite Abelian groups concerning involutions to which Hage [4] showed that the membership problem of switching classes can be reduced.

2 Preliminaries

We use $\mathbb{Z}$, $\mathbb{R}$ and $\mathbb{R}^+$ to refer to the sets of integers, real numbers and positive real numbers, respectively. The cardinality of a set X is denoted by $|X|$. The identity

J. Hage (✉)
Department of Information and Computing Sciences, Universiteit Utrecht,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
e-mail: jur@cs.uu.nl

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_30, © Springer-Verlag Berlin Heidelberg 2009

function on X is denoted by ι_X, from which the subscript X is omitted if it is clear from the context.

Let Γ be a group. For a function f, the set of its *fixed points* is $\mathrm{Fix}(\alpha) = \{a \in \Gamma \mid \alpha(a) = a\}$, and the set of its *inverted points* is $\mathrm{Inx}(\alpha) = \{a \in \Gamma \mid \alpha(a) = a^{-1}\}$.

A bijection $\alpha : \Gamma \to \Gamma$ is an *anti-automorphism*, if $\alpha(ab) = \alpha(b)\alpha(a)$ for all $a, b \in \Gamma$. For Abelian groups, an anti-automorphism is always an automorphism. An anti-automorphism α of the group Γ is an *involution*, if $\alpha^2 = \iota$, i.e., if α has order at most two.[1] Let $\mathrm{Inv}(\Gamma)$ denote the set of all involutions of Γ. We use $\#a$ to denote the order of a group element $a \in \Gamma$. The kernel of a group homomorphism $\alpha : \Gamma \to \Gamma'$ is the set $\ker(\alpha) = \{a \mid \alpha(a) = 1_{\Gamma'}\}$. The image of α is $\mathrm{Im}(\alpha) = \{b \in \Gamma' \mid b = \alpha(a), a \in \Gamma\}$. For other standard group notation, we refer to Rotman [8].

Example 1 Let $a \in \Gamma$, and let $\delta \in \mathrm{Inv}(\Gamma)$ be an involution. Then $\delta(1_\Gamma) = 1_\Gamma$, $\delta(a^n) = \delta(a)^n$ for integers $n \in \mathbb{Z}$, and $\#\delta(a) = \#a$. The inversion $a \mapsto a^{-1}$ is an involution of each group Γ.

Clearly, $\delta : \Gamma \to \Gamma$ is an involution if and only if the mapping $a \mapsto \delta(a)^{-1}$ is an automorphism of order at most two. Therefore, for instance, if Γ contains an element g with $\#g = 2$, then the mapping $a \mapsto (a^{-1})^g$ $(= ga^{-1}g^{-1})$ is an involution. This is the case among the finite groups γ of even order.

The following two results are proven, at least, in [6].

Lemma 1 *Let δ be an involution of a finite group Γ.*

1. *Either* $\mathrm{Fix}(\delta) \neq \{1_\Gamma\}$, *or δ is the inversion of Γ and Γ is of odd order.*
2. *Either* $\mathrm{Inx}(\delta) \neq \{1_\Gamma\}$, *or δ is the identity function and Γ is an Abelian group of odd order.*

Recall that the center of a group Γ is the normal subgroup $\mathrm{Z}(\Gamma) = \{x \in \Gamma \mid xy = yx \text{ for all } y \in \Gamma\}$.

Theorem 1 *The center $Z(\Gamma)$ of Γ is closed under every involution of Γ. In particular, if Γ has a nontrivial center, then for all involutions δ either $\delta(z) = z$ for all $z \in Z(\Gamma)$ or there exists an element $x \in Z(\Gamma)$ such that $\delta(x) = x^{-1}$ with $x \neq 1_\Gamma$.*

3 Involutions of Cyclic Groups

Each finite cyclic group is isomorphic to $\mathbb{Z}_n$ for some n, and by the fundamental theorem of Abelian groups, it can be written as a direct sum

$$\mathbb{Z}_n \cong \mathbb{Z}_{p_1^{m_1}} \oplus \mathbb{Z}_{p_2^{m_2}} \oplus \cdots \oplus \mathbb{Z}_{p_r^{m_r}}$$

[1] In the literature, an element of a group of order two is also called an involution.

of cyclic groups $\mathbb{Z}_{p_i^{m_i}}$, where $p_i \geq 2$ are distinct prime numbers such that $n = p_1^{m_1} p_2^{m_2} \dots p_r^{m_r}$.

Let $\delta \in \mathrm{Inv}(\mathbb{Z}_n)$, and suppose $\delta(1) = k$. Now, $\delta(i) \equiv ik \pmod{n}$ and so $1 = \delta(k) \equiv k^2 \pmod{n}$, i.e.,

$$k^2 \equiv 1 \pmod{n}. \tag{1}$$

Example 2 Let then $n = 16$. If (1) holds for k then it also holds for $n - k$, so $k = 15$ is also possible. It is easy to see that if $k > 1$, then $k \geq \sqrt{n+1}$, and hence k cannot be equal to 2, 3 or 4. From Example 1, we know that $\#1 = \#k$, and hence k generates $\mathbb{Z}_n$ as well, implying $(n, k) = 1$. This means that other possible values for k that have to be examined are 5 and 7. Of these, only 7 works (and thus also 9).

Let $\xi : \mathbb{N} \to \{-1, 0, 1\}$ be defined by

$$\xi(n) = \begin{cases} 1 & \text{if } 8|n, \\ -1 & \text{if } 2|n \text{ and } 4 \not| n, \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 2 *If $n = p^m$ for a prime p and $m \in \mathbb{N}$, then* (1) *has exactly* $2^{1+\xi(n)}$ *solutions k with* $1 \leq k \leq n - 1$.

Proof The two solutions $k = 1$ and $k = n - 1$ work for any cyclic group. In the case $p = 2$ and $m = 1$, these give the same unique solution.

Assume that (1) holds for $k \notin \{1, n - 1\}$. Now, $p^m | (k - 1)(k + 1)$, and thus $p = 2$; otherwise, $p^m = k + 1$ (because $k < n$) and so $k = n - 1$. Suppose $2^i | k - 1$ and $2^j | k + 1$ with $i + j = m$. We have $0 \leq i < m$.

If $i = 0$, and thus $j = m$ then $n = k + 1$; a contradiction. Suppose then that $i > 0$ in which case also $j > 0$. Let $r = \min(i, j)$. Then $2^r | (k + 1) - (k - 1) = 2$, i.e., $r = 1$. If $r = i$, then $j = m - 1$, and so $2^{m-1} | k + 1$. If $r = j$, then $i = m - 1$, and so $2^{m-1} | k - 1$. Hence, in both cases,

$$k \equiv \pm 1 \bmod 2^{m-1}. \tag{2}$$

For $m = 1$ and $m = 2$, we obtain the same solutions as in the above, and for $m \geq 3$, we have two solutions.

Summarizing, we have one solution if $p = 2$ and $m = 1$, four solutions if $p = 2$ and $m \geq 3$ and two if $p \neq 2$ or $p = 2$ and $m = 2$.

In the above, we have only proven half of what we need to prove, namely indicating possible solutions. The fact that these solutions do indeed always exist can be verified easily from (1).

Hence, for every n, the number of solutions equals $2^{1+\xi(n)}$. □

Theorem 2 *Let $n = p_1^{m_1} p_2^{m_2} \dots p_r^{m_r}$ be a prime decomposition of n with $p_i < p_{i+1}$, and $m_i > 0$ for $1 \leq i \leq r - 1$. Then* $|\mathrm{Inv}(\mathbb{Z}_n)| = 2^{r+\xi(n)}$.

Proof Lemma 2 can be applied to each $p_j^{m_j}$, with $1 \leq j \leq r$. We can then use Theorem 122 in the book of Hardy and Wright [7] to conclude that the total number of solutions equals the product of the numbers of solutions to the separate equations for $p_j^{m_j}$ for $1 \leq j \leq r$.

If $p_1 \neq 2$, then every prime number gives two solutions yielding a total of 2^r solutions, and indeed in this case $\xi(n) = 0$. If $p_1 = 2$, then we have three possibilities: $m_1 = 1$, and hence p_1 yields only one solution, and hence we have a total of 2^{r-1} solutions. The other two cases, $m_1 = 2$ and $m_1 > 2$, follow similarly, because now $\xi(n) = \xi(p_1^{m_1})$. □

Note that the involutions can be found by solving two sets of equations using the Chinese remainder theorem; see [7].

4 Involutions of Group Products

In the previous section, it turned out that finding involutions of cyclic groups is rather easy. In this section, it is shown that computing the involutions of a direct product of two groups involves taking the Cartesian product of the sets of involutions of both groups.

We remind (see, e.g., Rotman [8]) that the inner and outer direct products of groups coincide up to isomorphism, i.e., for a group Γ, if Γ_1 and Γ_2 are two of its normal subgroups such that $\Gamma = \Gamma_1\Gamma_2$ and $\Gamma_1 \cap \Gamma_2 = \{1_\Gamma\}$, then $\Gamma \cong \Gamma_1 \times \Gamma_2$. If $\Gamma = \Gamma_1\Gamma_2$ is a direct product, then $a_1a_2 = a_2a_1$ for all $a_1 \in \Gamma_1$ and $a_2 \in \Gamma_2$, and, moreover, each element $a \in \Gamma$ has a unique representation as a product $a = a_1a_2$, where $a_i \in \Gamma_i$.

Let $\Gamma = \Gamma_1\Gamma_2$ be a direct product, and let $\alpha : \Gamma \to \Gamma$ be any function. We define the *projections* $\alpha^{(i)} : \Gamma \to \Gamma_i$ for $i = 1, 2$ by: for each $a \in \Gamma$ let $\alpha(a) = \alpha^{(1)}(a)\alpha^{(2)}(a)$, where $\alpha^{(1)}(a) \in \Gamma_1$ and $\alpha^{(2)}(a) \in \Gamma_2$. By the uniqueness property of direct products, these functions are well defined. We also write

$$\delta^{[i]} : \Gamma_i \to \Gamma_i$$

for the restriction of $\delta^{(i)}$ onto the subgroup Γ_i for $i = 1, 2$.

The following example shows that an involution of a direct product cannot necessarily be obtained by projections of its components.

Example 3 Let $\Gamma = \Gamma_1 \times \Gamma_1$ for a group Γ_1, and let δ be the reversed inversion on Γ, that is,

$$\delta(a_1, a_2) = \big(a_2^{-1}, a_1^{-1}\big)$$

for all $a_1, a_2 \in \Gamma_1$. Then δ is an involution of Γ. Indeed, it is clear that $\delta^2 = \iota$, and, moreover, for all $a_i, b_i \in \Gamma_1$,

$$\begin{aligned}\delta(a_1,a_2)\cdot\delta(b_1,b_2) &= \big(a_2^{-1},a_1^{-1}\big)\big(b_2^{-1},b_1^{-1}\big) = \big(a_2^{-1}b_2^{-1},a_1^{-1}b_1^{-1}\big)\\ &= \big((b_2a_2)^{-1},(b_1a_1)^{-1}\big) = \delta(b_1a_1,b_2a_2)\\ &= \delta\big((b_1,b_2)\cdot(a_1,a_2)\big).\end{aligned}$$

However, δ is not of the form $\delta=(\delta_1,\delta_2)$ for any functions (let alone involutions) δ_1 and δ_2 of Γ_1, if Γ_1 is nontrivial.

However, we do have

Theorem 3 *Let $\Gamma=\Gamma_1\Gamma_2$ be a direct product.*

1. *If $\delta_i\in\mathrm{Inv}(\Gamma_i)$ for $i=1,2$, then the function $\delta:\Gamma\to\Gamma$ defined by*

$$\delta(a)=\delta_1(a_1)\delta_2(a_2)\quad \textit{for } a=a_1a_2 \textit{ with } a_i\in\Gamma_i$$

   is an involution of Γ.
2. *If $\delta\in\mathrm{Inv}(\Gamma)$, then there are normal subgroups Δ_1 and Δ_2 of Γ such that $\Gamma=\Delta_1\Delta_2$ is a direct product with $|\Delta_1|=|\Gamma_1|$, $|\Delta_2|=|\Gamma_2|$ for which $\delta^{[i]}:\Delta_i\to\Delta_i$ is an involution of Δ_i for $i=1,2$.*

Proof In order to prove (i), let δ_i be involutions as stated. Let $a=a_1a_2$ and $b=b_1b_2$ for $a_1,b_1\in\Gamma_1$ and $a_2,b_2\in\Gamma_2$. Now, for the function δ as defined in the claim,

$$\begin{aligned}\delta(ab) &= \delta(a_1a_2b_1b_2)=\delta(a_1b_1a_2b_2)=\delta_1(a_1b_1)\delta_2(a_2b_2)\\ &= \delta_1(b_1)\delta_1(a_1)\delta_2(b_2)\delta_2(a_2)=\delta_1(b_1)\delta_2(b_2)\delta_1(a_1)\delta_2(a_2)\\ &= \delta(b_1b_2)\delta(a_1a_2)=\delta(b)\delta(a)\end{aligned}$$

and thus δ is an anti-automorphism of Γ. Further, the condition $\delta^2(a)=a$ is easily checked.

For (ii), suppose first that $\delta\in\mathrm{Inv}(\Gamma)$, and define

$$\Delta_1=\{\delta(a)\mid a\in\Gamma_1\}\quad\text{and}\quad\Delta_2=\{\delta(b)\mid b\in\Gamma_2\}.$$

Clearly, $a\in\Delta_1$ (resp. in Δ_2) if and only if $\delta(a)\in\Gamma_1$ (resp. $\delta(a)\in\Gamma_2$). Since an involution is a bijection, we have immediately that $|\Delta_i|=|\Gamma_i|$ for $i=1,2$.

We show then that Δ_1 and Δ_2 are normal subgroups of Γ. Indeed, let $y=aua^{-1}$ for some $a\in\Gamma$ and $u\in\Delta_1$. Now, $\delta(y)=\delta(a)^{-1}\delta(u)\delta(a)\in\Gamma_1$, since $\delta(u)\in\Gamma_1$ and Γ_1 is a normal subgroup of Γ. This shows that Δ_1 is normal in Γ. The case for Δ_2 is symmetric.

Next, we observe that $\Delta_1\cap\Delta_2=\{1_\Gamma\}$ is the trivial subgroup of Γ. Furthermore, if $a\in\Gamma$, then $a=a_2a_1$ for some $a_i\in\Gamma_i$, because $\Gamma=\Gamma_2\Gamma_1$. Therefore, $\delta(a)=\delta(a_1)\delta(a_2)$, where $\delta(a_1)\in\Delta_1$ and $\delta(a_2)\in\Delta_2$. Since each element $b\in\Gamma$ is an image $b=\delta(a)$, we have shown that $\Gamma=\Delta_1\Delta_2$ is a direct product of Γ.

It is clear that $\delta^{[i]}$ is an involution of Δ_i for both $i=1$ and $i=2$. □

In particular, if Γ is an Abelian group, then it is a direct product (sum) of cyclic groups, and thus Theorem 4 states that the involutions of an Abelian group can be obtained from the cyclic groups $\mathbb{Z}_{p^k}$ that are its direct components. However, counting the number of involutions of Γ is *not* reduced in this way to the number of involutions of its direct components, because part (ii) of Theorem 4 uses 'swappings' of subgroups.

Example 4 Let $\Gamma = \mathbb{Z}_2 \oplus \mathbb{Z}_2$. The groups Γ and $\mathbb{Z}_2$ have only the identity function ι as their involution (equal to the group inversion), but in the case of the former, it is not the only one. Indeed, the following *swapping function* δ is an involution of Γ:

$$\delta((a,b)) = (b,a) \quad \text{for } a, b \in \{0,1\}.$$

5 The set $\Delta^2(\Gamma, \delta)$

In [4], the following set of decomposable values arose (called the skewed squares in [5]): for a group Γ and its inversion δ, let

$$\Delta^2(\Gamma, \delta) = \{a \in \Gamma \mid a = b\delta(b) \text{ for some } b \in \Gamma\}.$$

In this section, we investigate this set in more detail.

First, in the following proof, we have a connection of the above set with decompositions of Abelian p-groups.

Theorem 4 *Let Γ be a finite Abelian group of odd order, and let $\delta \in \mathrm{Inv}(\Gamma)$. Then Γ is isomorphic to the direct sum* $\mathrm{Fix}(\delta) \times \mathrm{Inx}(\delta)$.

Proof It is well known that in each Abelian group of odd order every element $a \in \Gamma$ has a unique "square root" x in Γ, i.e., $x^2 = a$, when adopting the multiplicative notation; see Rotman [9, p. 81]. Now, for each $a \in \Gamma$, $a = xy$ holds for some $x \in \mathrm{Fix}(\delta)$ and $y \in \mathrm{Inx}(\delta)$ if and only if $a^{-1}x = y^{-1} = \delta(y) = \delta(a)\delta(x^{-1}) = \delta(a)x^{-1}$ if and only if $x^2 = a\delta(a)$ and $y^2 = (a\delta(a))^{-1}$. This proves the claim since $\mathrm{Fix}(\delta) \cap \mathrm{Inx}(\delta) = \{1_\Gamma\}$. □

Example 5 If δ is the group inversion, then clearly $\Delta^2(\Gamma, \delta) = \{1_\Gamma\}$. Also, it is easy to determine that $\Delta^2(\mathbb{Z}, \iota)$ is the set of even numbers, $\Delta^2(\mathbb{R}, \iota) = \mathbb{R}$ for the additive group of reals, and $\Delta^2(\mathbb{R}^+, \iota) = \mathbb{R}^+$ for the multiplicative group of positive real numbers.

Given a fixed group Γ with an involution δ, we define the function $s_{\Gamma,\delta}$ by

$$s_{\Gamma,\delta}(a) = a\delta(a) \quad \text{for } a \in \Gamma$$

so that we have $\mathrm{Im}(s_{\Gamma,\delta}) = \Delta^2(\Gamma, \delta)$. When Γ and δ are obvious from the context we write s instead of $s_{\Gamma,\delta}$.

Lemma 3 *For any group Γ, $\Delta^2(\Gamma, \delta)$ is closed under the group inversion, and $\Delta^2(\Gamma, \delta) \subseteq \mathrm{Fix}(\delta)$.*

Proof Let $s(a) \in \Delta^2(\Gamma, \delta)$. Then $s(a)^{-1} = \delta(a)^{-1}a^{-1} = s(\delta(a)^{-1}) \in \Delta^2(\Gamma, \delta)$. For the second part, $\delta(s(a)) = \delta(a\delta(a)) = a\delta(a) = s(a)$. □

Because involutions of a direct product do not always project onto the factors, it is unlikely that we can determine the skewed squares of a group Γ with involution δ from the skewed squares of the groups in a decomposition of Γ, see Example 3. However, we do have that involutions δ_1 and δ_2 of groups Γ_1 and Γ_2, respectively, can be used to construct an involution (δ_1, δ_2) for $\Gamma_1 \times \Gamma_2$ by applying them componentwise. For involutions, thus constructed the set of skewed squares can be constructed from the sets of skewed squares of the factors as proved by the following result.

Theorem 5 *Let $\Gamma = \Gamma_1 \times \Gamma_2$ be a direct product of groups and let $\delta_i \in \mathrm{Inv}(\Gamma_i)$ for $i = 1, 2$. Then $\Delta^2(\Gamma, (\delta_1, \delta_2)) = \Delta^2(\Gamma_1, \delta_1) \times \Delta^2(\Gamma_2, \delta_2)$.*

Proof It holds that

$$(a, b)(\delta_1, \delta_2)(a, b) = (a, b)\big(\delta_1(a), \delta_2(b)\big) = \big(a\delta_1(a), b\delta_2(b)\big),$$

where $a\delta_1(a) \in \Delta^2(\Gamma_1, \delta_1)$ and $b\delta_2(b) \in \Delta^2(\Gamma_2, \delta_2)$. □

6 Example: The Case of the Identity Involution

In the rest of the article, we assume that δ is the identity function, ι. Note that ι is an involution of every Abelian group and no other, so Γ must be Abelian. Written additively, the definition of Δ^2 reduces to

$$\{a \in \Gamma \mid a = 2b \text{ for some } b \in \Gamma\}.$$

So, in a sense, Δ^2 contains the "even elements" of the group Γ.

Example 6 We can easily verify that $\Delta^2(\mathbb{Z}_2, \iota) = \{0\}$, $\Delta^2(\mathbb{Z}_3, \iota) = \{0, 1, 2\}$, $\Delta^2(\mathbb{Z}_4, \iota) = \{0, 2\}$, and $\Delta^2(\mathbb{Z}_6, \iota) = \{0, 2, 4\}$.

From this example, it emerges that for even n, $\Delta^2(\mathbb{Z}_n, \iota)$ contains exactly the even numbers and for odd n, it equals the entire Γ. The latter is not surprising since if $x \in \Gamma$ has order n and $(m, n) = 1$, then x is divisible by m.

It is plain that $\Delta^2(\mathbb{Z}_{2^k}, \iota) = \{0, 2, 4, 6, \ldots, 2^{k-1}\}$, for $k \geq 1$, and $\Delta^2(\mathbb{Z}_{p^k}, \iota) = \mathbb{Z}_{p^k}$ where $p > 2$ is prime and $k \geq 1$, and $\Delta^2(\mathbb{Z}, \iota)$ contains the even numbers and so by Theorem 5 and the fundamental theorem of finitely generated Abelian groups we get the following result.

Theorem 6 *Let Γ be a finitely generated Abelian group with a decomposition $\Gamma_1 \oplus \cdots \oplus \Gamma_n$ (unique up to the order of the summands) into infinite cyclic groups and cyclic p-groups. Then $\Delta^2(\Gamma, \iota) = \Delta^2(\Gamma_1, \iota) \oplus \cdots \oplus \Delta^2(\Gamma_\ell, \iota)$.*

The previous theorem says nothing about Abelian groups that are not finitely generated such as $\mathbb{R}^+$ under multiplication. To deal with $\mathbb{R}^+$, we recall the notion of divisibility: a group is divisible if for every element a and $n > 0$, we have $a = b^n$ for some b. Note that the set Δ^2 is only concerned with divisibility by two. Therefore, the following result is easy.

Theorem 7 *If Γ is divisible, then $\Delta^2(\Gamma, \iota) = \Gamma$.*

A result such as this suggests that it may be worthwhile to investigate other decompositions of groups, into divisible and reduced components, or into torsion and torsion free components.

Acknowledgement We thank professor R. Tijdeman and W. Kosters for discussions on the involutions of cyclic groups.

References

1. Corneil DG, Mathon RA (eds) (1991) Geometry and combinatorics: selected works of J.J. Seidel. Academic Press, Boston
2. Ehrenfeucht A, Hage J, Harju T, Rozenberg G (2004) Embedding in switching classes with skew gains. In: Ehrig H, Engels G, Parisi-Presicce F (eds) Graph transformations, second international conference, ICGT 2004. Lecture notes in computer science, vol 3256. Springer, Berlin, pp 257–270
3. Ehrenfeucht A, Harju T, Rozenberg G (1999) The theory of 2-structures. World Scientific, Singapore
4. Hage J (1999) The membership problem for switching classes with skew gains. Fundam Inform 39(4):375–387
5. Hage J (2001) Structural aspects of switching classes. PhD thesis. Leiden Institute of Advanced Computer Science. http://www.cs.uu.nl/people/jur/2s.html
6. Hage J, Harju T (2000) The size of switching classes with skew gains. Discrete Math 215:81–92
7. Hardy GH, Wright EM (1983) An introduction to the theory of numbers, 5th edn. Oxford Science Publications, Hong Kong
8. Rotman JJ (1973) The theory of groups, 2nd edn. Allyn and Bacon, Boston
9. Rotman JJ (2002) Advanced modern algebra. Pearson Education, Upper Saddle River
10. Seidel JJ (1976) A survey of two-graphs. In: Intern coll teorie combinatorie, Roma, 1973, vol I. Accad Naz Lincei, Rome, pp 481–511. Reprinted in [1]
11. Seidel JJ, Taylor DE (1981) Two-graphs, a second survey. In: Lovasz L, Sós V.T. (eds) Algebraic methods in graph theory, proceedings of the international colloqium, Szeged, 1978, vol II. North-Holland, Amsterdam, pp 689–711. Reprinted in [1]
12. Zaslavsky T (1989) Biased graphs, I: bias, balance, and gains. J Comb Theory, Ser B 47:32–52

Parallel Computing by Xeroxing on Transparencies

Tom Head

Abstract We illustrate a procedure for solving instances of the Boolean satisfiability (SAT) problem by xeroxing onto transparent plastic sheets. Suppose that m clauses are given in which n variables occur and that the longest clause contains k literals. The associated instance of the SAT problem can be solved by using a xerox machine to form only $n + 2k + m$ successive transparencies. The applicability of this linear time algorithm is limited, of course, by the increase in the information density on the transparencies when n is large. This same scheme of computation can be carried out by using photographic or other optical processes. This work has been developed as an alternate implementation of procedures previously developed in the context of aqueous (DNA) computing.

1 Introduction

This article records a talk given at the *Algorithmic Bioprocesses* meeting in Leiden in 2007. The brief style of the talk is adhered to, but the interested reader may see [5] for an alternate presentation of the fundamental concepts introduced here.

Suppose that we have a xerox machine that "reads" as input a rectangle of width w and height h and "writes" as output a rectangle of these same dimensions. We use transparent plastic sheets as the medium from which the input is read and as the medium onto which the output is written. Computing is made possible through iteration of such reading and writing by allowing the transparency that is read to be overlaid by other transparencies. We also allow the use of masking strips.

In a classical paper, Cook [1, 2] demonstrated the centrality of the algorithmic problem of deciding whether, for a given list of clauses, a truth setting can be found for the Boolean variables occurring in the clauses that results in each of the clauses having the value True. This is the problem we treat here.

We suppose that m clauses are given, that the maximum number of literals occurring in any of the clauses is k, and that the number of distinct variables occurring in the set of clauses is n. We suggest that the reader imagine the tactile experience of actually handling such transparencies as they are laid onto a xerox machine. We are confident that the essence of the procedure proposed for solving (in principle) any satisfiability (SAT) problem will be clear once we have illustrated such a solution

T. Head (✉)
Mathematical Sciences, Binghamton University, Binghamton, NY 13902-6000, USA
e-mail: tom@math.binghamton.edu

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_31, © Springer-Verlag Berlin Heidelberg 2009

for a case in which $m = 4$, $n = 3$ and $k = 3$. We treat the specific case in which the four clauses are: (1) $p \vee q$, (2) $p' \vee q \vee r'$, (3) $q' \vee r'$, and (4) $p' \vee r$. The traditional SAT question in this case is: Does a truth value assignment exist for p, q, r such that the four given clauses have the value True? We have chosen this example for two reasons: It is simple enough to be treated in complete detail. It is the same example that we treated previously by aqueous computing [3, 8] which allows a detailed comparison of xerographic computing and aqueous computing.

The computation is divided into two phases—each having its particular technique of parallelization. The table of all 2^n truth settings is constructed in n steps is Sect. 2. In Sect. 3, it is shown that by using at most $2k$ copies of the truth table, each clause can be evaluated at *every row* of the truth table *simultaneously*. In Sect. 4, it is then shown how, using an appropriate *negative* xerox copy associated with each of the m clauses, to complete the solution of the satisfiability problem using $n + 2k + m$ transparencies. In Sect. 5, we indicate a major (lateral) extension of our xerox computing method. Computing through xerography was born from a long term dream of using light as the tool for "writing on molecules" in aqueous computing [3]. Finally, the dawn: why not write with light on photographic film, or having no photo-lab, a xerox machine?

2 Constructing an *n* Variable Truth Table by Xeroxing onto Only *n* Transparencies

Lines of a truth table for the ordered triple of Boolean variables p, q, r will consist of triples of absolutely *opaque* squares, ■, and absolutely *transparent* squares. We indicate the location of each transparent square by writing, _. Here, an opaque square ■ denotes the truth value True and a transparent square _ denotes the value False. Thus, when occurring as rows in a truth table, ■ _ ■ expresses: $p =$ True, $q =$ False, $r =$ True and _ _ _ expresses: $p = q = r =$ False. The identifications ■ = True and _ = False are the appropriate choices when OR statements are to be evaluated as we shall see in Sect. 3. The reverse identifications are appropriate when AND statements are to be evaluated as we shall see in Sect. 4. The first line of the truth table is initialized with a black band of thickness $h/2^3$ across the top edge of the transparency. This initial transparency is denoted here by the display:

p, q, r
■■■
7 rows

where the notation "7 rows" reminds that the transparency has a still transparent region lying below the black band that is adequate to store seven more rows.

From this initial transparency, the complete truth table is constructed as follows:

■■■ 7 rows	_■■ 7rows	Each display on the right is obtained from the neighboring display at its left by xeroxing the left display with the appropriate column masked.

```
■ ■ ■      ■ _ ■
_ ■ ■      _ _ ■
6 rows     6 rows
```

Each new display at the left is created by laying the transparency at the right (above) under the transparency at its left (above) in the appropriate manner.

```
■ ■ ■      ■ ■ _
_ ■ ■      _ ■ _
■ _ ■      ■ _ _
_ _ ■      _ _ _
4 rows     4 rows
```

```
■ ■ ■
_ ■ ■
■ _ ■
_ _ ■
■ ■ _
_ ■ _
■ _ _
_ _ _
```

With $n = 3$ xeroxing operations, as indicated above, the complete $n = 3$ column $2^n = 8$ row truth table has been produced. This method applies in the general case to produce, for any positive integer n, the n variable table. Practicality is, of course, challenged by sufficiently large n. This limitation may be expected to continue through the future even though the precision of xerography, photography, or *other applicable optical processes* may be greatly increased. Perhaps it will eventually prove not to exclude practicality in significant cases.

3 Evaluating a Clause over All Truth Settings Simultaneously

Form $k = 3$ identical xerox copies of the truth table constructed in Sect. 2. The clause $p \vee q$ can be evaluated at each of the lines of the truth table in Sect. 2 by choosing two of the three copies of the truth table and laying the column headed by p on the first copy over the column headed by q on the second copy. This gives the complete truth column for $p \vee q$. For use in Sect. 4, record this column as follows: Make a *negative* copy of this column of values on a transparency after masking all columns other than the column for $p \vee q$ using an opaque mask. (Yes, xerox machines *do* make *negative* as well as *positive* copies.)

To be prepared to treat clauses, for example, $p' \vee q' \vee r'$, in which literals involving negations occur, form $k = 3$ identical *negative* copies of the truth table constructed in Sect. 2. Thus, we evaluate $p' \vee q \vee r'$ by using two of the negative copies and one of the positive copies of the truth table and bringing the columns for p', q, and r' into coincidence. Finally, after masking all columns other then this single triple layer column, a negative copy of this column is made. Truth columns for the remaining clauses can be made and recorded in this same way.

4 Completing the Solution

How are the negative copies of the value columns for the four clauses produced in Sect. 3 used to complete the solution? Below are the four *negative* columns followed by a column obtained from the first four by overlaying them. The original truth table is also copied below:

$p \vee q$	$p' \vee q \vee r'$	$q' \vee r'$	$p' \vee r$	Solution?	Truth Table
_	_	■	_	■ No	■ ■ ■
_	_	■	_	■ No	_ ■ ■
_	■	_	_	■ No	■ _ ■
■	_	_	_	■ No	_ _ ■
_	_	_	■	■ No	■ ■ _
_	_	_	_	_ YES	_ ■ _
_	_	_	■	■ No	■ _ _
■	_	_	_	■ No	_ _ _

By taking the negatives of the value columns of the four clauses, we have effectively reversed the significations of ■ and _. The presence of ■ in the value column of a clause now signifies that the clause is False and the presence of _ signifies that the clause is True. Thus, when these value columns are overlaid, there will exist a solution of the related SAT problem if and only if there appears in the overlaid column an occurrence of _. Moreover, when solutions exist they can all be read immediately by laying the overlaid column (headed: Solution?) beside the original truth table. In the present case, we see above that there is exactly one solution of this instance of the SAT problem and that the truth settings that give this solution are: $p =$ False, $q =$ True, $r =$ False.

5 Extending the Scope and Observing the Flexibility of Computing by Xerography

In the construction of the n variable truth table in Sect. 2, we concatenated each newly created transparency at the foot of the previous. A second option exists: the new transparency can be set adjacent to the previous one. Our meaning here will be immediately clear on examining one simple example. For an $n = 4$ variable SAT, having variables p, q, r, s, we display at the left below the result of the first three xerox and concatenate steps. The appropriately masked fourth xeroxing step is displayed, *not* below but *adjacent* at the right. We say that we have displayed the four variable truth table in *two segments*. This gives a display having width $4 \times 2 = 8$ columns and $2^3 = 8$ rows. One should take a moment to realize that evaluation of clauses works in the multi-segment case just it does in the single segment case treated in Sect. 3. The only difference is that the truth values of clauses also occur in similar segments. One need only imagine computing the truth values for $p \vee q \vee s$ by lining up, by hand, the columns for p, q, *and* s in the *first* segments of three copies of the two-segment truth table and realizing that the three appropriate columns of the *second* segment are thereby *automatically* correctly lined up. The reader may wish to imagine the appropriate handling of one positive copy and two negative copies of the truth table that result in the truth column(s) for $p' \vee r \vee s'$.

p, q, r, s		$p \vee q \vee s$	$p' \vee r \vee s'$	p', q', r', s'	
■ ■ ■ ■	■ ■ ■ _	■ ■	■ ■	_ _ _ _	_ _ _ ■
_ ■ ■ ■	_ ■ ■ _	■ ■	■ ■	■ _ _ _	■ _ _ ■
■ _ ■ ■	■ _ ■ _	■ ■	■ ■	_ ■ _ _	_ ■ _ ■
_ _ ■ ■	_ _ ■ _	■ _	■ ■	■ ■ _ _	■ ■ _ ■
■ ■ _ ■	■ ■ _ _	■ ■	_ ■	_ _ ■ _	_ _ ■ ■
_ ■ _ ■	_ ■ _ _	■ ■	■ ■	■ _ ■ _	■ _ ■ ■
■ _ _ ■	■ _ _ _	■ ■	_ ■	_ ■ ■ _	_ ■ ■ ■
_ _ _ ■	_ _ _ _	■ _	■ ■	■ ■ ■ _	■ ■ ■ ■

A caution is in order: Observe that if a five variable table were to be used with the number of rows held at eight, then one more xerox would be made of the *two segment table*—producing a *four* (not a three) *segment* table when laid adjacent to the two segment stage. The general formula for the number of rows and columns required when an $n = i + j$ table is constructed to have j segments is: $n \times 2^j$ columns and 2^i rows.

How does all this scale up? Suppose we wish to solve 10 variable SAT problems using the same size transparencies as used for the three variable SAT solved in previous sections. Suppose we use three segments. This will require $10 \times 2^3 = 80$ columns and $2^7 = 128$ rows. This could be done with care by hand. For 20 variable SATs using 8 segments would require $20 \times 2^8 = 5120$ columns and $2^{12} = 4096$ rows. Although this would require special technology, it should still be possible. Once the three positive and three negative copies of the 20 variable truth table have been prepared, any 3-SAT with 20 (or fewer) variables could be solved quickly because only *one* additional xerox operation for *each* clause of any given 3-SAT instance would be required.

If it should be practical to solve substantial SAT problems in this manner, the fantasy for the future would be: The truth table with the largest number of variables, say N, capable of being processed successfully using the highest level of then available technology would be produced. Then at that time every 3-SAT instance having N, or fewer, variables could be solved by the method described here in time linear in the number of clauses. The computation process could be easily automated based on a stack of six (analogs of the) transparent sheets (three positive and three negative copies of the truth table). Presumably in the future some quite different optical technology would be involved—but at least the transparency based xerox procedures illustrated here provide assurance that a technology is possible.

6 The Immediate Future

Since presenting the talk reported above, we have solved instances of the following algorithmic problems by xerography using the methodology illustrated here for SAT problems: (1) finding all minimal dominating sets in a graph, (2) finding all maximum independent sets in a graph, and (3) finding all permutations contained in a binary relation. These procedures have arisen by taking the previously given aqueous solutions ([4, 7, 9], respectively) "out of the water and into the light." The minimum and maximum conditions in (1) and (2) use xerography techniques we have recently developed for: (4) adding arbitrarily long lists of pairs of n-bit binary numbers simultaneously in parallel, and (5) finding the maximum value and all its locations in arbitrarily long lists of n-bit binary numbers. We are now developing a unified exposition of the xerographic approach to combinatorial problems. The reader is likely to "see" the whole program from these remarks: View Sect. 2 above as showing that the traditional trees of possibilities of depth n, in which the number of leaves grows exponentially in n, can be constructed with only n xeroxing (photo) operations. Notice that during the construction, substantial pruning can be carried out in cases such as (1), (2), and (3) above. With the resulting "table" replacing the traditional tree diagram, further restrictions can be tested simultaneously in parallel as in Sect. 3 above. When needed (as in (1) and (2), but not (3) above), parallel *addition* (4) can be used to count in parallel the numbers of "ones" in each row. Then *maximum* (and using a negative xerox, *minimum*) values can be determined with their locations using (5). The interested reader is invited to carry out this program for him/her self immediately—and to go on to apply it to further combinatorial problems. Correspondence is invited at: tom@math.binghamton.edu.

Older readers will recall that several decades ago computing was based on a triumvirate: (1) a lowly key punch machine, (2) decks of data cards, and (3) a monolithic computer. The first two members have since disappeared. The insight here is that with a sufficiently flexible "key punch," the data cards could have done the computing, and the computer could have been returned directly to the manufacturer.

Acknowledgements The concept of computing with xerography, also called *photocomputing* in [5], was "distilled" from the previous concept of *aqueous computing* that was clarified as a concept during work with Susannah Gal and Masayuki Yamamura in Gal's laboratory in Binghamton [8]. Work done with Grzegorz Rozenberg and Herman Spaink in Spaink's laboratory in Leiden has also suggested the basis for this approach to DNA computing [4]. The progress of aqueous computing has been reviewed at length in [6], but work continues with a recent notable contribution [10]. I thank Susannah again for introducing me to wet-lab work. And now I thank her for encouraging me to talk on computing by xerography at the Leiden conference recognizing Grzegorz's birthday—and for criticism of an earlier presentation that resulted in a major improvement.

References

1. Cook SA (1971) The complexity of theorem-proving procedures. In: Proceedings of the 3rd annual ACM symposium on theory of computing. Association for Computing Machinery, New York, pp 151–158
2. Garey MR, Johnson DS (1979) Computers and intractability—a guide to the theory of NP-completeness. Freeman, San Francisco
3. Head T (2001) Splicing systems, aqueous computing, and beyond. In: Antoniou I, Calude CS, Dineen MJ (eds) Unconventional models of computation UMC'2K. Springer, Berlin, pp 68–84
4. Head T (2002) An aqueous algorithm for finding the bijections in a binary relation. In: Brauer W, Ehrig H, Karhumaki J, Salomaa A (eds) Formal and natural computing: essays dedicated to Grzegorz Rozenberg. Lecture notes in computer science, vol 2300. Springer, Berlin, pp 354–360

5. Head T (2007) Photocomputing: explorations with transparency & opacity. Parallel Process Lett 17:339–347
6. Head T, Gal S (2006) Aqueous computing: writing on molecules dissolved in water. In: Chen J, Jonoska N, Rozenberg G (eds) Nanotechnology: science and computation. Springer, Berlin, pp 321–331
7. Head T, Rozenberg G, Bladergroen R, Breek CKD, Lomerese PHM, Spaink H (2000) Computing with DNA by operating on plasmids. Bio Syst 57:87–93
8. Head T, Chen X, Yamamura M, Gal S (2002) Aqueous computing: a survey with an invitation to participate. J Comput Sci Technol 17:672–681
9. Henkel C, Bladergroen R, Balog C, Deelder A, Head T, Rozenberg G, Spaink H (2005) Protein output for DNA computing. Nat Comput 4:1–10
10. Pagano AM, Gal S (2007) An approach for using modified nucleotides in aqueous DNA computing. In: Proceedings of the 13th international computing workshop, Memphis, TN

Some Undecidable Dynamical Properties for One-Dimensional Reversible Cellular Automata

Jarkko Kari and Ville Lukkarila

Abstract Using the fact that the tiling problem of Wang tiles is undecidable even if the given tile set is deterministic by two opposite corners, it is shown that the question whether there exists a trajectory which belongs to the given open and closed set is undecidable for one-dimensional reversible cellular automata. This result holds even if the cellular automaton is mixing. Furthermore, it is shown that left expansivity of a reversible cellular automaton is an undecidable property. Also, the tile set construction gives yet another proof for the universality of one-dimensional reversible cellular automata.

1 Introduction

1.1 History and Brief Outline of the Results

The history of cellular automata dates back to John von Neumann and Stanislav Ulam. This makes cellular automata one of the earliest biologically motivated models of computation. In cellular automata, a regular network of elementary devices operate synchronously, interacting locally with each other. The collective behavior of these cells can be amazingly complex as seen, for example, in the Game-of-Life. This emergence of complexity is an essential feature that makes cellular automata useful models and simulation tools in the study of emergent behaviors.

In physics, cellular automata are good models for microscopic systems consisting of massive numbers of simple particles interacting locally with each other. Time reversibility is a property of microscopic physics that can be programmed into cellular automata by choosing the local interaction rule appropriately. Such reversible cellular automata are interesting objects of study, and also the topic of this article. We

J. Kari's research supported by the Academy of Finland grant 211967.
V. Lukkarila's research supported by the Finnish Cultural Foundation and the Fund of Vilho, Yrjö and Kalle Väisälä.

J. Kari (✉)
Department of Mathematics, University of Turku, 20014 Turku, Finland
e-mail: jkari@utu.fi

J. Kari
Turku Centre for Computer Science, 20520 Turku, Finland

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_32, © Springer-Verlag Berlin Heidelberg 2009

are interested in determining which reversible cellular automata possess some well-known dynamical properties associated to chaotic behavior. Properties investigated include variants of transitivity, sensitivity, and expansivity, and we concentrate on one-dimensional cellular automata only. All our results are for undecidability, and they are based on reductions from a reversible version of the well-known domino problem proved undecidable by Berger in 1966 [3].

The domino problem, also known as the tiling problem of Wang tiles, deals with plane tilings using unit square tiles with colored edges. The problem is to determine, given a finite collection of such tiles, whether the entire plane can be covered by copies of the tiles in such a way that everywhere the abutting edges have identical colors. The tiling problem has become the basis of reduction for many undecidability proofs concerning cellular automata. So far, however, reversible one-dimensional cellular automata have resisted this approach. In this work, we introduce 2-way deterministic tile sets to resolve the problem: we call a set of tiles 2-way deterministic if each tile is uniquely determined by the colors on its north and east sides, as well as by the colors on the opposite south and west sides. A valid tiling of the plane by such tiles can be viewed as a space-time evolution of a reversible cellular automaton. From the fact that the domino problem remains undecidable even when the input instance is 2-way deterministic, we conclude new undecidability results for reversible cellular automata.

Along the way, we also obtain a simple tiling description of a result of Dubacq stating that any Turing machine (reversible or not) can be simulated in real time by a one-dimensional reversible cellular automaton [5]. Some undecidability problems related to the dynamical properties of cellular automata are presented in Sect. 4 for reversible one-dimensional cellular automata. Finally, it is shown that it is undecidable whether a given one-dimensional cellular automaton is left expansive. Previously, no undecidability results regarding dynamical properties of one-dimensional reversible cellular automata have been known.

1.2 Cellular Automata

Cellular automata are dynamical systems which update the variables on an infinite d-dimensional lattice according to some function with a finite number of arguments. Formally, a *cellular automaton* is a 4-tuple $\mathcal{A} = (d, A, N, f)$, where d is the *dimension*, A is the *state set*, $N = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is the *neighborhood vector* consisting of vectors in $\mathbb{Z}^d$ and $f : A^n \to A$ is the *local rule*. A *configuration* $c \in A^{\mathbb{Z}^d}$ is a mapping which assigns a unique state for each cell location in $\mathbb{Z}^d$. The cells in locations $\mathbf{x} + \mathbf{x}_i$ are called *neighbors* of the cell in location $\mathbf{x}$.

At every time step, the new configuration c' is determined by

$$c'(\mathbf{x}) = f\big(c(\mathbf{x}+\mathbf{x}_1), \ldots, c(\mathbf{x}+\mathbf{x}_n)\big), \tag{1}$$

that is, the new state of cell in location $\mathbf{x}$ is computed by applying the local rule to its neighbors. The *global rule* $F : A^{\mathbb{Z}^d} \to A^{\mathbb{Z}^d}$ is defined by setting $F(c) = c'$ in the sense of (1). State q is said to be *quiescent* if $f(q, \ldots, q) = q$.

A cellular automaton is said to be *reversible* if the global rule F has an inverse mapping F^{-1}. It can be shown that if the inverse mapping F^{-1} exists, it is a global rule of a cellular automaton, that is, it is defined by a local rule. It is also known that F is reversible if and only if it is injective. Furthermore, for case of cellular automata, the injectivity of the global rule implies surjectivity of the global rule [12]. It is in general undecidable to know if a given cellular automaton is reversible [11], but in the one-dimensional case reversibility is decidable [1]. A cellular automaton is said to be *nilpotent* if it has a quiescent state q and there exists such a bound n_0 that

$$F^n(c)(i) = q$$

for every $n \geq n_0$, $i \in \mathbb{Z}$ and $c \in A^{\mathbb{Z}^d}$. It is undecidable to determine if a given cellular automaton is nilpotent [10].

The distance between two different configurations c and e can be defined to be

$$d(c,e) = \left(\frac{1}{2}\right)^{\min\{\|\mathbf{x}\|_\infty | c(\mathbf{x}) \neq e(\mathbf{x})\}},$$

where $\|\cdot\|_\infty$ is the max-norm. Function $d(\cdot,\cdot)$ is also a metric thus making the set of configurations a metric space. The balls in the metric are called *cylinders* and they form a basis for the topology. Radius r cylinder containing configuration c is the set

$$\mathrm{Cyl}(c,r) = \left\{e \in A^{\mathbb{Z}^d} | c(\mathbf{x}) = e(\mathbf{x}) \text{ when } \|\mathbf{x}\|_\infty \leq r\right\}$$

For every radius r, there are only finitely many cylinders and these cylinders are by definition disjoint. Therefore, radius r cylinders form a finite partition of the space of configurations. Hence, every cylinder is clopen because the complement of every cylinder is a union of other cylinders with the same radius.

Pair (X,F) is a *dynamical system* if X is compact topological space and $F: X \to X$ is a continuous mapping. In particular, d-dimensional cellular automata are dynamical systems of the form $(A^{\mathbb{Z}^d}, F)$. The properties of cellular automata as dynamical systems have been extensively studied. Further background information on cellular automata can be found in [12]. In this work, we concentrate on one-dimensional reversible cellular automata, so from now on $d = 1$.

Transitivity, (topological) mixing and different variants of expansivity are among dynamical properties related to chaotic behavior. Cellular automaton $(A^{\mathbb{Z}}, F)$ is *transitive* if for all non-empty open sets U and V there exists such a positive integer n that $F^n(U) \cap V \neq \emptyset$. Cellular automaton $(A^{\mathbb{Z}}, F)$ is *mixing* if for all non-empty open sets U and V there exists such a positive integer n that $F^k(U) \cap V \neq \emptyset$ for every $k \geq n$. By definition, a cellular automaton which is mixing is transitive also. A reversible cellular automaton $(A^{\mathbb{Z}}, F)$ is *expansive* if there exists such a constant ϵ that for any two different configurations c and e

$$d\big(F^n(c), F^n(e)\big) \geq \epsilon \tag{2}$$

for some integer $n \in \mathbb{Z}$.

A cellular automaton $(A^{\mathbb{Z}}, F)$ is *positively expansive* if there exists such a constant ϵ that for any two different configurations c and e (2) holds for some positive integer $n \in \mathbb{N}$.

It is known that only one-dimensional cellular automata can be expansive or positively expansive [7, 21]. It can be easily shown that a reversible cellular automaton cannot be positively expansive. Furthermore, positive expansivity implies topological mixing (and transitivity) [4]. A simple example of a cellular automaton, which is reversible, expansive and transitive, is the *shift* $\sigma : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$. The shift mapping is defined according to

$$\sigma(c)(i) = c(i+1).$$

A reversible cellular automaton $(A^{\mathbb{Z}}, F)$ is *left expansive* if there exists such a constant ϵ that for any two different configurations c and e with $c(i) \neq e(i)$ for some $i < 0$ (2) holds for some integer $n \in \mathbb{Z}$. Similarly, a reversible cellular automaton is *right expansive* if there exists such a constant ϵ that for any two different configurations c and e with $c(i) \neq e(i)$ for some $i > 0$ (2) holds for some integer $n \in \mathbb{Z}$.

A cellular automaton $(A^{\mathbb{Z}}, F)$ is *positively left expansive* if there exists such a constant ϵ that for any two different configurations c and e with $c(i) \neq e(i)$ for some $i < 0$ (2) holds for some positive integer $n \in \mathbb{N}$. Similarly, the cellular automaton is *positively right expansive* if there exists such a constant ϵ that for any two different configurations c and e with $c(i) \neq e(i)$ for some $i > 0$ (2) holds for some positive integer $n \in \mathbb{N}$.

The original definition of left and right expansivity given by Kurka actually meant positive left and right expansivity [14]. Here, Kurka's left expansivity is called positive left expansivity and right expansivity is called positive right expansivity because of the similar difference between expansivity of reversible cellular automata and positive expansivity of the irreversible cellular automata. Intuitively positive left expansivity means that if two configurations differ by some cell then their later images will differ by cells further to the right. Similarly positive right expansivity means that if two configurations differ by some cell then their later images will differ by cells further to the left.

1.3 Tilings

A *Wang tile* (or a *tile* in short) is a unit square with colored edges. The edges of a Wang tile are called *north*, *east*, *west*, and *south* edges in a natural way. Each edge of a Wang tile has a *color* which is a label from a finite alphabet. For the given tile t, expressions t_N, t_E, t_W and t_S are used to denote north, east, west, and south edge colors, respectively. A *Wang tile set* T (or a *tile set* in short) is a finite set containing Wang tiles.

Given tile sets $T_1, \ldots, T_n$, a tile set $T \subseteq T_1 \times \cdots \times T_n$ is a *sandwich tile set*. Elements of T are called *sandwich tiles*. Tile set T_i is said to be *layer* i of the sandwich tile set $T \subseteq T_1 \times \cdots \times T_n$. Let $t \in T$ be an element of a sandwich tile set and $t = (t_{i_1}, \dots, t_{i_n})$. Then the colors of t are sequences of corresponding colors of the original tiles, for example, $t_N = (t_{i_1 N}, \dots, t_{i_n N})$.

A *tiling* is a mapping $f : \mathbb{Z}^2 \to T$, which assigns a unique Wang tile for each integer pair of the plane. A tiling f is said to be *valid*, if for every pair $(x, y) \in \mathbb{Z}^2$ the tile $f(x, y) \in T$ matches its neighboring tiles (e.g. the south side of tile $f(x, y)$ has the same color as the north side of tile $f(x, y-1)$).

A tiling $f : \mathbb{Z}^2 \to T$ is called *periodic* with period (a, b) if $f(x, y) = f(x+a, y+b)$ for all $(x, y) \in \mathbb{Z}^2$ and $(a, b) \neq (0, 0)$. Otherwise, the tiling f is called *nonperiodic*. A tile set T is called *aperiodic*, if there exists some tiling with the tile set T, but no tiling with the tile set T is periodic. If the tile set T admits a periodic tiling $f : \mathbb{Z}^2 \to T$ with some period, then it admits also a *doubly periodic* tiling $g : \mathbb{Z}^2 \to T$, that is, there exists such non-zero integers a and b that $g(x, y) = g(x+a, y)$ and $g(x, y) = g(x, y+b)$ for all $(x, y) \in \mathbb{Z}^2$ [20].

The following decision problem is referred to as the *tiling problem*: "Given a Wang tile set T, does there exist a valid tiling of the plane?" A tiling $f : \mathbb{Z}^2 \to T$ is said to *contain* tile $t \in T$, if for some integers $x, y \in \mathbb{Z}$ equation $f(x, y) = t$ holds. The following decision problem is referred to as the *tiling problem with a seed tile*: "Given a Wang tile set T and a tile $t \in T$, does there exist a valid tiling of the plane that contains the tile t?"

If the tiling problem with a seed tile was decidable, then the tiling problem would be decidable. Let T be the tile set of the given instance of the tiling problem. Then the answer for the tiling problem is affirmative, if, and only if, for some tile $t \in T$ the answer for the tiling problem with a seed tile is affirmative considering the tile set T as the tile set of the instance and the tile t as the seed tile of the instance. It is already known that the tiling problem is undecidable [3, 10, 16, 20].

A Wang tile set T is said to be *NE-deterministic*, if within the tile set there does not exist two different tiles with the same colors on the north and east sides. In general, a Wang tile set is *XY-deterministic*, if the colors of X- and Y-sides uniquely determine a tile in the given Wang tile set. A Wang tile set is 4-*way deterministic*, if it is NE-, NW-, SE-, and SW-deterministic. A tile set will be called 2-*way deterministic*, if it is NE- and SW-deterministic.

The definition of determinism for tile sets was originally motivated by the theory of cellular automata [10, 12]. A tile set can be considered as a one-dimensional cellular automaton, if the tile set contains all the possible color pairs at one corner and the tile set is deterministic by the same corner. The tiles can be seen as states of the cells and the diagonal rows of tiles as configurations of the cellular automaton. Therefore, the rule which determines whether neighboring tiles match can in this case be considered as the local rule of a cellular automaton. It has been shown that the tiling problem is undecidable with tile sets that are deterministic by at least one corner. From this, it follows that nilpotency of one-dimensional cellular automata is undecidable [10]. If the Wang tile set is 2-way deterministic by opposite corners (and all color pairs are met at these corners), then the tile set can be seen as a reversible cellular automaton.

1.4 Turing Machines

In what follows, the reader is assumed to be somewhat familiar with the concept of Turing machine [9, 17]. In this article, a *Turing machine* $\mathcal{M}$ is considered to be a four-tuple $\mathcal{M} = (Q, \Gamma, \delta, q_0)$, where Q is the state set, Γ is the tape alphabet, δ is the partial transition function and $q_0 \in Q$ is the initial state. No "accept", "reject", or "halt" states are defined explicitly. The tape of a Turing machine is defined to be two-way infinite and symbol ε is used to denote the empty symbol of the Turing machine. The transition function is a partial mapping

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{\triangleleft, \triangleright\},$$

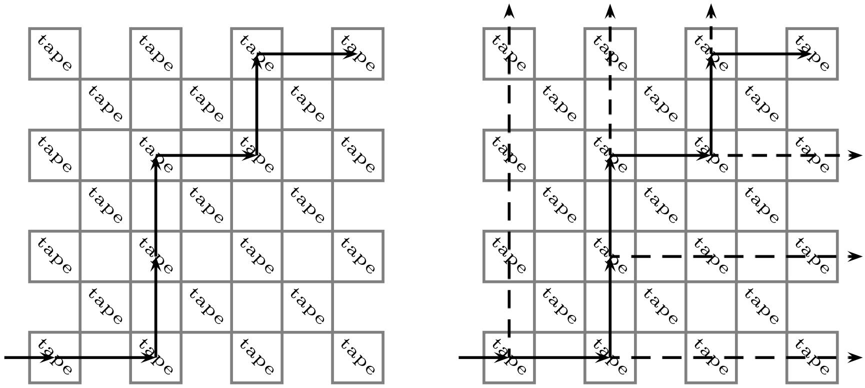
that is, at every time step the read-write head moves either to the left or to the right or halts. A Turing machine is said to *halt*, if it is in state q reading symbol s and $\delta(q, s)$ in undefined. The *Turing machine halting problem* is the following decision problem: "Does the given Turing machine $\mathcal{M}$ halt when started on an empty tape?" The halting problem is known to be undecidable.

2 The 2-Way Deterministic Tiling Problem with a Seed Tile

It is known that the tiling problem is undecidable in the 4-way deterministic case [16] with and without a seed tile. However, in the 4-way deterministic case, the construction is unnecessarily complicated in terms of cellular automata. Therefore, it is shown in this section that the tiling problem with a seed tile is undecidable for 2-way deterministic tile sets. The construction is much more simple and it provides the same undecidability results with respect to cellular automata while the argumentation is more readable. Furthermore, the 2-way deterministic construction gives another proof for the fact that any Turing machine can be simulated with a reversible one-dimensional cellular automaton, which could not be achieved with the 4-way deterministic construction in [16].

2.1 The Idea for the Undecidability Proof

The basic idea is to represent the Turing machine tape on diagonal rows as in [10]. It is easy to show that an arbitrary Turing machine computation can be represented on diagonal rows. The computation on diagonal rows is done in the manner of Fig. 1(a). Every second diagonal row in the northwest-southeast direction is used to represent the Turing machine configuration at a certain moment. One tile at each diagonal row represents the read-write head and the current symbol to be read. The other tiles of the diagonal row represent the other symbols on the tape located to the left and to the right from the read-write head.



(a) The rough idea of representing Turing machine computation on diagonal rows.

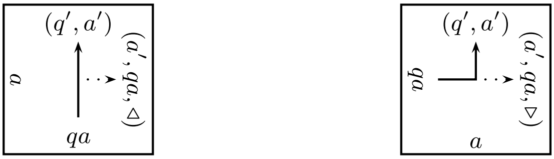
(b) The Turing machine computation with additional information signals of earlier read-write operations.

Fig. 1 The general idea of representing the computation on diagonal rows

Since a Turing machine is a deterministic method of computation, the tile set constructed in this manner is clearly deterministic in (at least) one direction. More specifically, it is the direction to which the computation advances in time. To force determinism also in the opposite direction, some modifications are needed. On every operation of the read-write head, a "signal" is sent to the direction that is opposite to the read-write head movement. This signal contains information about the read-write operation which is currently being conducted and the direction from which the read-write head entered the current cell after the previous move. The computation with signals is represented in Fig. 1(b). In Fig. 1(b), if the read-write head moves to the left, then the signal is sent toward the east, and if the read-write head moves to the right, then the signal is sent toward the north. In practice, the signal is just a component of a side color which is moves onward unobstructed. These signals containing information about the previous move and the current one are referred to as the *move signals*. The move signals are started on the tiles in Figs. 2 and 3 (i.e. the tiles that represent the read-write head). The tiles in Fig. 5 (i.e. the tiles that represent the tape) just move the possible move signals onward. This construction will make the tile set representing the given Turing machine 2-way deterministic.

2.2 The Tile Set for the Given Turing Machine

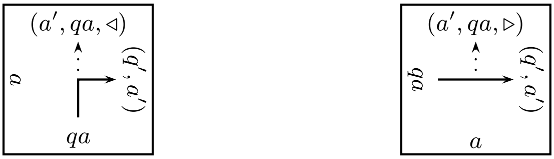
In this subsection, a 2-way deterministic tile set is constructed for the given Turing machine. In what follows, the diagonal rows of tiles are referred to as *diagonals* in short.



(a) The new move is to the left and the previous move was to the left.

(b) The new move is to the left and the previous move was to the right.

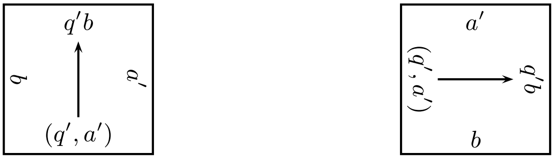
Fig. 2 Action tiles for move $\delta(q,a)=(q',a',\triangleleft)$



(a) The new move is to the right and the previous move was to the left.

(b) The new move is to the right and the previous move was to the right.

Fig. 3 Action tiles for move $\delta(q,a)=(q',a',\triangleright)$



(a) The tile for read-write operation $\delta(q,a)=(q',a',\triangleleft)$.

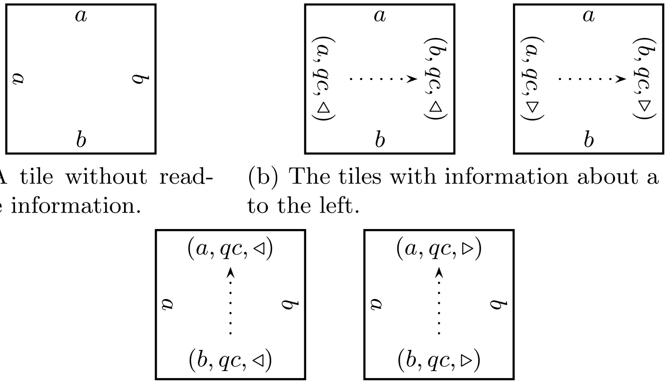
(b) The tile for read-write operation $\delta(q,a)=(q',a',\triangleright)$.

Fig. 4 Merging tiles. The tiles depend on the new state q', the new symbol a' to be written, the move direction and on the new symbol b to be read

The tiles to represent read-write operations For every possible move of the Turing machine, either the tiles in Fig. 2 and the tile in Fig. 4(a), or the tiles in Fig. 3 and the tile in Fig. 4(b) are added to the tile set.

The tiles for a left move Assume that the Turing machine contains move $\delta(q,a)=(q',a',\triangleleft)$. Then the tiles in Fig. 2 and the tile in Fig. 4(a) are added to the tile set. The tile in Fig. 2(a) is used if the previous move was to the left and the current move is to the left. If the previous move was to the right, then the tile in Fig. 2(b) is used.

The tiles for a right move Assume that the Turing machine contains move $\delta(q,a)=(q',a',\triangleright)$. Then the tiles in Fig. 3 and the tile in Fig. 4(b) are added to the tile set.



(a) A tile without read-write information.

(b) The tiles with information about a move to the left.

(c) The tiles with information about a move to the right.

Fig. 5 The tiles to represents the symbols on the tape. Here, q denotes an arbitrary state and symbols a, b, and c denote arbitrary elements of the tape alphabet

The tile in Fig. 3(a) is used if the previous move was to the left and the current move is to the right. If the previous move was to the right and the current move is to the right, then the tile in Fig. 3(b) is used.

The tiles in Figs. 2 and 3 will be called *action tiles* and the tiles in Fig. 4 will be called *merging tiles*. Together these tile are referred to as *move tiles* or the tile set $M_{\mathcal{M}}$.

The tiles to represent tape contents For every state q and every element a, b, and c of the tape alphabet, the tiles in Fig. 5 are added to the tile set. The tile in Fig. 5(a) is used to represent a cell (or the border between two cells if $a \neq b$) of the tape without any information about an earlier read-write operation.

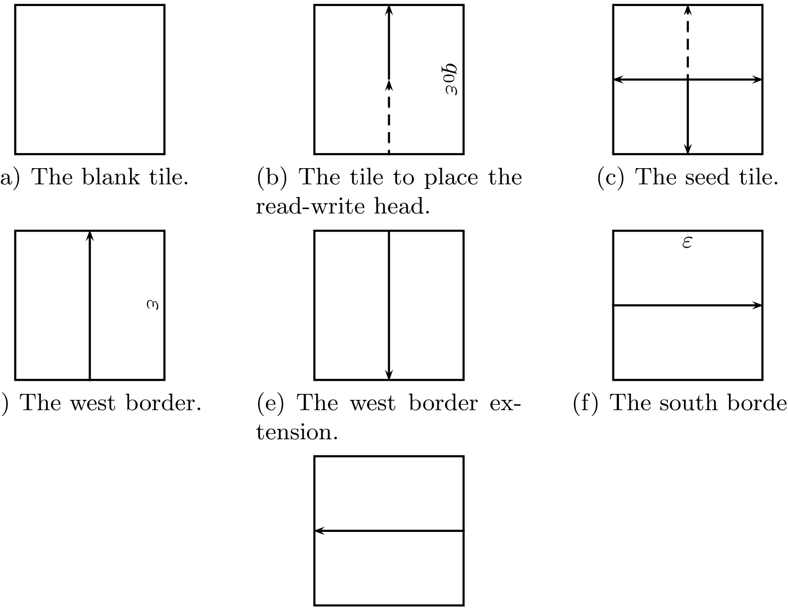
The tiles in Fig. 5(b) represent tape contents likewise, but contain also information about a read-write operation during which the read-write head moved to the left. That is, the east side and the west side have colors of form $(\cdot, qc, \cdot)$ if, and only if there exist a move of form $\delta(q, c) = (\cdot, \cdot, \triangleleft)$.

The tiles in Fig. 5(c) are similar to the tiles in Fig. 5(b) with the exception that they contain information about a read-write operation during which the read-write head moved to the right and not to the left. The north side and the south side have colors of form $(\cdot, qc, \cdot)$ if and only if there exist a move of form $\delta(q, c) = (\cdot, \cdot, \triangleright)$.

The tile set is being constructed so, that if the seed tile (i.e. the tile in Fig. 6(c)) is located on an even diagonal, then on every odd diagonal symbols a and b in Fig. 5 are equal.

The tiles that are used to represent the tape contents of the given Turing machine $\mathcal{M}$ are referred to as *alphabet tiles* or as the tile set $A_{\mathcal{M}}$.

The starting tiles To force the Turing machine to start on a blank tape only, the tiles in Fig. 6 are added to the tile set. One of these tiles (namely, the tile in Fig. 6(c)) is chosen to be the seed tile. If the seed tile is contained within a tiling, then the tiling represents a Turing machine computation. Other tiles in Fig. 6 force the Turing



(a) The blank tile.

(b) The tile to place the read-write head.

(c) The seed tile.

(d) The west border.

(e) The west border extension.

(f) The south border.

(g) The south border extension.

Fig. 6 Starting tiles

machine to start on a blank tape. The blank initial configuration of the Turing machine is represented by the tile pattern shown in Fig. 8. In short, if the seed tile is located in the origin, then the Turing machine simulation is done in the first quadrant.

For the given Turing machine $\mathcal{M}$, the tiles in Fig. 6 are referred to as *starting tiles* or as the tile set $S_{\mathcal{M}}$.

For every Turing machine $\mathcal{M}$, the tile set constructed using the method above is denoted by $T_{\mathcal{M}}$, that is,

$$T_{\mathcal{M}} = M_{\mathcal{M}} \cup A_{\mathcal{M}} \cup S_{\mathcal{M}}.$$

An example of a Turing machine operation is shown in Fig. 7.

Let (q,a) be any preimage pair for which the transition $\delta(q,a)$ is not defined. Then there will be no tile that would have the color qa on its west side or south side. Therefore, if the Turing machine halts, that is, if at some moment of time the read-write head in state q reads symbol a, then the tiling cannot be completed to cover the entire plane in a valid way.

Lemma 1 *For any given Turing machine $\mathcal{M}$, the tile set $T_{\mathcal{M}}$ is both NE- and SW-deterministic.*

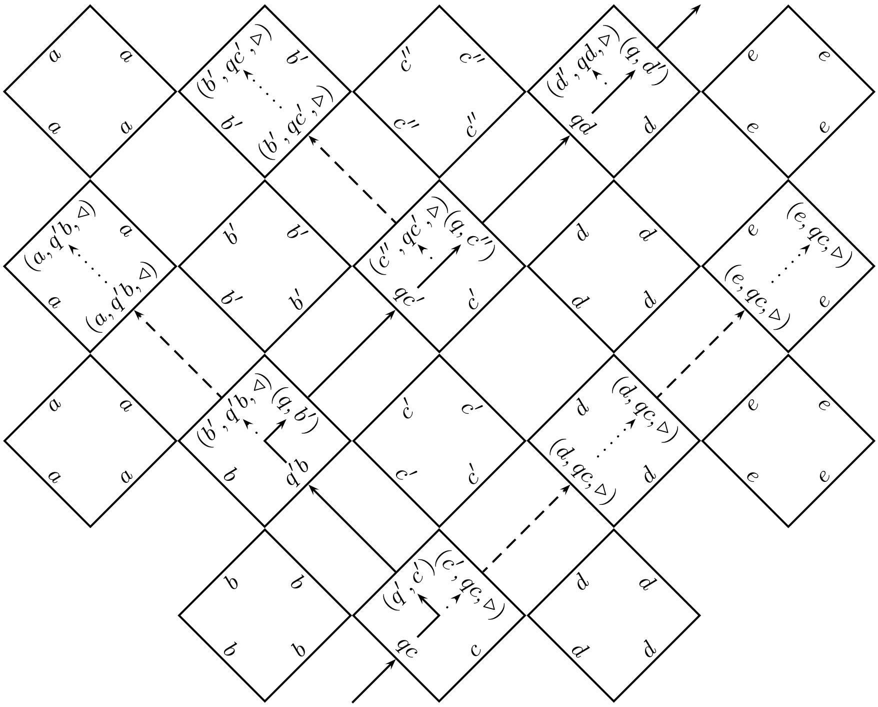
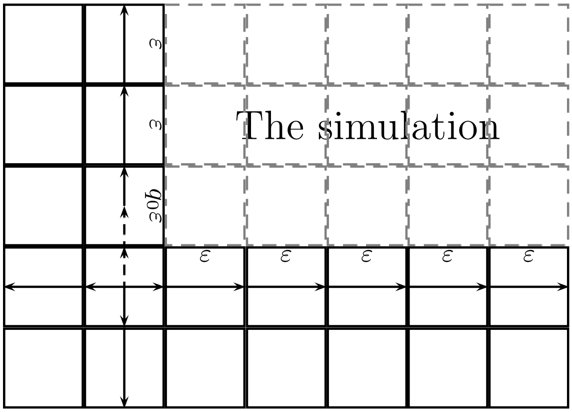


Fig. 7 Rewrite operation $abqcd \vdash aq'bc'de \vdash ab'qc'de \vdash ab'c''qde$

Fig. 8 Using the tiles in Fig. 6 to start the Turing machine simulation on a blank tape



Proof The tile set is SW-deterministic, since clearly it has no two tiles having the same colors on the south side and the west side.

Similarly, the tile set is NE-deterministic. No two tiles in Figs. 2, 4, 3, and 5 have the same colors on the north side and the east side. □

Theorem 1 *The following question is undecidable: "Given a Turing machine $\mathcal{M}$, does the tile set $T_{\mathcal{M}}$ admit a valid tiling of the plane containing the tile in Fig.* 6(*c*)*?"*

Proof The tile set $T_{\mathcal{M}}$ quite obviously corresponds the actions and configurations of the given Turing machine $\mathcal{M}$. Requiring the seed tile to be the tile in Fig. 6(c), the structure in Fig. 8 is forced to be tiled on the plane.

The structure in Fig. 8 obviously corresponds the initial configuration with a blank tape. Therefore, the plane can be tiled correctly if and only if the given Turing machine does not halt (when started on a blank tape). Of course, the halting problem with a blank tape is undecidable. □

Since the tiling problem with a seed tile is a generalization of the problem in Theorem 1, it is seen that the tiling problem with a seed tile is undecidable for tile sets that are 2-way deterministic. Moreover, the tile set $T_{\mathcal{M}}$ would be NE-deterministic even if the Turing machine $\mathcal{M}$ was non-deterministic. No matter what the state q and symbol a are, the tiles in Figs. 2, 4, and 3 are uniquely defined by the colors of their north and east sides.

A 2-way deterministic tile set can be constructed even for any non-deterministic Turing machine. This tile set is constructed by modifying the tile set $T_{\mathcal{M}}$. Modification is based on using signals containing information about the particular move that was chosen. These signals could be referred to as *decision signals*. The tile in Fig. 2 is modified so that it sends a decision signal to the left and backward in time (i.e. toward west since the computation advances toward the northeast). Likewise, the tile in Fig. 3 is modified to send a decision signal to the right and backward in time (i.e. toward the south). Furthermore, the tiles in Figs. 5 and 6 are modified to allow crossings with any kinds of decision signals. It is quite straightforward to see that the new modified tile set is indeed both NE- and SW-deterministic.

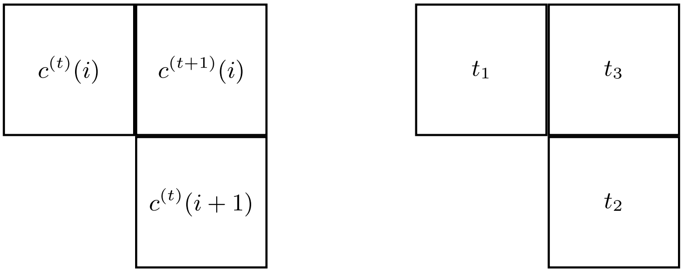
3 Tile Sets and Cellular Automata

3.1 Interpreting Tile Set as a Cellular Automaton

Following the presentation in [10], it is possible to regard Wang tile sets (that are deterministic at least in one direction) as one-dimensional cellular automata.

If the given tile set is, say, SW-deterministic, it is possible to consider the tiles as states of a cellular automaton. As shown in Fig. 9, with a cellular automaton the next state of a cell is determined with a similar manner as the next tile (to the northeast) in a tiling with a SW-deterministic tile set. With a cellular automaton, the new state depends on the old states and in a tiling (with a SW-deterministic tile set) the new tile is determined by the colors of its neighbors.

It should be noted that the given Wang tile set may not contain all the possible color pairs in the southwest corners of the tiles. If the given tile set T is assumed to be deterministic in only one direction, say, by the southwest corner, it is enough to



(a) Cells $c(i)$ and $c(i+1)$ determine the next state of the cell $c(i)$.

(b) Tiles t_1 and t_2 determine tile t_3 SW-deterministically.

Fig. 9 The tiles of a SW-deterministic tile set can be considered as states of a cellular automaton

add a tile to the original tile set for every missing southwest corner color pair. For example, if there is no tile t with $t_W = x$ and $t_S = y$ in the given tile set T, a tile t with $t_N = t_E = z$, $t_W = x$ and $t_S = y$, where z is any color of the tile set T, could be added to the tile set while maintaining SW-determinism.

If the given tile set is assumed to be both NE- and SW-deterministic, equally many color pairs are missing as northeast corners and southwest corners. It is trivial to construct (for example, by some ordering method) a one-to-one correspondence between the missing colors in the southwest corners and the missing colors of the northeast corners. This bijection can clearly be considered as a NE- and SW-deterministic set of tiles. Moreover, the union of the initial tile set and this new tile set is both NE- and SW-deterministic tile set containing N^2 tiles, where N is the number of colors in the original tile set. Let $T^{\complement}$ denote the new tile set which was constructed for the missing color pairs. Now it can be seen that the tile set $T \cup T^{\complement}$ can be considered as a reversible cellular automaton.

A bit more formally, for the NE- and SW-deterministic tile set T, a reversible cellular automaton $(T \cup T^{\complement}, F_T)$ simulates the tiling procedure. The global rule F_T is defined using local rule

$$f_T(x, y) = z \quad \text{if } x, y \in T \cup T^{\complement},\ x_E = z_W \text{ and } y_N = z_S.$$

The function $f_T : (T \cup T^{\complement})^2 \to T \cup T^{\complement}$ is total and well defined, since the tile set $T \cup T^{\complement}$ is both NE- and SW-deterministic.

3.2 Universality of One-Dimensional Reversible Cellular Automata

It has been shown by Morita and Harao that one-dimensional reversible cellular automata are computationally universal [18]. More precisely, they have shown that any reversible Turing machine can be simulated with some reversible one-dimensional cellular automaton. Since any Turing machine can be simulated with a reversible

Turing machine [2], the universality of one-dimensional reversible cellular automata follows.

However, the requirement of reversibility for the given Turing machine is not necessary for the machine to be simulated with a reversible one-dimensional cellular automaton. In fact, Dubacq has given a construction for a family of reversible cellular automata to simulate any irreversible Turing machine in real time [5]. Dubacq's approach was more from the cellular automata point of view. The construction of the family of tile sets given in Sect. 2.2 gives a tiling view for Dubacq's result. Namely, an instantaneous description of a Turing machine can be represented by a configuration where exactly one of the cells contains a move tile representing the read-write head and all the rest of the cells have alphabet tiles as states. The cellular automaton executes one Turing machine computation step in two time steps. If the initial configuration c of the cellular automaton (with global rule F) represented valid Turing machine configuration, so does $F^{2t}(c)$. The additional move signals can be ignored and the instantaneous description of the Turing machine computation can be read directly.

Theorem 2 (J.-C. Dubacq, 1995 [5]) *Any Turing machine can be simulated using a reversible one-dimensional cellular automaton in real time.*

Proof The given Turing machine $\mathcal{M}$ can be simulated with the cellular automaton $(T \cup T^{\mathsf{C}}, F_T)$, where $T = M_{\mathcal{M}} \cup A_{\mathcal{M}}$, that is, the state set of the cellular automaton consists of the move tiles and the alphabet tiles and the tiles of (some) set $(M_{\mathcal{M}} \cup A_{\mathcal{M}})^{\mathsf{C}}$. Because for every computation step of the Turing machine $\mathcal{M}$ the cellular automaton $(T \cup T^{\mathsf{C}}, F_T)$ conducts two computation steps, the cellular automaton with global rule F_T^2 simulates the given Turing machine in real time. □

Corollary 1 (K. Morita and M. Harao, 1989 [18]) *Reversible one-dimensional cellular automata are computationally universal.*

4 Undecidability Results for Reversible Cellular Automata

4.1 Undecidability of the 2-Way Deterministic Tiling Problem

It can be shown that the 4-way deterministic tiling problem with a seed tile can be reduced to the 4-way deterministic tiling problem [16]. With the same construction, one can reduce the 2-way deterministic tiling problem with a seed tile to the 2-way deterministic tiling problem without losing the 2-way determinism of the original tile set instance. With this reduction, Theorem 3 follows from Theorem 1. Of course, the result is also a direct consequence of the 4-way deterministic variant of [16].

Theorem 3 ([16]) *The tiling problem is undecidable for tile sets that are* 2*-way deterministic.*

This theorem is the basis of the proofs of the undecidability results in the following sections.

4.2 Some Undecidability Results for Cellular Automata

A cellular automaton $(A^{\mathbb{Z}}, F)$ is said to be *globally immortal* with respect to subset $B \subseteq A$ if there exists such a configuration $c \in B^{\mathbb{Z}}$ that $F^n(c)(i) \in B$, for all integers $n, i \in \mathbb{Z}$. The following decision problem is referred to as the *global immortality problem*: "Given a cellular automaton $(A^{\mathbb{Z}}, F)$ and subset $B \subseteq A$, is $(A^{\mathbb{Z}}, F)$ globally immortal with respect to B?"

A cellular automaton is said to be *locally immortal* with respect to subset $B \subseteq A$ if there exists such a configuration $c \in B^{\mathbb{Z}}$ that $F^n(c)(0) \in B$, for all integers $n \in \mathbb{Z}$. The following decision problem is referred to as the *local immortality problem*: "Given a cellular automaton $(A^{\mathbb{Z}}, F)$ and subset $B \subseteq A$, is $(A^{\mathbb{Z}}, F)$ locally immortal with respect to B?"

These definitions of "immortality" are motivated by the immortality problem of Turing machines [8]. Hooper has shown that it is undecidable if a given Turing machine halts on every extended configuration (i.e. the tape may contain infinitely many non-blank cells).

The problems of global immortality and local immortality are slightly related to the nilpotency. In a nilpotent cellular automaton every cell enters to a quiescent state q within a bounded number of time steps. In the immortality problems, the elements of $A \setminus B$ correspond to the quiescent state of a nilpotent cellular automaton with the distinction that a cell does not have remain in a state of $A \setminus B$. With this difference, the immortality problems make sense for reversible cellular automata also.

Theorem 4 *The global immortality problem is undecidable for reversible one-dimensional cellular automata.*

Proof Undecidability of the question follows by a reduction from the problem of Theorem 3. Let $(T \cup T^{\mathtt{C}}, F_T)$ be the cellular automaton constructed in Sect. 3.1 and let $B = T$ in the definition of the problem.

Assume first, that the given tile set T admits a valid tiling. Then one can choose any northwest-southeast diagonal row of tiles of the valid tiling to be the configuration c. Since the tile set is NE-deterministic and configuration c is part of a valid tiling, $F_T^n(c)(i) \in T$, for all integers $n, i \in \mathbb{Z}$.

Assume second that the given tile set T does not admit a valid tiling. If for some configuration c (considered again as a northwest-southeast diagonal row of a valid tiling), the condition of the problem did hold, then it would be possible to construct a valid tiling. However, this contradicts the assumption.

Hence, a configuration c exists if, and only if, the tile set T admits a valid tiling. □

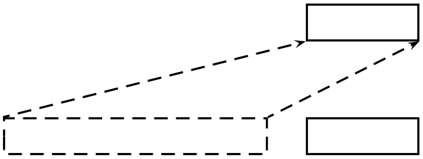


Fig. 10 Combining the global rule with a shift causes the preimages of two different patterns to be transferred to disjoint locations

Any surjective cellular automaton with global rule F and radius r can be converted to a mixing cellular automaton by forming a new global rule which is a composition of the original global rule and a sufficiently high power of the shift rule σ. For example, composition $F \circ \sigma^{r+1} : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ is a mixing cellular automaton. The idea is illustrated better in Fig. 10. By applying a shift, the preimage of the requested pattern is moved away from the origin.

Lemma 2 *Let $(A^{\mathbb{Z}}, F)$ be a surjective cellular automaton with radius r. Then $(A^{\mathbb{Z}}, F \circ \sigma^{r+1})$ is mixing.*

Lemma 3 *Let $(A^{\mathbb{Z}}, F)$ be an injective cellular automaton with radius r_1 and let its inverse local rule have radius r_2. Then $(A^{\mathbb{Z}}, F \circ \sigma^{r+1})$ is expansive where $r = \max(r_1, r_2)$.*

Corollary 2 *The global immortality problem is undecidable for reversible one-dimensional cellular automata that are both expansive and mixing.*

Proof The property in question is shift-invariant. Therefore, by using Lemmas 2 and 3, the claim follows from Theorem 4. □

Theorem 5 *The local immortality problem is undecidable for reversible one-dimensional cellular automata that are left expansive.*

Proof Let T_1 be the tile set in Fig. 11(a) and let T_2 be the tile set in Fig. 11(b). Let t_b be the blank tile in set T_1. Form a new sandwich tile set

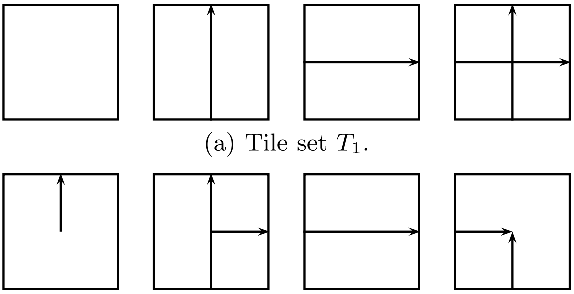
$$A = (T \times T_1) \cup \left(T^{\mathtt{C}} \times T_2\right)$$

and let $B = T \times \{t_b\}$.

This tile set can also be considered as a reversible cellular automaton because all the possible color pairs occur both at northeast and southwest corners. The cellular automaton simulates tiling on two layers with some additional constraints. On the first layer are tiles $T \cup T^{\mathtt{C}}$ and on the second layer are tiles $T_1 \cup T_2$. Let G be the global rule of this cellular automaton.

Let $c \in B^{\mathbb{Z}}$, that is, let c be a configuration consisting of states $T \times \{t_b\}$ only. If c is diagonal row of tiles in a valid tiling, then $G^n(c) \in B^{\mathbb{Z}}$ for every $n \in \mathbb{Z}$. If c does not generate a valid tiling, then $G^n(c)(i) \in T^{\mathtt{C}} \times T_2$ for some $n, i \in \mathbb{Z}$. Due to the structure of tile sets T_1 and T_2, once the “arrow signal” is generated using the tiles of T_2, it is always forwarded at least to the left (i.e. northwest) and either forward

Fig. 11 New tiles to pair with the original tile set



(a) Tile set T_1.

(b) Tile set T_2.

(i.e. north) or backward (i.e. west) in time without ever being canceled. This implies that $G^n(c)(0) \in A \setminus B$ for some $n \in \mathbb{Z}$. Hence, $G^n(c)(0) \in B$ for every $n \in \mathbb{Z}$ if, and only if, T admits a valid tiling. This would conclude the proof for reversible cellular automata.

For left expansive cellular automata, the rule G is modified so that underlying tiling rule of state components $T \cup T^{\mathsf{C}}$ is combined with shift σ. Let this new global rule be F. This converts the tiling procedure by $T \cup T^{\mathsf{C}}$ into an expansive cellular automaton. The second rule on the second layer is not modified because tile set T_1 defines an expansive cellular automaton and tile set T_2 defines a left expansive cellular automaton. Therefore, using either tiles T_1 or T_2 on predefined locations atop tiles $T \cup T^{\mathsf{C}}$ defines a left expansive cellular automaton. □

It is not known whether the local immortality problem is decidable or undecidable for expansive cellular automata. If the problem was an undecidable for expansive cellular automata, then also expansivity would be an undecidable property. This would follow by applying the same construction as later in the proof of Theorem 7.

Theorem 6 *The local immortality problem is undecidable for reversible one-dimensional cellular automata that are both left expansive and mixing.*

Theorem 6 will be proved by modifying the cellular automaton in the proof of Theorem 5 so that it becomes mixing.

The only reason why the cellular automaton is not already mixing is that the signals may cause dependence between two configurations in the same computation. That is, configurations of two different cylinders might not appear in the same orbit because the contents of one cylinder might cause some undesired pattern of signals (of the ones generated by T_1 and T_2) appear throughout the computation near the origin. For the given two open sets U and V, this might prevent from finding a configuration $c \in U$ so that $F^k(c) \in V$ for some k (i.e. the cellular automaton would not be even transitive). The problem is illustrated in Fig. 12.

Unfortunately, the shift rule cannot be applied to the signals of tile sets T_1 and T_2 because then a column of tiles without the signals might appear even if there was tiling error (i.e. a state from $T^{\mathsf{C}} \times T_2$) in the tiling represented by the computation.

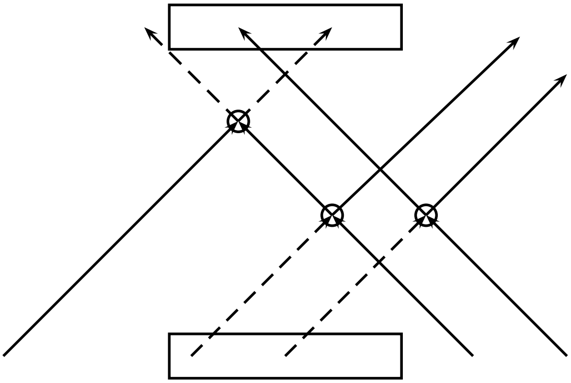


Fig. 12 The signal generated by the tile set T_2 may cause dependence between cells of two configurations of the same orbit in the proximity of origin. *Solid lines* represent the signals in the tiles. *Dashed lines* represent absence of the signals. *Circles* denote tiling errors (where T_2 is used). The *rectangles* represent domains defining the different cylinders

Combining the original rule with shift would make the cellular automaton mixing, but the problem might not be undecidable anymore. Therefore, the signals in tiles T_1 and T_2 need to be modified.

Proof (sketch) Let $(T \cup T^{\mathtt{C}}, G)$ be the cellular automaton defined by the 2-way deterministic tile set T. It can be assumed that $(T \cup T^{\mathtt{C}}, G \circ \sigma^{r+1})$ is expansive and mixing for some positive integer r. The theorem will be proved by a reduction from the problem of Corollary 2.

The state set $T \cup T^{\mathtt{C}}$ and the rule $G \circ \sigma^{r+1}$ is modified by adding a three different kinds of signals. The first set of signals consists of signals that move with a speed of one cell per time step either from left to right or right to left. For the sake of the argument, let these signals be called *type* 1 *signals*. The second set consists of signals that move with a speed of one cell per two time steps either from left to right or right to left. These signals will be called *type* 2 *signals*.

The third set of signals consists of signals that move with a speed of one cell per time step either from left to right or right to left. These signals will be called *type* 3 *signals* and they correspond the signals of tile sets T_1 and T_2. If a tiling error occurs in the underlying tiling, the type 3 signals in the same cell are modified as in tiles T_2 if and only if there is no type 1 and type 2 signal present in the cell. That is, type 1 and type 2 signals are used to "cancel out" the effects of tiling errors on type 3 signals. Signals of type 1 and 2 are not altered at any point. Let the new global rule be denoted by F.

Given two radius r cylinders $C_1 = \mathrm{Cyl}(c_1, r)$ and $C_2 = \mathrm{Cyl}(c_2, r)$, it is shown in Fig. 13 how the type 1 and 2 signals can be used to cancel out tiling errors everywhere else except in the immediate vicinity of the origin. Type 1 and 2 signal from different sides of the origin coincide after $t_0 = 2r + 2$ time steps. Therefore, configurations belonging to any two cylinders can appear in the same trajectory for every $t \geq \max(t_0, n_0)$ time steps apart, where n_0 is the bound given implicitly by Lemma 2. Hence, cellular automaton F is mixing. Also, adding signals type 1, 2, and 3 does not remove left expansivity.

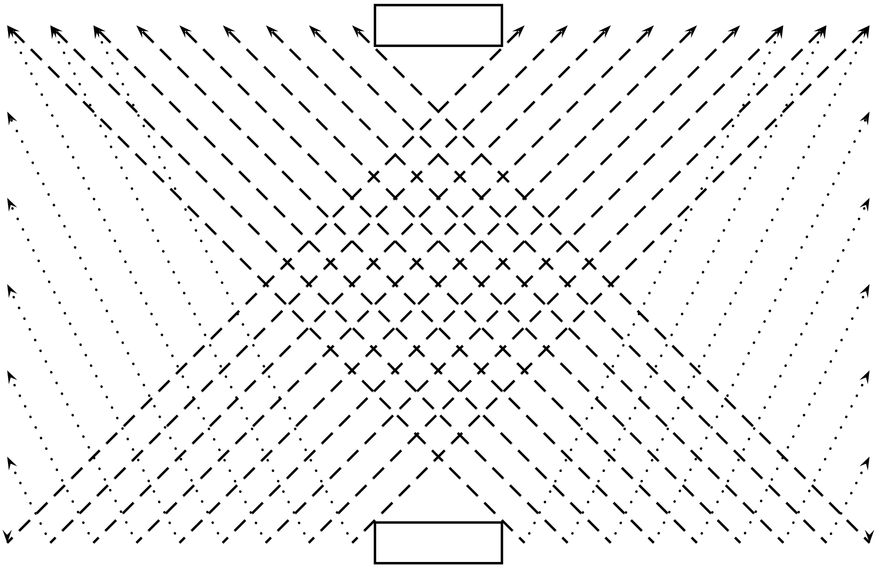


Fig. 13 If two configurations are sufficiently far apart in the same orbit, the type 1 and 2 signals can be used to cancel out tiling errors that would cause constraints between the cylinders of the two configurations. *Dashed lines* denote the type 1 signals, *dotted lines* denote the type 2 signals and the *rectangles* denote the cell state sequences defining the cylinders

It is now undecidable whether there exists such a configuration c that state $F^n(c)(0)$ contains none of the signals type 1, 2, or 3 for any integer n. □

Interpreting the definition of local immortality problem in a different way, Corollary 3 follows.

Corollary 3 *Given a left expansive and mixing cellular automaton* $(A^{\mathbb{Z}}, F)$ *and clopen set* $C \subseteq A^{\mathbb{Z}}$*, it is undecidable whether there exists such a configuration* $c \in C$ *that* $F^n(c) \in C$ *for every* $n \in \mathbb{Z}$.

4.3 Left Expansivity is Undecidable

In this section, it is shown that is undecidable whether the given cellular automaton is left expansive. However, it still remains an open problem whether expansivity or positive expansivity is a decidable property.

Theorem 7 *Given a reversible cellular automaton* $(A^{\mathbb{Z}}, F)$*, it is undecidable whether* F *is left expansive.*

Proof Let $((T \cup S)^{\mathbb{Z}}, G)$ be the cellular automaton of Theorem 5. The states that do not contains the signals generated by tile T_2 are denoted by T and the states that do contain the signals are denoted by S. The new state set is $A = (T \cup S) \times \{0, 1\} \times \{0, 1\}$. The new rule is defined by

$$F(c_1, c_2, c_3)(i) = \begin{cases} (G(c_1)(i), \sigma(c_2)(i), c_3(i)) & \text{if } c_1(i) \in T, \\ (G(c_1)(i), c_3(i), \sigma(c_2)(i)) & \text{if } c_1(i) \in S, \end{cases}$$

where $(c_1, c_2, c_3) \in A$. That is, the new cellular automaton contains three different layers. On the first layer, the original cellular automaton is simulated. The second and the third layer contain only binary states that either remain in place or move one cell per time step to the left. If the state of the first layer belongs to S, then the second and the third component are interchanged. The idea is that if all the cells of the first layer are in states of T, the states of the second layer are moved one step to the left and the states of the third layer remain unchanged. The cellular automaton $(A^{\mathbb{Z}}, F)$ is left expansive if and only if the answer to the question of Theorem 5 is affirmative for $((T \cup S)^{\mathbb{Z}}, G)$.

Assume that for every initial configuration every cell of $((T \cup S)^{\mathbb{Z}}, G)$ must enter to a state in S at some point. Due to the compactness, this happens for every cell infinitely often and there exists such a bound N that during N time steps every cell must enter to a state in S at least once. This means that a binary state value of originally the third component of cell i must travel at least one cell to the left for every N time step. This means that no single state of the second of the third layer remains in place, but travel infinitely far to the left eventually. Hence, the new cellular automaton is left expansive.

Assume that for some initial configuration c cell i of $((T \cup S)^{\mathbb{Z}}, G)$ does not enter a state in S ever. Let c_0 be such a configuration that $c_0(j) = 0$ for every j. Let c_1 be such a configuration that $c_1(j) = 1$ if $i = j$ and $c_1(j) = 0$ otherwise. Then $d(F^n(c, c_0, c_0), F^n(c, c_0, \sigma^{-i}(c_1))) \geq \epsilon$ for any ϵ by choosing suitably large i. That is, the information of the difference only at the third component of cell i does not travel arbitrarily far to the left. Hence, the new cellular automaton is not left expansive. □

5 Discussion

We have introduced 2-way deterministic Wang tiles to prove undecidability results on one-dimensional reversible cellular automata. Our goal, yet unreached, was to use this approach to prove that expansivity is an undecidable property. To establish this, it would be sufficient to show that local immortality is undecidable among expansive cellular automata. We were able to prove this for left expansive automata (Theorem 5), which then directly yields the undecidability of left expansivity (Theorem 7).

Another, closely related open question is a conjecture of Nasu [19] that all expansive one-dimensional cellular automata are conjugate to subshifts of finite type.

It would seem that for our approach to work, a counter example to the conjecture would be needed. For example, it is easy to infer from the proof of Theorem 5 a left expansive cellular automaton in which possible history sequences of cells are not a subshift of finite type.

Finally, we point out that the property of being periodic (i.e. F^n being equal to the identity function for some positive n) was recently proved undecidable [13]. Also, it has been recently shown that sensitivity, transitivity and topological mixing are undecidable properties for reversible one-dimensional cellular automata [15]. Previously, the non-reversible case of sensitivity was known to be undecidable [6].

Acknowledgements The financial support by the Academy of Finland (grant 211967), the Finnish Cultural Foundation and the Fund of Vilho, Yrjö and Kalle Väisälä is gratefully acknowledged.

References

1. Amoroso S, Patt Y (1972) Decision procedures for surjectivity and injectivity of parallel maps for tessellation structures. J Comput Syst Sci 6:448–464
2. Bennett CH (1973) Logical reversibility of computation. IBM J Res Dev 6:525–532
3. Berger R (1966) The undecidability of the domino problem. Mem Am Math Soc 66:1–72
4. Blanchard F, Maass A (1997) Dynamical properties of expansive one-sided cellular automata. Israel J Math 99:149–174
5. Dubacq J-C (1995) How to simulate any Turing machine by reversible one-dimensional cellular automaton. Int J Found Comput Sci 6(4):395–402
6. Durand B, Formenti E, Varouchas G (2003) On undecidability of equicontinuity classification for cellular automata. Discrete Math Theor Comput Sci, pp 117–128
7. Finelli M, Manzini G, Margara L (1998) Lyapunov exponents versus expansivity and sensitivity in cellular automata. J Complex 14:210–233
8. Hooper P (1966) The undecidability of the Turing machine immortality problem. J Symbol Log 31(2):219–234
9. Hopcroft JE, Ullman JD (1979) Introduction to automata theory, languages and computation. Addison–Wesley, Readings
10. Kari J (1992) The nilpotency problem of one-dimensional cellular automata. SIAM J Comput 21:571–586
11. Kari J (1994) Reversibility and surjectivity problems of cellular automata. J Comput Syst Sci 48:149–182
12. Kari J (2005) Theory of cellular automata: a survey. Theor Comput Sci 334:3–33
13. Kari J, Ollinger N (2008) Periodicity and immortality in reversible computing. In: MFCS 2008. Lecture notes in computer science, vol 5162. Springer, Berlin, pp 419–430
14. Kurka P (2001) Topological dynamics of cellular automata. In: Markus B, Rosenthal J (eds) Codes, systems and graphical models. IMA volumes in mathematics and its applications, vol 123. Springer, Berlin, pp 447–498
15. Lukkarila V (2008) Sensitivity and topological mixing are undecidable for reversible one-dimensional cellular automata. J Cell Autom (submitted)
16. Lukkarila V (2009) The 4-way deterministic tiling problem is undecidable. Theor Comput Sci 410:1516–1533
17. Manna Z (1974) Mathematical theory of computation. McGraw–Hill, New York
18. Morita K, Harao M (1989) Computation universality of one-dimensional reversible (injective) cellular automata. Trans IEICE Jpn E72:758–762

19. Nasu M (1995) Textile systems for endomorphisms and automorphisms of the shift. Mem Am Math Soc, vol 546. AMS, Providence
20. Robinson RM (1971) Undecidability and nonperiodicity for tilings of the plane. Invent Math 12:177–209
21. Shereshevsky MA (1993) Expansiveness, entropy and polynomial growth for groups acting on subshifts by automorphisms. Indag Math N S 4:203–210

On Using Divide and Conquer in Modeling Natural Systems

Yaki Setty, Irun R. Cohen, Avi E. Mayo,
and David Harel

Abstract In recent years, we have been studying approaches to the realistic modeling of natural systems, especially biological systems. We have tested several of these in a project devoted to modeling pancreatic organogenesis, a complex system that dynamically promotes structural and molecular development. Here, we describe one of these approaches—a kind of 'divide and conquer' technique, in which the system is disassembled into modules to specify behaviors on the scale of the organ (i.e., morphogenesis) and the cell (i.e., molecular interactions). At run-time, these modules are re-assembled to direct development in individual cells. This approach employs multi-scaling and dynamics, two important characteristics of natural systems, but avoids cross-scaling. It thus appears to be useful for systems in which the importance of cross-scaling seems to be less significant, such as the development of phyllotaxis in plants. In pancreatic organogenesis, cross-scaling was found to be a significant characteristic, and thus by using 'divide and conquer' we could cover only its preliminary stages. We discuss the approach and our use of it, as well as he various methods to analyze the achievements and limitations of the end result.

1 Introduction

Natural systems, such as organs and organisms, are large-scale complex systems with numerous elements and interactions. Modeling such systems can lead to better understanding thereof and may help in efforts to save on resources and development time. During the past years, we have been focused on developing approaches for modeling natural systems. We developed several approaches to simulate the natural systems through changes in 4 dimensions: time and three dimensions of space. The chain of reactions leading to cellular and structural development emphasizes the need for multi-scaling. Furthermore, the importance of 4D dynamics in natural systems underlies the time-dependent developmental processes that affect the developing organ. Thus, for the very least, a plausible approach should conduct dynamic formation of a 3D structure.

Y. Setty (✉)
Computational Biology Group, Microsoft Research, Cambridge CB3 0FB, UK
e-mail: yaki.setty@microsoft.com

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_33, © Springer-Verlag Berlin Heidelberg 2009

We tested our approaches on pancreatic organogenesis, a complex system in which the pancreas is developed and formed. This system involves multiple scales, dynamics, and a unique 3D anatomic structure. During organogenesis, reactions in cells lead to cellular behavior and anatomic structuring of the organ. In turn, the cell's position feeds back and influences gene expression. In pancreatic organogenesis, cells act in concert and aggregate to a cauliflower-shape structure. Zooming into the formation process discloses the molecular interactions that drive each cell during its life cycle.

The end result approach, which we term *autonomous cell*, defines a cell as a 3D entity that senses its environment and acts accordingly. This approach was found very beneficial for modeling pancreatic organogenesis and conduct multi-scale, cross-scale, and dynamics as well as emergence of the 3D organ structure from a collection of molecular interactions. The approach enables integration of the genetic, molecular, and cellular components, along with environmental signaling into a dynamically running, three-dimensional simulation (see [40]).

However, as a step toward the *autonomous cell* approach, we have developed several approaches that served as important milestones in our work, and enhanced different aspects in the task of modeling natural systems. The 'divide and conquer' approach[1] that we describe here suggests decomposing a system into independent modules and reassembled them at run time. Accordingly, we defined two main modules; one specifies the structural development of the system and the other formalizes the molecular interactions in an individual cell.

In natural systems, such as pancreatic organogenesis, these two modules correspond to different scales of the system [7]. The specifications of molecular interactions correspond to behavior of the system at the cell scale, while the formation of the organ specifies the morphogenesis at the organ scale. From the biological point of view, the 'divide and conquer' approach suggests to separate scales in natural systems, namely, the behavior of the organ from the behavior of individual cells. Thus, this approach conducts multi-scaling and 3D dynamics, two important characteristics of realistic modeling [7].

To apply the 'divide and conquer' approach to pancreatic organogenesis, we formalized a eukaryotic cell as an object with a sub-object for its nucleus. We specified in a molecular interaction module the relevant interactions in a cell using the language of statecharts [16], as implemented in the Rhapsody tool [44]. Separately, we specified a module for structural development using a concept inspired by *sweeps to branched skeleton* algorithms [33]. Accordingly, an autonomous object serves as the morphogenesis module, which holds predefined skeletons of the structure at various developmental stages. The two modules direct at run time the development of instances of a cell in the simulation. Events that are generated by the molecular module drive the molecular development of cells, which are in parallel directed by the morphogenesis module toward the predefined skeleton.

[1] Here, we abuse the term 'divide and conquer', which is generally used in computer science to describe an algorithm design paradigm that works by recursively breaking down a problem into two or more sub-problems of the same (or related) type, until these become simple enough to be solved directly.

We linked the model to a 3D animated front-end (in 3DGameStudio [1]) and an analysis GUI (in MATLAB [23]), using the idea of Reactive Animation [10] and its generic platform [18]. At run-time, the simulation is visualized and analyzed at a front-end and a GUI, respectively.

To analyze the model, we separately examined the behavior of each module. The emerging structure was compared against images of 2D histological sections, while the molecular interactions were compared to a simple mathematical model that describes the kinetics of cell populations. The analysis revealed qualitatively similar results, but we found the overall approach insufficient for realistic modeling of the complete system. In particular, when we attempted to extend the coverage, we had to introduce cross-scaling to capture the interplay between structural development and molecular interactions (i.e., the two independent modules).

2 Pancreatic Organogenesis

In mice, pancreatic organogenesis is initiated on the eighth embryonic day, and is roughly divided into two transitions, primary and secondary [29]. During the *primary transition*, cells in the appropriate regions of the flat gut are specified as pancreatic and form a bud; during the *secondary transition*, the bud evolves and becomes a branched structure [20, 41]. Organogenesis depends on simultaneous interactions across molecular and morphogenetic mechanisms that act in concert to form the organ. The molecular mechanisms involve processes that regulate the differentiation and development of individual cells (see, e.g., Fig. 1, left), whereas the morphogenic mechanisms gather the cells together to form a cauliflower-like shaped organ (see, e.g., Fig. 1, right). Pancreatic morphogenesis is initiated by a budding process, which leads to thickening of the flat gut. Later, followed by mesenchyme intervention, the pancreas undergoes a process of branching to create the branched structure of the matured pancreas [20]. In parallel, different molecular processes (e.g., proliferation and pancreatic specification) promote the molecular development of the organ [4, 5, 20]. These processes are driven by interactions between different elements including intra-cellular signaling and cell-cell interaction [9, 21, 26, 37].

3 Methods

3.1 Modeling Approach

We studied the morphogenetic behavior and the molecular interactions in pancreatic organogenesis. Using the language of statecharts [16, 17], as it is implemented in the Rhapsody tool [44], we formalized the molecular interactions of a pancreatic cell. Statecharts define behavior using a hierarchy of states with transitions, events, and conditions. The language can be compiled into executable code using the Rhapsody tool. To promote aggregation of cells toward a specific structure, we employed a concept that directs objects toward a pre-defined fixed skeleton. This concept was

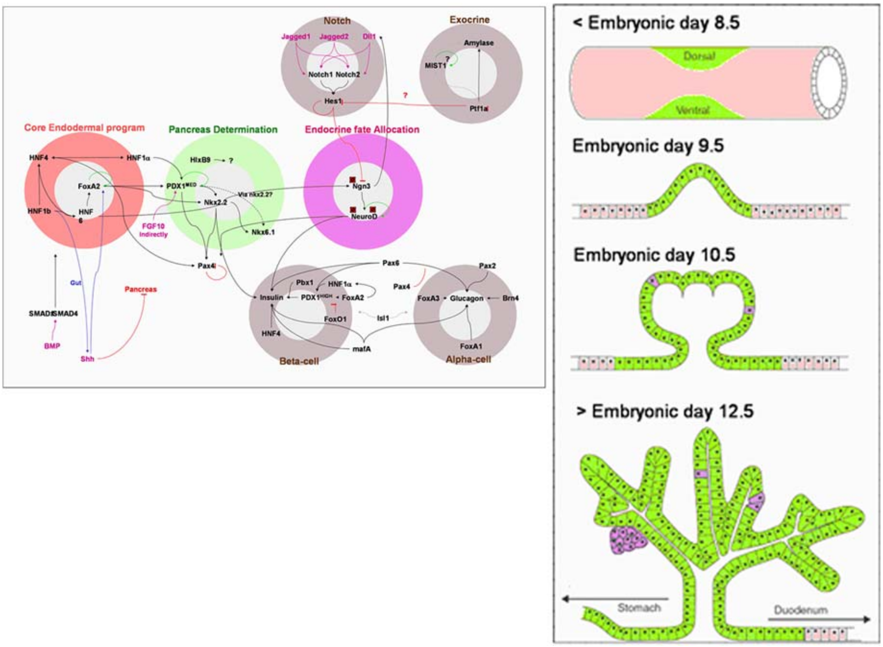


Fig. 1 *Left*: An illustration of the many molecular interactions driving the pancreatic differentiation in a single cell (adapted from [20]; Reprinted with permission of Wiley–Liss, Inc., a subsidiary of John Wiley & Sons, Inc.). *Right*: An illustration of pancreatic formation at four different time points, as depicted in the literature (adopted from [21])

inspired by sweeps to branched skeleton algorithms [25, 33] and was adjusted to pancreatic organogenesis by adding relevant features such as 3-dimensionality and proliferation.

To visualize the model, we used the idea of Reactive Animation [10], a technique that links a reactive executable model with an animated front-end to form a visualized, interactive, and dynamic model. Using a generic platform for Reactive Animation [18], we linked the model to a 3D animated front-end (in 3DGS [1]) and a mathematical GUI (in MATLAB). The front-end visualizes the simulation and provides the means to interact with it, while the mathematical GUI monitors and analyzes the progress. At run time, processes in the simulation are reflected in the front-end in different ways, e.g., by color and position changes. Separately, the mathematical GUI displays various graphs and statistics of the simulation. A prerecorded clip of the simulation at run-time is available at research.microsoft.com/~yakis/runs.

3.2 *Modeling Molecular Interactions*

A eukaryote cell consists of many concurrent sub-cellular and molecular mechanisms that drive development and function over its life cycle. Each sub-cellular

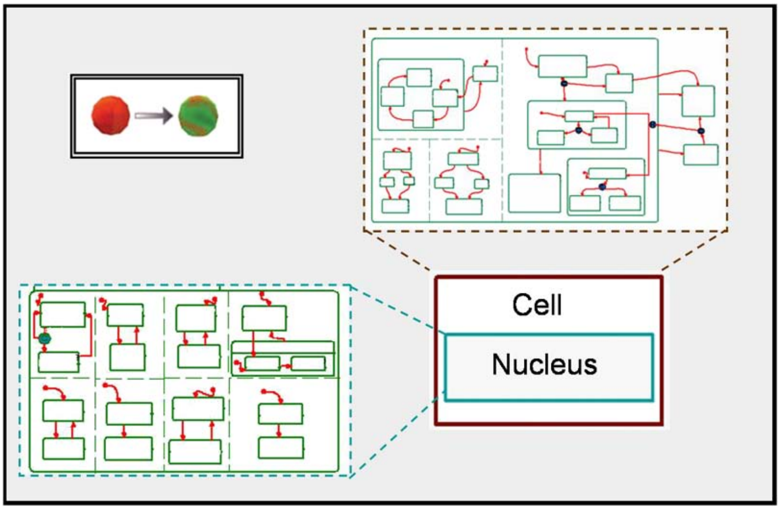


Fig. 2 Modeling eukaryotic cell. The two objects, `Cell` and `Nucleus`, accompanied by a pseudo statechart for their behavior

element consists of many concurrent processes and mechanisms that dynamically drive the cell's function over time. In the model, we formalized a cell as an object and its nucleus as a sub-object to indicate (strong) composition between them. Accordingly, the `Nucleus` object specifies behavior for the gene expression in the cell, while the `Cell` object itself specifies the behavior of molecular mechanisms (e.g., proliferation). This setup is illustrated in Fig. 2, which shows the cell object accompanied by schematic versions of their statecharts.

The nucleus, the core of a cell, contains the DNA and consists of genes that regulate its development. The genes are expressed in response to various signals in the cell. Genes express proteins that influence the cell's behavior. To model the nucleus, we took a simplistic approach, defining each gene as an independent component that can be either `Expressed` or `Unexpressed`. The effect of gene expression is diverse and depends on the process. Gene expression is visualized in different manners. For example, in the case of markers, proteins that determine differentiation, expression is visualized by color changes of the animated cell. In pancreatic organogenesis, one important gene is PDX1, which is considered as the pancreatic marker. In the model, when the active state of the `PDX1` component moves to `Expressed` the corresponding animated figure changes its color from red to green. The color change indicates that this specific cell accomplished pancreatic specification and is now specified as pancreatic progenitor and no longer as an endodermal cell.

The `Cell` itself describes the behavior of various molecular mechanisms (such as, differentiation, proliferation, death) in a cell during its lifespan. This element also carries the spatial 3D coordinates of the cell and updates their values at run-time as the simulation progresses. We specify the mechanisms as orthogonal components, which at run-time act concurrently to drive the cell's behavior over time. As an example, consider the cell proliferation process. The `Proliferation` component defines a state for each stage of the cell cycle. At run-time, when the `Proliferation` ends (i.e., its active state moves to state `M`), the `Cell` duplicates itself by creating an identical `Cell` instance (i.e., a new instance of a cell is created and its active states are set to be identical to its parent). At the front-end, an additional animated cell is created in the appropriate location, which was calculated during its parent division.

3.3 Modeling Structural Formation

To model the morphogenetic behavior of cells, we use a concept inspired by *sweeps to branched skeleton* algorithms (STBS). STBS algorithms are used to model lobed leaves by predefining a branch skeleton to reconstruct the plant structure [31, 33]. In the past, these algorithms were mainly used to extract the structure of a 2D scanned object. In organogenesis of natural systems, we have to consider a population of objects that, among other things, proliferate and interact. Thus, we adjusted the concept to support morphogenesis of proliferating population in 3D.

As mentioned, among other things, we assigned a property in the `Cell` object that specifies its spatial coordinates. Accordingly, when a newborn `Cell` is initiated, its spatial properties are updated based on its parent location. In addition, we defined another object, which utilizes pre-defined 3D skeletons that determines generic structures of the organ at different stages. This STBS object corresponds to mechanisms in the environment that promote cells to aggregate and form the pancreatic structure. At run time, the object directs cells by minimizing the distance between the cell the skeleton. Thus, cells are 'swept' toward the skeleton to form the branched structure of the pancreas. At the front-end, the animated cells continuously update their positions. As the animation progresses, the visualization discloses how the population aggregates to form the branched structure of the pancreas.

3.4 Combining the Two: The Simulation at Run-Time

When the model is executed, instances of the `Cell` (Fig. 2) are created and appear in the front-end as a sheet of red spheres on the proper location at the flat endodermal `Gut`. Once a `Cell` instance is created, one state in each concurrent component of the statechart is set to be an active state. At this point, the `Cells` are uniform and their active states are set to the initial states (designated by a stubbed arrow). As the simulation advances, cells respond to various signals by changing their active states accordingly. Hence, the sheet loses uniformity at a very early stage of the simulation.

To illustrate the simulation in progress, consider a conceptual process that unifies many biological processes such as signaling pathways or cell migration. Such a process is stimulated by a signal that initiates a chain of various reactions. Consequently, events are sent to the `Nucleus`, which initiates expression in various `Genes`. In turn, the active state in the relevant components moves to the `Expressed` state and the corresponding animated cell changes its color. Eventually, an event is generated and the relevant molecular mechanisms move to new states. For example, an event may promote a cell to proliferate, and thus the active state in the `Proliferation` component in the `Cell` becomes `M`. Consequently, the cell duplicates itself, and a new animated cell appears in the front-end. In parallel, cells are specified as pancreatic and change the color of their corresponding spheres (Fig. 2 top-left).

At the same time, the STBS object directs cell movement towards the predefined skeleton. Accordingly, the spatial properties of cells are continuously updated to simulate cell migration. At the front-end, animated cells change their location accordingly to form the pancreatic branched structure. When a cell proliferates, the new instance interacts with the STBS object and is directed based on its parent position. Other molecular mechanisms, such as differentiation or specification, act concurrently. The simulation achieves equilibrium when cells are differentiated and are located in a proper position on the skeleton, where they cannot proliferate anymore.

4 Results

4.1 Behavior of Population Fits Simple Mathematical Model

To analyze the molecular development of the cell population in the simulation, we recorded the cell count of the endodermal and pancreatic populations over time. Each run of the simulation generated a slightly different growth, but the overall course maintained similar characteristics. Figure 3 (left) shows a typical time dependent molecular development of endodermal (red), pancreatic (green), and overall cell population (black).

We observed three major regimes. In the primary regime (embryonic day 8–9) endodermal cells proliferate but do not specify as pancreatic. Thus, the endodermal population increases but the pancreatic one does not. At the secondary regime, specification signals are introduced and endodermal cells start specifying as pancreatic cells and the proliferation continues in both. In this regime, we observed decrease of the endodermal population and a rapid growth in the pancreatic one. In the last regime (starts approximately at embryonic day 13), both populations enter a steady state, in which the endodermal population is fully specified and the pancreatic population achieved maximum.

To test the plausibility of the results, we constructed a simple mathematical model that simulates the molecular behavior of cell populations in the model. We formalized the system using ordinary differential equations (see, e.g., [8, 38, 39]) to describe the kinetics of cell population. The model (1) considers a population of endodermal cells (E) and a population of pancreatic cells (P). E proliferates at rate

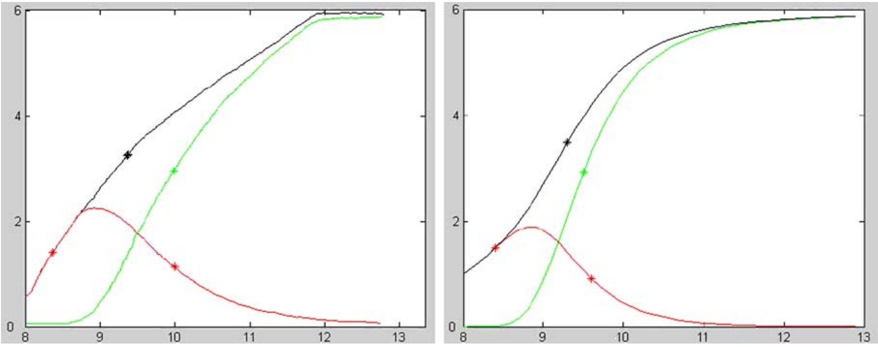


Fig. 3 Molecular development in pancreatic organogenesis: Cell count (in thousands) of endodermal (*red*), pancreatic (*green*) and overall (*black*) population as function of time (in embryonic days). (Notice the star symbol, indicating the median values.). *Left*: Cell count in the computational statechart-based model. *Right*: Mathematical ODE-based model with the parameters $k_p = 1$, $k_s = 3$, $f_p = 4/(t^n + 4)$ and $f_s = t^n/(t^n + 1)$

k_p and specifies as pancreatic in rate k_s, which is regulated by additional signals formalized in f_s. The pancreatic population P is increased by the specification and proliferation processes. Pancreatic cells proliferate at the same rate an endodermal but are bounded by additional signals, thus we assign the f_p element to describe effect of such. Figure 3 (right) shows a solution for the mathematical model. This result manages to reproduce the behavior of the cell population and the three regimes described above.

$$\begin{aligned} dE/dt &= k_p E - k_s f_s E, \\ dP/dt &= k_p f_p P + k_s f_s E. \end{aligned} \tag{1}$$

The difference between the two models emerges from the modeling perspective. While the mathematical model describes the behavior of cells as a population, the computational model specifies behavior of an individual cell and the behavior of the population emerges from execution of many instances with the same specification. Although simplified, the mathematical model revealed similar characteristics for the populations as emerged from the computational simulation. Here, we used the mathematical model for qualitative analysis only and did not examine the model in detail. However, a quantitative analysis is possible, but is beyond the scope of this paper.

4.2 The Emerging Structure Generates Histological Sections

As the simulation advances, cells proliferate and are directed by the morphogenesis module toward the branched structure of the pancreas. Figure 4 shows four steps in the formation of the structure from flat sheet into the matured pancreas

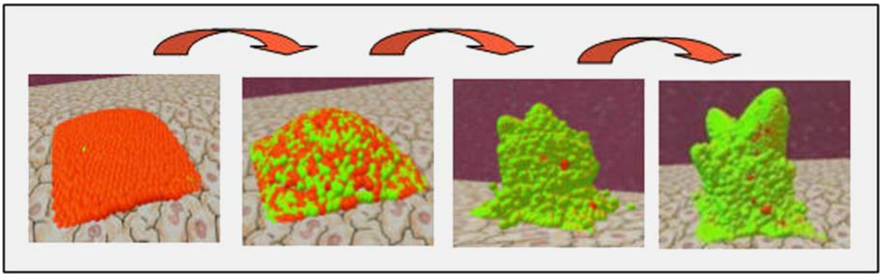


Fig. 4 Four snapshots of the structure formation in the simulation: endodermal (*red*) and pancreatic (*green*) cells act in concert to form the branched pancreatic structure

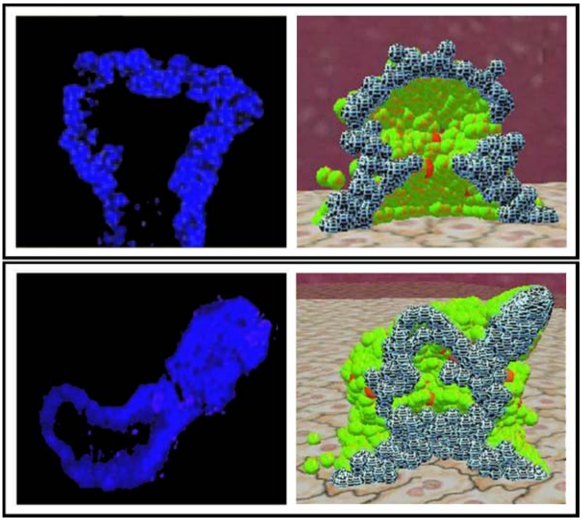


Fig. 5 Comparison of the emerging structure and 2D histological sections at embryonic day 10 (*top*; adapted from [20]) and 10.5 (*bottom*; contributed by Dor's lab)

(see a recorded clip at research.microsoft.com/~yakis/branching). To compare the 3D emerging structure with the two dimensional data (e.g., Fig. 1, right), we enabled a 'halt and slice' option in the animated front-end. Accordingly, at any stage of the simulation, the user can halt the simulation and view slices of the emerging structure over the x- and the y-axis. The cross-section slices are then compared with the histological section. A comparison of 2D histological sections (left) and cross sections of the simulated structure (right) is shown in Fig. 5. The results indicated that the emerging structure manages to qualitatively illustrate somewhat of the dynamic of early stages of pancreatic morphogenesis. Notice that in addition to the structural formation, the molecular behavior is also visualized in the front-end at run time. For example, cells proliferate by creating new instances and change their properties (e.g., color) to indicate gene expression. It is important to emphasize that the molecular changes are not effected by the cell position, which is being regulated separately.

4.3 The Need for Cross-scaling in Pancreatic Organogenesis

As mentioned, the ‘divide and conquer’ approach, by its very nature, separates the molecular interactions from the structural formation. Thus, it does not conduct the interplay between main modules. To proceed with realistic modeling of pancreatic organogenesis, in particular to advance to the ‘secondary transition’ in which cells adopt their final fate, we had to introduce cross-scaling [7] to enable interplay between the molecular interactions and the structural formation.

The principles underlying cross-scaling in modeling natural systems are illustrated in Fig. 6. In the ‘divide and conquer’ approach (Fig. 6, top), a module for molecular interactions (left) is separated from the structural behavior (right). At run time, the two modules interact with cells (middle), to drive in parallel their molecular and structural development. An approach that considers cross-scaling (Fig. 6, bottom) links the two modules and enables interplay between them. At run time, molecular interactions and structural information direct cell development as before, but also interact with each other to promote different aspects in their development.

The Delta–Notch mechanism emphasizes the need for cross-scaling in organogenesis. In pancreatic organogenesis, this mechanism directs cells towards their final fate. It is partly initiated by external signals (from the extra-cellular space) and is enhanced by cell-cell interaction. At the same time, cells that adopted an endocrine fate, aggregate to form clusters named ‘islet of Langerhans’. In this case, the stage of a cell is significantly determined by external signals in its spatial position. Thus, the molecular development relies much on morphogenesis. In turn, a cell that adopted an endocrine fate, starts aggregating in a cluster, and thus the molecular interactions determine the structural formation. Such a mechanism forces cross-scaling for simulating its behavior.

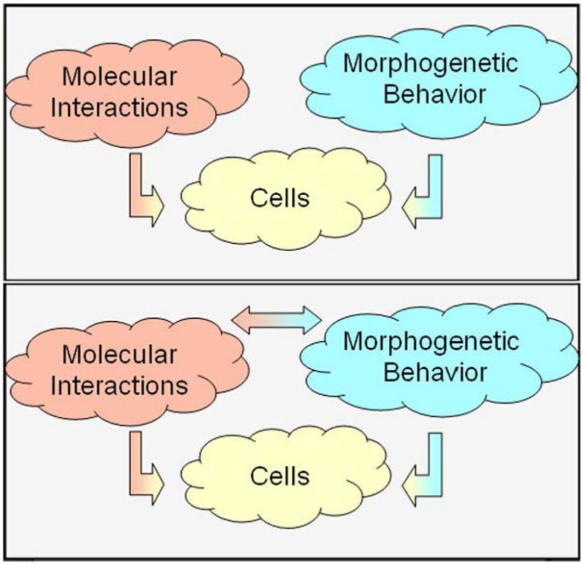


Fig. 6 *Top*: An illustration of the interaction scheme between the cells and the morphogenetic and molecular interactions in the model. *Bottom*: Cross-scaling in modeling (notice the extra arrow indicating interactions between the modules)

5 Discussion

During the last decade, increasing interdisciplinary work modeled aspects in natural systems to explain relevant experimental findings (see, e.g., [2, 6, 14, 27]). However, most of this work describes the system at a specific scale and ignores others. Moreover, the work focuses on a single mechanism in each scale and disregards multiple interactions. Another type of modeling work formalizes gene expression and protein activity using a variety of mathematical and computational tools (for example, see [3, 19, 30, 34, 35]). However, most of the relevant work ignores multiple concurrent activities and focuses on a single mechanism in the entire system. An example of comprehensive modeling is the beating heart project [28], which over the last fifty years formalized the electric activities in the heart. However, by its mathematical nature, the model is not interactive and does not support the kind of visualization we seek.

Recently, various papers use computational modeling approaches for natural systems. In [13], hybrid automata are used to model the Delta–Notch mechanism, which directs differentiation in various natural systems. In [11], computational challenges of systems biology are described and various approaches for achieving them are discussed. A similar motivation for model-driven engineering approaches is discussed in [36]. In [12], computational and mathematical approaches are reviewed and the term *executable biology* is used to describe the kinds of modeling carried out in our group, and recently also elsewhere. In [45], a model for a eukaryotic cell is built, in which a UML class diagram was used to formalize the relations between a cell and its sub-cellular elements. The setup was empowered by specifying behavior of different cell types (e.g., red blood cell) using the ROOM formalism. A similar approach was employed in [43] to model the Ethylene-Pathway in Arabidopsis thaliana using statecharts and LSCs.

To simulate the development of natural systems, we confront a complex system that dynamically evolves over time and space. This task requires one to understand the many time-dependant aspects of the system. A comprehensive modeling approach of such systems should provide the means to specify behavior of its different aspects [7]. Moreover, the approach should deal with the system on different levels of abstraction, which in turn drive different scales of the development. At the very least, such an approach should provide the means to simulate the dynamics of formation of a 3D structure.

The 'divide and conquer' approach presented here was inspired by many industrial reactive systems, which can be separated into fundamental components that interact at run time. In natural systems, this approach provides the means to specify dynamic behavior of the structural and molecular development of a natural system as separated modules. From the biological point of view, these modules correspond to different scales of the system, organ, cell, gene, etc. However, by its nature, the approach does not provide the means to specify interplay between the different modules. Thus, it conducts multi-scaling and 3D dynamics but avoids cross-scaling.

In this paper, we describe how the 'divide and conquer' approach was employed for modeling pancreatic organogenesis. The end result gave rise to dynamic representation of the system and managed to capture the preliminary stages of its development. Furthermore, the model provided plausible results for the preliminary period. However, more complex mechanisms of the system could not be specified using this approach. These mechanisms involve interplay between structural and molecular behaviors, which are specified separately in the model. Thus, we move that this approach can be used to model such systems as the development of phyllotaxis in plants, which might be separated into fundamental modules. For example, structural phyllotaxis principles can be specified as L-Systems [22, 32], while molecular behavior can be formalized using deferential equations. Furthermore, existing models (such as, e.g., [15, 24, 42]), which describe structural formation of natural systems, can be extended using the 'divide and conquer' approach to support molecular behavior.

In the case of pancreatic organogenesis, the 'divide and conquer' approach emphasized the need for cross-scaling in organogenesis, and served as a significant milestone in our work toward the model described in [40], which employs the 'autonomous cell' approach.

Acknowledgements We thank Yuval Dor and Judith Magenheim for providing histological images of the pancreas.

References

1. 3D Game Studio. www.3dgamestudio.com
2. Axelrod JD (2006) Cell shape in proliferating epithelia: a multifaceted problem. Cell 126:643–645
3. Cardelli L (2005) Abstract machines of systems biology. Trans Comput Syst Biol 3:145–168
4. Chakrabarti SK, Mirmira RG (2003) Transcription factors direct the development and function of pancreatic beta cells. Trends Endocrinol Metab 14:78–84
5. Chu K, Nemoz-Gaillard E, Tsai MJ (2001) BETA2 and pancreatic islet development. Recent Prog Horm Res 56:23–46
6. Ciliberto A, Novak B, Tyson JJ (2003) Mathematical model of the morphogenesis checkpoint in budding yeast. J Cell Biol 163:1243–1254
7. Cohen IR, Harel D (2007) Explaining a complex living system: dynamics, multi-scaling and emergence. J R Soc Interface 4:175–182
8. Edelstein-Keshet L (2005) Mathematical models in biology. Society for Industrial and Applied Mathematics, Philadelphia
9. Edlund H (2002) Pancreatic organogenesis—developmental mechanisms and implications for therapy. Nat Rev Genet 3:524–532
10. Efroni S, Harel D, Cohen IR (2005) Reactive animation: realistic modeling of complex dynamic systems. IEEE Comput 38:38–47
11. Finkelstein A, Hetherington J, Li L, Margoninski O, Saffrey P, Seymour R, Warner A (2004) Computational challenges of systems biology. IEEE Comput 37(5):26–33
12. Fisher J, Henzinger TA (2007) Executable cell biology. Nat Biotechnol 25:1239–1249
13. Ghosh R, Tomlin C (2004) Symbolic reachable set computation of piecewise affine hybrid automata and its application to biological modelling: delta–notch protein signalling. Syst Biol (Stevenage) 1:170–183
14. Gibson MC, Patel AB, Nagpal R, Perrimon N (2006) The emergence of geometric order in proliferating metazoan epithelia. Nature 442:1038–1041

15. Gorgevik D, Loskovska S, Mihajlov D (1994) Lindenmayer system application on the human kidney arterial system. In: Proceedings of the 12th international congress of the European federation for medical informatics, pp 127–131
16. Harel D (1987) Statecharts: a visual formalism for complex systems. Sci Comput Program 8:231–274
17. Harel D, Gery E (1997) Executable object modeling with statecharts. IEEE Comput 30:31–42
18. Harel D, Setty Y (2007) Generic reactive animation: realistic modeling of natural complex systems (submitted)
19. Heath J, Kwiatkowska M, Norman G, Parker D, Tymchyshyn O (2006) Probabilistic model checking of complex biological pathways. In: Priami C (ed) Proceedings of the computational methods in systems biology (CMSB'06). Lecture notes in bioinformatics, vol 4210. Springer, Berlin, pp 32–47
20. Jensen J (2004) Gene regulatory factors in pancreatic development. Dev Dyn 229:176–200
21. Kim SK, MacDonald RJ (2002) Signaling and transcriptional control of pancreatic organogenesis. Curr Opin Genet Dev 12:540–547
22. Lindenmayer A (1968) Mathematical models for cellular interaction in development. J Theor Biol 18:280–315
23. The MathWorks. www.mathworks.com
24. Mundermann L, Erasmus Y, Lane B, Coen E, Prusinkiewicz P (2005) Quantitative modeling of arabidopsis development. Plant Physiol 139:960–968
25. Mundermann L, MacMurchy P, Pivovarov J, Prusinkiewicz P (2003) Modeling lobed leaves. In: Proceedings of computer graphics international
26. Murtaugh LC, Melton DA (2003) Genes, signals, and lineages in pancreas development. Annu Rev Cell Dev Biol 19:71–89
27. Nelson CM, Vanduijn MM, Inman JL, Fletcher DA, Bissell MJ (2006) Tissue geometry determines sites of mammary branching morphogenesis in organotypic cultures. Science 314:298–300
28. Noble D (2005) The heart is already working. Biochem Soc Trans 33:539–542
29. Pictet RL, Clark WR, Williams RH, Rutter WJ (1972) An ultrastructural analysis of the developing embryonic pancreas. Dev Biol 29:436–467
30. Priami C, Quaglia P (2004) Modelling the dynamics of biosystems. Brief Bioinform 5:259–269
31. Prusinkiewicz P (2004) Modeling plant growth and development. Curr Opin Plant Biol 7:79–83
32. Prusinkiewicz P, Hanan J (1989) Lindenmayer systems, fractals and plants. Springer, New York
33. Prusinkiewicz P, Rolland-Lagan AG (2006) Modeling plant morphogenesis. Curr Opin Plant Biol 9:83–88
34. Regev A, Shapiro E (2002) Cellular abstractions: cells as computation. Nature 419:343
35. Regev A, Silverman W, Shapiro E (2001) Representation and simulation of biochemical processes using the pi-calculus process algebra. In: Pacific symposium on biocomputing, pp 459–470
36. Roux-Rouquié M, daRosa DS (2006) Ten top reasons for systems biology to get into model-driven engineering. In: GaMMa'06: proceedings of the 2006 international workshop on global integrated model management. New York, ACM, pp 55–58
37. Schonhoff SE, Giel-Moloney M, Leiter AB (2004) Minireview: development and differentiation of gut endocrine cells. Endocrinology 145:2639–2644
38. Segel LA (1984) Modeling dynamic phenomena in molecular and cellular biology. Cambridge University Press, New York
39. Segel LA (1991) Biological kinetics. Cambridge University Press, New York
40. Setty Y, Cohen IR, Dor Y, Harel D (2008) Four-dimensional realistic modeling of pancreatic organogenesis. Proc Natl Acad Sci USA 105(51):20374–20379
41. Slack JM (1995) Developmental biology of the pancreas. Development 121:1569–1580
42. Smith RS, Guyomarc'h S, Mandel T, Reinhardt D, Kuhlemeier C, Prusinkiewicz P (2006) A plausible model of phyllotaxis. Proc Natl Acad Sci USA 103:1301–1306

43. Taubner C, Merker T (2005) Discrete modelling of the ethylene-pathway. In: ICDEW'05: proceedings of the 21st international conference on data engineering workshops. Washington, IEEE Computer Society, p 1152
44. Telelogic. www.telelogic.com
45. Webb K, White T (2006) Cell modeling with reusable agent-based formalisms. Appl Intell 24(2):169–181

No Molecule Is an Island: Molecular Evolution and the Study of Sequence Space

Erik A. Schultes, Peter T. Hraber, and Thomas H. LaBean

Abstract Our knowledge of nucleic acid and protein structure comes almost exclusively from biological sequences isolated from nature. The ability to synthesize arbitrary sequences of DNA, RNA, and protein in vitro gives us experimental access to the much larger space of sequence possibilities that have not been instantiated in the course of evolution. In principle, this technology promises to both broaden and deepen our understanding of macromolecules, their evolution, and our ability to engineer new and complex functionality. Yet, it has long been assumed that the large number of sequence possibilities and the complexity of the sequence-to-structure relationship preempts any systematic analysis of sequence space. Here, we review recent efforts demonstrating that, with judicious employment of both formal and empirical constraints, it is possible to exploit intrinsic symmetries and correlations in sequence space, enabling coordination, projection, and navigation of the sea of sequence possibilities. These constraints not only make it possible to map the distributions of evolved sequences in the context of sequence space, but they also permit properties intrinsic to sequence space to be mapped by sampling tractable numbers of randomly generated sequences. Such maps suggest entirely new ways of looking at evolution, define new classes of experiments using randomly generated sequences and hold deep implications for the origin and evolution of macromolecular systems. We call this promising new direction sequenomics—the systematic study of sequence space.

1 Introduction

Almost a century ago, as it came to be understood that protein and RNA molecules were linear chain polymers, it also became clear that the evolution of these molecules occurred as natural selection chose from among a vast sea of sequence possibilities. Although extraordinary progress has been achieved in understanding the sequence, structure, function, and evolution of biological proteins and RNAs; this corpus of theory has had little to say about the properties of the much larger space of potential, yet unrealized sequences. Since our knowledge of molecular structure remains idiosyncratic to the vanishingly small, profoundly biased biological sampling of the enormous space of possible sequences, it has been impossible to draw

E.A. Schultes (✉)
Department of Computer Science, Duke University, Durham, NC 27708, USA
e-mail: schultes@hedgehogresearch.info

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_34, © Springer-Verlag Berlin Heidelberg 2009

truly general conclusions about molecular structure and evolutionary history. For example, without unevolved sequences in the analysis, it is impossible to resolve properties of proteins and RNAs that must be true in principle (due to physiochemical properties), from those that are the result of historically contingent accidents propagated by evolutionary descent. Hence, it is impossible to understand the ultimate origins of complex protein and RNA structures and to account for evolutionary transformations from one structure to another.

The explanation for why unevolved sequences have remained so neglected can be found in the historical development of molecular evolutionary theory. This began when concepts of molecular evolution were being forged under the twin influences of Darwin's theory of evolution and the new science of structural biology at the molecular scale. Experimental discoveries in protein and RNA biochemistry revealed molecular structures of unprecedented size and complexity, and discovery of highly ordered yet aperiodic structure in these macromolecules needed an explanation that chemistry and physics at the time could not offer. As it was then understood, the conformational entropy of molecules of such size should overwhelm the folding process, and thus the 3-dimensional (3D) structures of proteins and RNAs should resemble amorphous materials, such as liquids or glasses, rather than the observed, highly-ordered structures, more reminiscent of organic crystals. In the absence of a physical explanation, Darwin's theory of natural selection was used to escape from this dilemma: vast periods of time (billions of years) and shear numbers of trials (global populations of organisms) allowed nature to find those exceedingly rare sequences that fortuitously had the ability to overcome conformational entropy and fold to stable 3D structures. Supplanting God as the creator of life, natural selection also became the accepted mechanism accounting for the "miracle" of folded proteins and RNAs. This early fusion of structural biology and Darwinian theory implied that unevolved sequences were disordered, and so they became scientifically irrelevant. Despite ample evidence to the contrary, this assumption took root early and continues to influence thinking today.

In what follows, we first outline the historical development of molecular evolutionary theory in order to trace the origins of the assumption that sequences coding for well-ordered molecular structures can only be located through the strenuous labors of natural selection and that unevolved sequences are predestined to be unstructured. From this point of view, much of what we have to say about proteins is true for RNA and vice versa, and we will interchangeably draw on examples from each. We then summarize a few recent approaches that have challenged this assumption and have not only legitimized the analysis of the structures of unevolved sequences, but have gone on to embrace the much larger space of sequence possibilities as a missing, though essential component of a complete molecular evolutionary theory. We then conclude with what we view as a promising new direction for molecular evolution in a post-genomic era—a research program we call *sequenomics*.

2 The Delicate Clockwork Hypothesis of Molecular Evolution

It is hard to believe that within living memory, there was a time when molecular sequence and structural information for protein and single-stranded RNA molecules was non-existent (for brief reviews, see [51, 68]). At the risk of oversimplification, it can be said that our current understanding of RNA and protein structure has emerged from three fundamental experimental discoveries (recognized by five Noble Prizes) all before circa 1960.

First, through a series of genetic experiments on the bread mold *Neurospora*, Beadle and Tatum [3] had established a correspondence between genes and protein enzymes, formulating the famous *one-gene-one-enzyme hypothesis*.

Second, using analytical centrifugation, Theodore Svedberg and his colleagues had established definitively that proteins were macromolecules of unprecedented size (not colloidal aggregates as had been conjectured) [64]. In conjunction with the first protein sequences that were chemically determined (Sanger's work on insulin, [55]) and the first protein structures that were solved by x-ray crystallography (Perutz's and Kendrew's work on haemoglobin [20] and myoglobin [28]), it became clear that *function at the molecular level* (*specific binding and catalysis*) *could be rationalized by atomic-scale molecular structure*.

Third, expanding on the ideas of Mirsky and Pauling [45], and working with the protein ribonuclease, Anfinsen demonstrated that the sequence information of the polypeptide chain can be sufficient to account for the acquisition of the functional, folded structure known as the *native* fold (1957, reviewed in [1]). As it was assumed that the native conformation was also the minimum free energy conformation (an assumption consistent with contemporary structural studies on crystals and small molecules), Anfinsen's proposal became known as the *thermodynamic hypothesis*.

Taken together, these seminal experiments link evolution, sequence information, folded structure, and biochemical function into a single, coherent framework that can be expressed via the well-known mantra: *sequence dictates structure*, *structure dictates function*. The relationship between sequence and structure implies that if the rules governing folding can be deduced, then it might be possible to predict structure (and function) from sequence information alone, a proposition referred to as the *folding problem*. Concurrent discoveries in nucleic acids also suggested an analogous sequence-structure folding problem for single-stranded RNAs [15]. It is notable, however, that these experiments say nothing regarding the structures of unevolved molecules.

At the time when the first RNA and protein x-ray structures became available, conformational studies of molecules had focused on low molecular weight organic structures having few conformational states. But as single macromolecules, it was unclear how conformational entropy could be overcome. Instead, proteins and RNA were expected to access a large number of conformations either dynamically (i.e., as "random-coils") or as a disordered collapsed (i.e., as a "glass") [9]. Yet, as Anfinsen had demonstrated, despite the vast number of possible conformations, nature could solve the folding problem in seconds.

As Kauffman reviews in his *Origins of Order* [31], natural selection had come to be seen, for a variety of different reasons, as the sole source of order in biology. In the absence of a physical theory of complex polymer folding, researchers adopted natural selection as an explanation for how proteins and RNAs acquired unique folded conformations. For example, Levinthal had noted early on that the number of possible conformations in protein polymeric chains is far too large to allow exhaustive conformational searches for minimum free energy structures—a conundrum that came to be known as *Levithal's Paradox* [39, 40]. Levinthal noted that protein backbones have two, independent dihedral angles for each amino acid and additional rotations permitted for each side chain. Hence, for proteins of even modest length, say 150 amino acids (to use Levinthal's original formulation), there are 450 degrees of freedom with 10^{300} possible conformations. For RNA, the situation is worse, with six independent backbone angles, phosphate rotamers, distinct sugar conformations, and dozens of hydrogen-bond-mediated base interactions [4].

Levinthal's own answer to the Paradox was that the native, functional fold need not necessarily be a global minimum free energy structure (in contradiction to the thermodynamic hypothesis) and the exploration of conformation space by a protein need not be exhaustive. An exhaustive search implies that the energy associated with each conformation (the so-called energy landscape) is high (unstable) and approximately the same, thus permitting a random-walk. Levinthal postulated that in reality, the energy landscape must be structured into a kind of high-dimensional funnel with correlations between neighboring conformations that could constrain and expedite the folding process. Levinthal commented that this informed energy landscape was presumably an evolutionary adaptation, implying that unevolved proteins would have uncorrelated energy landscapes.

More recently, Frauenfelder and Wolynes [18], also starting with a "random-energy" model of protein folding, have described this evolutionary-induced modification of the energy landscape in their Principle of Minimum Frustration. In this case, the energy landscape is molded by evolution as a "random heteropolymer" is altered by mutation and selection in such a way as to minimize conflicting intramolecular interactions in the native fold. Folding funnels and minimally frustrated structures were used to explain folding in evolved proteins, however, uncorrelated, random energy landscapes were assumed to be reasonable approximations for unevolved molecules. This central role of natural selection in accounting for macromolecular conformations remains pervasive and continues to impact contemporary research. For example, the website for IBMs recent $100M Blue Gene Initiative in protein folding states unequivocally that unevolved "Heteropolymers typically form a random coil in solution and do not 'fold' to any reproducible structure in experimentally accessible times" and that "Arbitrary strings of amino acids do not, in general, fold into a well-defined three-dimensional structure."

The ability of a protein or RNA polymers to position thousands of atoms into energetically stable, kinetically accessible and functional configurations suggested that selection had to "work hard" to "engineer" the mysteriously complex solutions to the folding problem. Hence, evolved molecular structures came to be seen as delicate clockworks, carefully "designed" and "adjusted" as natural selection chose from among sequence possibilities. Just as randomly swapping gears and sprockets in a clockwork mechanism would almost certainly be disastrous, the random sequence mutations that are the basis of molecular evolution were expected to be almost always deleterious, if not catastrophic to the Ångstrom-scale architectures. For our purposes here, we refer to this perspective as the *Delicate Clockwork Hypothesis* (DCH) of molecular evolution.

The DCH was a reasonable response to the unprecedented complexity of native protein and RNA structures, especially at this time when natural selection dominated thinking in biology [21]. However, by the late 1960s, molecular sequence data began to contradict the DCH. It had become possible to observe that homologous proteins and RNAs (functionally equivalent molecules from different organisms, e.g., hemoglobin from cow, pig, shark, fish, and worms) had recognizable similarities in their sequences, and that there tend to be more similarities between sequences from closely related species [74]. It turned out that the rare mutations that had been accepted by natural selection, and accumulated over time, could thus be used to define the evolutionary relationships of different species. Genealogical lineages could be mapped, mutation-by-mutation, and for molecules like ribosomal RNA that were common to all species, it was possible to construct universal phylogenetic trees [71].

This diversity among observed molecular sequences was difficult to explain if the DCH was an accurate description of molecular structure. This prompted Salisbury [54] to point out that "If life really depends on each gene being as unique as it appears to be, then it is too unique to come into being by chance mutations." In an analysis of molecular evolution in sequence space that is curiously reminiscent of Levinthal's approach to folding in conformation space, Salisbury calculated the probabilities (as estimates of time) of finding a particular protein sequence by random mutation in the enormous space of sequence possibilities. In what we might refer to as *Salisbury's Paradox*, it was clear that under the prevailing idea of the uniqueness of the gene sequence space would be simply too large, and the number of sequences with folded structures too few for meaningful sampling by random mutation. The DCH assumes that there would be insufficient raw material (favorable mutations) for selection to work on.

The accumulating sequence data forced Salisbury to challenge what he called the "dogma of high gene specificity". Referencing a previous mathematical analyses by Quastler [50] who speculated that with respect to the amino acid sequence, "a certain neighborhood of structurally related amino acid polymers... can perform the same function" and "identical functions can be associated with multiple neighborhoods that are structurally unrelated." Salisbury concluded that biological functions could not be as rare or randomly distributed in sequence space as the DCH implied. Protein structures must somehow be insensitive to some, or even the majority, of amino acid substitutions, as if clockwork could be randomly rearranged and still keep perfect time. Apparently, in direct contradiction to the DCH, a significant, if unexpected, redundancy in protein sequences' encoding of structure was the key to molecular evolution. As Levinthal's Paradox had been solved with the idea of correlated energy landscapes, so was Salisbury's Paradox solved with correlations in sequence space. Functional proteins were not islands in sequence space, but archipelagos of interconnected sequences that folded to the same structures and having the same function. In either case, however, these isle refuges of coherent folding dotted a sea of disordered sequences.

Shortly thereafter, John Maynard Smith [62] offered a more rigorous solution to Salisbury's Paradox. Not only did Smith reconsider the fraction of molecular sequences that must be functional, but he also modeled the inherent organization of sequence space: "Suppose now that we imagine all possible amino-acid sequences to be arranged in a 'protein space', so that two sequences are neighbors if one can be converted into another by a single amino-acid substitution." Where A is the number of monomers ($A = 20$ amino acids for proteins, $A = 4$ nucleotides for RNA) and N is the length of the sequence, for any polymer sequence X, there are $(A - 1)N$ sequences that are single-step mutations. For functional proteins, Smith then defined the fraction, f, of these local neighbors that are at least as active as X. So long as the product of f and $19N$ is greater than 1, "meaningful proteins will form a network, and evolution by natural selection is possible." To demonstrate what he had in mind, Smith invoked the image of a popular word game whereby one word can be converted into another word by a series of one letter substitutions, each substitution creating a viable word. So, WORD can be converted into GENE as:

WORD
WORE
GORE
GONE
GENE

The analogy was that viable words were like functional proteins, and the substitution of letters like the substitution of amino acids. Depending on the size and extent of these networks of neutral sequence variants in sequence space (i.e., neutral networks), random mutations would always be able to access a sufficient number of viable sequences to ensure molecular diversification. As evidence that $f19N > 1$, Smith had cited the then recent paper by King and Jukes [30] that presented empirical evidence that a large fraction of amino acid substitutions are selectively neutral. Along with the work of Kimura [29], this observed preponderance of selectively neutral mutations resulted in the formulation of the then heretical *Neutral Theory* of molecular evolution. The Neutral Theory was in direct contradiction to the DCH and ignited surprisingly acrimonious debates that have been well documented (for example, see the Dibner Institute's excellent online resources for "Early Reception of the Neutral Theory" and "Ideology in the Neutralist-Selectionist Debates"). The vigor of these debates is testimony to the degree to which the DCH was held and defended. More recently, Meier and Özbek [44] acknowledge that even today "protein structures and their concurrent functions have remained largely mysterious, as the destruction of these structures by mutation seems far easier than their construction."

Although Smith's concept of neutral networks was consistent with Salisbury's and Quastler's analyses, it takes the extra step of considering sequence space, not as a grab-bag of probabilities, but as a network or graph, with definite relations, subject to mathematical analysis, metrics, and the possibility of establishing coordinate systems. Smith was envisioning molecular evolution as a diffusion process on neutral networks and molecular phylogenies as “samplings” of such neutral networks.

The diversity of molecular sequence data suggests that neutral networks must be extensive in sequence space. It has now been demonstrated that only 5–20% of a given protein’s amino acid sequence remains invariant during evolution [47] and that sequences with little if any measurable sequence identity can nonetheless fold into identical conformations [63, 66]. The same appears true for RNA: only seven nucleotides are strictly conserved among group I self-splicing introns, yet the secondary (and presumably the tertiary) structure of the ribozyme is preserved [42]. Because these disparate group I isolates have the same fold and function, it is thought that they descended from a common ancestor by taking distinct paths on the same neutral network.

In the last 20 years, well-established methods of nucleic acid and protein synthesis have permitted direct, quantitative evaluation of the DCH. Automated DNA synthesis can be used to construct arbitrary sequences or combinatorial pools of sequences that serve as templates for the genetically-encoded expression of RNA and protein molecules. Although this technology has been primarily used in the synthesis and analysis of evolved, biological sequences (or their mutated counterparts), it can also be adapted to the synthesis of random-sequence, unevolved RNA and proteins. For example, LaBean et al. [35–37], designed and synthesized DNA sequences that, when cloned, expressed unevolved proteins having sequence lengths and amino acid compositions matching small globular proteins found in nature. Surprisingly, they discovered ample evidence for solubility, specific secondary structure, and cooperative unfolding transitions, among these unevolved protein sequences. Davidson and Sauer [11, 12], Prijambada et al. [48], Chiarabelli et al. [8] and Doi et al. [14] have also expressed, purified, and analyzed the structures of unevolved, random-sequence proteins. Although these systems produce oligopeptides that are smaller than biological proteins and sometimes used a restrictive set of the 20 amino acids, they also obtained evidence for secondary structure. Schultes et al. [56] expressed and analyzed a set of 20 RNA molecules whose sequences had been randomly-generated in a computer. Using a battery of physical and chemical techniques for probing the folded conformations of these RNAs, they demonstrated sequence specific, magnesium-dependent folding to structures that were often as compact as evolved sequences having analogous size and nucleotide composition. Hence, for both proteins and RNAs, it appears that at least some elements of folded structure are common in sequence space and are independent of natural selection.

In addition to these structural studies, Schultes and Bartel [57] synthesized a series of RNA to verify the existence of RNA neutral networks for two catalytically active RNAs. This study was also able to demonstrate the close proximity of neutral networks to one another in sequence space (these results will be discussed in more detail in the next section).

Furthermore, by screening combinatorial pools of random-sequence proteins or RNAs for predefined function (i.e. in vitro selection), it has now been amply demonstrated that specific, biological-like binding and catalysis are at least as common in sequence space as one in 10^{10} to 10^{12} [70]. Moreover, since combinatorial pools have been shown to produce functional sequences for particular, arbitrary physiochemical tasks, these same pools, by implication, must contain functional sequences for *any* physiochemical task [10, 26, 27]. Hence, from a space of 10^{60} possible sequences (RNA oligonucleotides having 100 randomized positions; $4^{100} \approx 10^{60}$ possibilities), a typical combinatorial pool containing as few as 10^{14} sequences (only 1 part in 10^{46} of the possible), nevertheless appears to contain every function found in the whole of sequence space.

Taken together, these experimental results demonstrate that sequence space is, contrary to the DCH, densely populated with sequences able to fold into stable, well ordered and even native-like conformations. Furthermore, the diversity of structures and functions available in relatively small samples of sequence space points toward correlations and symmetries within the complex multi-dimensional encoding of structure that have yet to be fully appreciated for their roles in the evolution of new biochemical functions. These results indicate the need for a new theory of molecular structure that is independent of natural selection, evolution, or even biology. This theory would certainly accommodate the dynamics of selection and mutation in accounting for the diversity and disparity among functional sequences and their structures, but this theory would also make specific predictions about how the much larger space of possible sequences facilitates evolution.

Starting from Smith's conception of a protein space, and its analogue for RNA, we have been developing some of the conceptual approaches and the computational and experimental tools of such a theory. We review some of this work in the next section.

3 Theory and Experiments with Unevolved Sequences

Smith's example made clear the formal organization of related sequences into neighborhoods where nearest neighbors are related by single-step amino acid substitutions. Smith restricted his discussion to proteins of a single length, N, ignoring for the sake of clarity, deletion, and insertion mutations. He also assumed a fixed alphabet size ($A = 20$ amino acids). As each amino acid position of a given sequence can mutate to any one of the other 19 amino acids, it is obvious that each sequence is connected to $19N$ neighboring sequences by single-step substitutions. As this is true for all A^N sequences, protein sequence space is a $(19N)$-regular graph. In this representation, each node on the graph is a sequence and each single-step substitution is an edge. Generalizing to RNA which has only four monomeric building blocks ($A = 4$ nucleotides), RNA sequence space is a $(3N)$-regular graph. Table 1 summarizes some of the fundamental mathematical properties of sequence spaces using well-known combinatorial formulas. Figure 1 represents a trivial sequence space for binary strings of length $N = 4$, drawn from the monomers G and A.

To make the concept of protein sequence space more concrete, imagine you are gazing with your $(19N)$-dimensional eye at the regular graph of a protein sequence space. Since this is the totality of sequence possibilities, all of protein evolution must take place inside the finite boundaries of this graph. From this hyper-bird's-eye

Table 1 Fundamental properties of sequence space

Alphabet size	A, the cardinality of the set of monomers $\{a_1, a_2, a_3, \ldots, a_A\}$
Sequence length	N
Number of sequence possibilities	A^N
Number of nearest neighbors (dimensionality of the space)	$(A-1)N$
Number of neighbors k-steps away	$(A-1)^k_N C_k$
Number of composition classes	${}_{(N+(A-1))}C_N$
Number of sequences per composition class	$N!/(a_1! \cdot a_2! \cdot a_3! \cdot \cdots \cdot a_A!)$

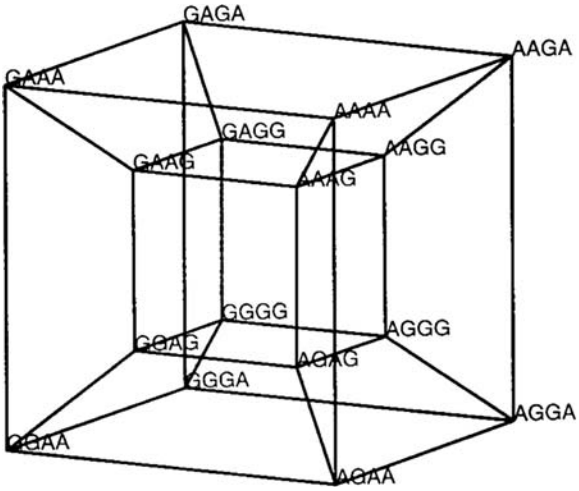


Fig. 1 A Boolean hyper-cube is the sequence space for binary sequences, $N = 4$, where each sequence is connected to its four single-step mutational neighbors

view of evolution, imagine color-coding sequences according to various structural or functional properties. First, color all the nodes that are capable of folding into one specific, compact, globular conformation such as a native myoglobin fold (this is, in the context of an aqueous, buffered saline solution at room temperature, conditions typically used in in vitro experiments). Although Salisbury would be interested to know the fraction of sequences that have lit up, we might go on to ask how these colored nodes are distributed across the graph. Are they isotropically dispersed or clustered? Are there multiple clusters? If so, are the different clusters interconnected, or are they isolated by regions of sequence space devoid of myoglobin folds? Using a different color, now mark all those myoglobin sequences that have been actualized in the course of evolution. Are they interconnected by a single $(19N)$-dimensional phylogenetic tree or do they belong to multiple, independent evolutionary lineages? Using a third color, light up the constellations of sequences capable of folding and functioning in the context of a fibrous protein such as collagen. Do the myoglobin and collagen distributions occupy distinct regions of sequence space, or are they interwoven? What happens to sequences "in between" the colors? Are they half-globular, half-fibrous, or something altogether different? Now imagine repeating these experiments while parametrically scanning the experimental conditions (e.g., changing temperature or salt concentration or pH). How sensitive are the two distributions of colors to these changes? Under what conditions do the distributions grow or contract or change shape? Are there certain critical values of conditions for which the distributions undergo abrupt changes in size and shape?

The purpose of this *gedankenexperiment* is to demonstrate the range and scope of fascinating and important questions that were not permissible under the DCH. Of course, in reality, we can never hope to directly view a high-dimensional space, but we can conceive of useful projections. Also, for even short polymers, it will never be possible to exhaustively analyze all possible sequences. However, although any particular sampling of sequence space will always be necessarily sparse, there are nonetheless, practical strategies for sampling sequence spaces that can yield meaningful data. As we will see, sequence spaces contain a number of profound symmetries and complex correlations that when understood will not only help to rationalize the evolution of RNA and protein molecules in nature, but will also serve as a powerful guide in laboratory-based searches for novel molecular structures and function.

Practical approaches to visualizing sequence space must involve a drastic reduction in dimensionality, from $19N$ or $3N$ down to just 2 or 3. Although there are a number of mathematical techniques one could use to perform this dimensional reduction, without proper constraints these projections would tend to confound visualization and obscure the patterns we are seeking. However, based on what we know from the theory of regular graphs and from the physiochemical properties of proteins and RNAs, a useful projection scheme immediately presents itself.

As can be seen in Table 1, all A^N sequence possibilities can be partitioned into a relatively small number of *composition* classes: sequences that have the same proportions of monomers. The number of sequences that belong to a composition class varies dramatically, and can be calculated using the multi-nominal function. The notion of composition class has particular relevance to real molecular sequences because each of the monomers are chemically and structurally distinct, and so sequences biased in one composition are expected to have structural properties that are different from sequences having other compositional biases. Extreme cases are the homopolymers, sequences composed of only a single type of monomer. These sequences are unique in that they contain no sequence information, and their physical properties reflect only the intrinsic propensities of the monomer itself. Depending on the monomer, the homopolymer may or may not be structured. For example, in the case of RNA, poly-guanine is expected to have a collapsed structure (primarily due to base stacking interactions) whereas poly-uridine is not. Indeed, poly-uridine is probably a good example of a true random-coil [56]. For proteins, the same could be said for poly-lysine at alkaline pH (ordered) and at neutral and acidic pHs (disordered). Homopolymers composed of the other amino acids would be more or less the same, but in any case, for both proteins and RNA, the homopolymer sequences act as internal references with information theoretic, physiochemical, and biological significance. Hence, homopolymers act as universal reference points for regular

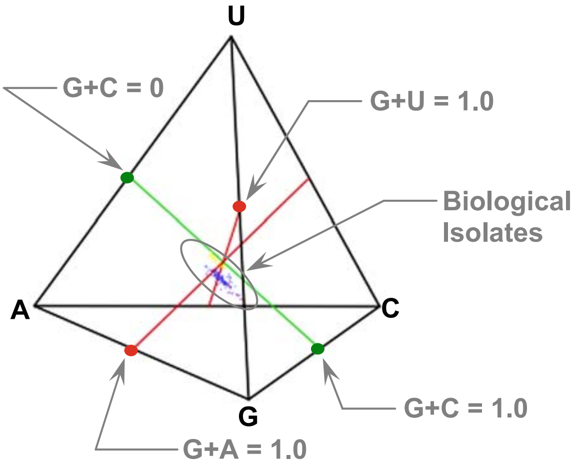


Fig. 2 The RNA simplex. Chargaff's Axis (*green line*) defines the familiar gradient in GC content. The simplex also makes explicit additional composition parameters: G + A and G + U (*red axes*). Isoheteropolymers are located at the intersection of the red and green axes. Plotted are 147 16S rRNA biological isolates (*oval*): Archaea, *red*; Bacteria, *blue*; Eucarya, *yellow* (from [60])

graph of sequence space. Using the homopolymers, we could even "triangulate" distances (in single-step substitutions) to any arbitrary sequence.

To be concrete, the sequence space for RNAs having exactly 85 nucleotides has 255 dimensions and over 10^{51} sequence possibilities. This enormous number sequences, however, is partitioned among only 109,736 distinct composition classes. Although sequence spaces have no absolute "center" or "inside/outside" sequences, the space of composition classes does. In the case of RNA, each of the composition classes can be geometrically arranged as points in the volume of a tetrahedron, where the four vertices are the composition classes of the four homopolymers, and the composition class at the center-of-gravity of the tetrahedron represents sequences having a uniform distribution of the four nucleotides (i.e., 25% each A, C, G, and U), Fig. 2. We refer to these maximum entropy sequences as the isoheteropolymers. Because all possible sequences are partitioned among the composition classes, the tetrahedron can be considered a 3-dimensional projection, or simplex, of the $(3N)$-dimensional regular graph of RNA sequence space.

Because the four nucleotide bases are sterically and chemically distinct, different ratios of the four bases impose an anisotropic distribution of chemical properties among RNA with differing compositions. The most important anisotropy, due to Watson–Crick base-pairing (A pairs with U, C pairs with G), is the symmetry defined by the set of compositions extending from the mid-point of the CG-edge through the center of the simplex, to the mid-point of the AU-edge (green line in Fig. 2). Referred to as Chargaff's Axis (after Chargaff's Rule for base composition in genomic DNA, where molar fractions of A = T & C = G [7]), it is the locus of composition classes formally permitting the maximum possible Watson–Crick base-pairing in RNA. In this way, the RNA simplex projection combines the formal properties of sequence space with well established biophysical properties of RNA.

Taking a lead from the thought experiment described above, we can imagine that evolving RNA sequences map trajectories in sequence space as they undergo modification by mutation and selection. Trajectories involving changes in monomer composition can be plotted and observed in the RNA simplex. This is done by simply extracting the frequency of nucleotide bases composing individual sequences, and then calculating their simplex coordinates as G + A, G + U and G + C contents. In our first analysis, we compiled 2800 distinct sequences representing 15 distinct functional classes of biological RNA [60]. Remarkably, we found that these diverse biological RNAs universally occupied a restricted volume of the simplex, forming narrow clouds that parallel Chargaff's Axis, yet are displaced (by ~5%) toward the AG-edge (i.e., toward the purines), Fig. 2. The diversity in sequence and structure among the 15 functional classes of RNAs examined, imply that these molecules share little, if any, evolutionary history and that their coincidence in this region of the simplex was a heretofore unknown example of adaptive convergence. This was observational evidence that a universal principle of macromolecular structure was being independently exploited by different genealogical lineages of RNA.

At the time, we were completing these first analyses, the first complete genomes of free-living organisms were being published, and the idea of calculating and plotting base composition was seen by some to be a step backward to days before efficient sequencing methods had been developed. To the contrary however, by casting RNA evolution first in the context of sequence space, and then in the context of composition space, our analysis immediately revealed profound patterns that even the most comprehensive cladistic methods had failed to detect. This is because cladistic methods seek patterns within evolutionary lineages and are thus unable to resolve trends that are due to universal constraints from trends that merely reflect genealogical descent. Of course, phylogenetic analyses and the approach taken with the RNA simplex are entirely complementary, and either method is diminished in its explanatory power absent the other.

The displacement of the biological distributions from Chargaff's Axis reflects the structural constraints inherent to the folding of the linear phosphodiester backbone of RNA polymers under the influence of basepairing. The acquisition of arbitrarily complex folds in single-stranded RNAs emerges from the alternating composition of double-helical stem structures separated by unpaired single-stranded joining structures. As the bases in the canonical stem structures must, by definition, fulfill Chargaff's Rule (and, therefore, must lie on Chargaff's Axis), it is primarily the composition of unpaired bases in the joining structures that dictate the magnitude and direction of the displacement away from Chargaff's Axis.

For the biological isolates, the *magnitude* of the displacement from Chargaff's Axis is dictated by the compromise between the thermodynamic stability of the folds (favoring stems) and need for complex macromolecular structures (favoring single-stranded joining regions). The *direction* of the displacement of biological isolates from Chargaff's Axis could be, a priori, in any direction, yet the unpaired residues are universally purine-biased. This observed "purine-loading" in nature may reflect the unique chemical properties of purines contributing to stable and specific folding in RNA [34]. For example, X-ray crystallographic analyses of native-fold RNAs have demonstrated the ubiquity of purine-associated structural primitives including the A-minor motif, a tertiary interaction whereby an unpaired adenosine residue docks in the minor groove of a helical stem elsewhere in the macromolecular fold [46]. It has been established that the A-minor motif plays an essential role in stabilizing overall three-dimensional folded structures and maintaining biological activity [13].

The patterns revealed by the RNA simplex, and the purine-loading observed in structural studies comprise circumstantial evidence for the existence of general principles governing the complex folds required for biological activity. To explain this coincidence of biological sequences in restricted volumes of the RNA simplex, it is necessary to analyze and compare the macromolecular folding of sequences that are distinct from biological isolates. For example, if purine-loading is a universal principle, then purine-depleted sequences would be less likely to acquire biologically relevant folds. Perhaps purine-depleted sequences tend to have lower thermodynamic stability or kinetic barriers that somehow prevent sufficiently complex or adaptable structures. Answering such questions require the synthesis and biophysical characterization of unevolved RNA sequences. Because in vitro methods are laborious even for well-behaved RNA structures [67], we began our probing of unevolved RNA sequences in silico. This required a fast RNA folding algorithm, and a strategy for meaningful, if sparse sampling.

Michael Zuker's program, *mfold*, employs a dynamic programming algorithm to compute the minimum free energy secondary structure of a specified RNA sequence [72, 73]. It also incorporates empirically derived thermodynamic parameters of base-pair interactions [43]. mfold is limited to secondary structure prediction (forgoing any attempt at three-dimensional structure prediction) which is to say that it is better at finding reverse-complementary sub-sequences than correct backbone topologies. Nonetheless, it is a robust and efficient algorithm for modeling sequence specific base-pair interactions, which contribute the bulk of the free energy of folding.

At first, the idea of sampling RNA sequence space seems futile. For even short RNAs of only 100 nucleotides have over 10^{60} sequence possibilities. Using mfold, it is a heroic task to fold 10^{8} sequences, yet this is only one-part in 10^{52}. How could we possibly produce a meaningful analysis of sequence space with such an exceedingly sparse sample? This conundrum can be resolved by employing methods of "perfect sampling." Rather than sampling sequence space by generating random sequences from a uniform distribution of the four nucleotide bases (repeating the sampling from within or near the isoheterpolymers), we instead generate random sequences that have specific base frequencies spanning uniformly and systematically, the entire volume of the RNA simplex. In a method, we call *Constrained Independent Random Sampling* (CIRS), a given composition class is repeatedly and independently sampled producing a cohort of randomly generated sequences that are unrelated, but have identical base compositions. CIRS essentially "shuffles" a sequence within a given composition class, creating a random permutation that is one of many possible molecular isomers.

In our simulations, 100 arbitrary sequences (each having 100 nucleotide bases) were sampled from 1771 compositions classes differing by 5% composition intervals throughout the simplex [58]. The resulting thermodynamic free energies computed from the folded RNAs were then averaged for each composition class and plotted in the simplex (Fig. 3).

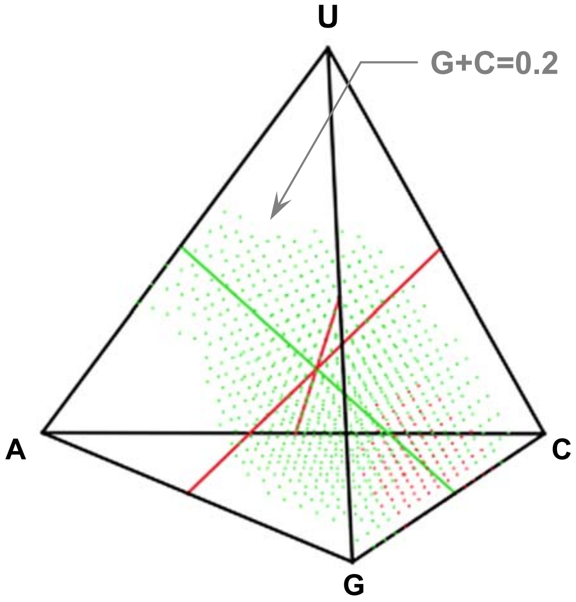


Fig. 3 Constrained Independent Random Sampling of the RNA simplex. Each composition class is colored according to the average computed thermodynamic free energy of folding for 100 arbitrary sequences. Due to the asymmetries of the CG and AU base-pair interactions, the most stable folds (*red*) tend to occur near the CG-edge of the simplex. Folds having intermediate stability (*green*) fan out along Chargaff's Axis toward the AU-edge. Only in CG-depleted sequences, are there ample opportunities for AU base-pair interactions. Blank space indicates composition classes sparse in Watson–Crick partners and, therefore, sparse in RNA polymers having stable, unique folds (from [60]

According to these computations, the most stable folds occur for sequences near the mid-point of the CG-edge (red points). Intermediate stabilities (green points) occur along Chargaff's Axis, where the potential for Waston–Crick base pairing is maximal. The "bottle neck" in this distribution near $G + C = 0.2$ reflects the frustration of stable G:C pair formation by the abundance of A and U residues (and A:U pairs). Blank space is occupied by RNAs lacking Watson–Crick partners and, therefore, stable secondary structure. This complex distribution of RNA folding with respect to composition, maps the spontaneous base-pairing propensity of RNA polymers throughout sequence space. As the RNA simplex makes clear, spontaneous macromolecular properties (spontaneous in the sense that they are independent of selection or rational design) can sometimes provide as much or even more ordered structure than those found in biology. This spontaneous ordering of macromolecular folds, is a purely physiochemical processes driven by the release of free energy of folding and has nothing to do with natural selection or biological evolution. Indeed, from this point of view, it is the preponderance of well-ordered macromolecular folds in sequence space that permits evolutionary adaptation [31]. In the context of sequence space, natural selection is seen merely as a culling mechanism, rather than as a creative force.

Rob Knight's group at the University of Colorado in Boulder used mfold and the RNA simplex to demonstrate that for unevolved sequences having the same base composition as biological isolates, the predicted structures had the same compositional preferences among their structural elements as observed among the biological isolates [61]. That is, like the biological RNAs, these unevolved sequences folded such that purines predominated the unpaired residues. Hence, purine-loading is not a consequence of natural selection, but is a manifestation of the self-ordering properties intrinsic to RNA polymers.

Curious as to how the complex distribution of folding in the RNA simplex might constrain or facilitate RNA evolution, we proposed a stochastic model for mapping evolutionary trajectories within the simplex. Making simple assumptions about selection (tending to maximize thermodynamic stability) and mutation (tending to randomize the sequence) we were able to evaluate the evolutionary "potential" for each composition class, creating the analogue of a "attractor-basin portrait" for RNA evolutionary dynamics in the simplex. Surprisingly, this simple model predicted the mean $G + A$ and $G + U$ values of 928 tRNAs and 382 5SrRNAs to within 3% [58, 59].

In a more sophisticated analysis of the distribution of specific RNA structural motifs in sequence space, Knight et al. [33] discovered (after folding several hundred million arbitrary RNAs on a computational grid) that the sequences capable of folding to the isoleucine aptamer and hammerhead ribozyme structures are most likely to be found in distinct regions of the simplex. This curious result also implied that the neutral networks for these two RNA structures probably has local variation in the degree of connectivity. Furthermore, this result suggests that random-sequence pools of RNAs used in vitro selection experiments could be optimized by compositionally biasing the pool synthesis, thereby focusing the sparse sampling of sequence space into the most promising composition classes.

mfold is explicitly a secondary structure prediction algorithm and ignores tertiary-level interactions completely, so although computational analyses have an important role to play in a survey of sequence space, they are inherently limited and must ultimately be supplemented with analogous empirical data. As mentioned previously, Schultes et al. [56] have implemented CIRS in vitro, synthesizing 10 arbitrary sequences (having 85 nucleotides) at two distinct composition classes: the isoheteropolymers (a useful reference point when considering base composition) and a composition corresponding to the genomic form of the Hepatitis Delta Virus (HDV) self-cleaving ribozyme. The HDV ribozyme and other model sequences were used as structural controls against which to compare the conformations of unevolved RNAs. The structures of these 20 unevolved RNA molecules were then probed using three independent methods: native gel electrophoresis; analytical ultracentrifugation; and lead(II) chemical probing. These experiments demonstrated that these unevolved RNAs had sequence-specific secondary structure configurations and compact magnesium-dependent conformational states comparable to those of evolved RNAs. But unlike evolved sequences, unevolved sequences were prone to having multiple competing conformations. So, by comparing structures of only two dozen RNAs, it was possible to begin teasing apart properties of RNA structure that are dependent on natural selection from those that are independent of natural selection: Evolution appears necessary to achieve uniquely folding sequences, but not to account for the well-ordered secondary structures and overall compactness commonly observed in nature.

In CIRS, each sequence in constitutes an independent random sample and is therefore unable to address the local network architecture of sequence space. To gain insight into the local neighborhood structure of sequence space we use a different method of sampling called *Local Network Sampling* (LNS). In this case, a particular sequence (called the anchor) is used as the basis for generating a sample set of mutant sequences. In this way, the sample gathers data about how structural correlations between sequences are distributed in sequence space. For instance, LNS may generate all the single- and double-step substitution mutations of the anchor sequence (either in silico, or in vitro using combinatorial pools or microarrays). Such a LNS would provide a direct measurement of the fraction of neighbors that are neutral (f as defined by Smith [62]). In principle, the anchor may be an evolved or unevolved sequence. Using LNS in this way, we could compare the degree of neutrality among evolved and unevolved sequences and/or sequences having different monomer compositions.

In a different application of LNS, a pair of sequences (an anchor and a target) is connected by a series of sequences that form a connected path through sequence space, linking the anchor and target by single- or double-step substitutions (double-step substitutions are particularly relevant in nucleic acids as they are basis of compensatory mutations in helical stems). The intervening sequences can be generated by random mutations or by mutations that preserve some aspect of structure (i.e., as neutral mutations, Grüner et al. [22, 23]), [17, 52]. For example, Schultes and Bartel [57] used rational design and methods of site-directed mutagenesis to experimentally implement a LNS that demonstrated the proximity of RNA neutral networks in sequence space.

In these LNS experiments, the anchor and target sequences were two different ribozymes: the class III ligase and the antigenomic form of the HDV self-cleaving ribozyme. The class III ligase is a synthetic ribozyme isolated from a pool of random RNA sequences. It joins an oligonucleotide substrate ($5'$-GAACCAGUC) to its $5'$ terminus using a $2'$–$5'$ linkage that is distinct from the typical $3'$–$5'$ linkage used in biological RNAs. The HDV self-cleaving ribozyme carries out the site-specific cleavage reactions needed during the life cycle of the virus (at the position indicated by the arrow in Fig. 5, near G10). The prototype class III and HDV ribozymes have no more than the 25% sequence identity expected by chance.

Using what was known about the structures of these two ribozymes, it was possible to rationally design a single sequence that simultaneously satisfied the base-pairing requirements of both the HDV and ligase ribozymes. Although this sequence design was initially done by hand, the procedure was later automated as a simple computer program (Graham Ruby, personal communication). Indeed, "on paper" a large number of such "intersecting" sequences can be conceived.

One such sequence (Fig. 4) was 42 mutational steps away from the ligase anchor (39 base substitutions, one point deletion, and two single-nucleotide insertions) and

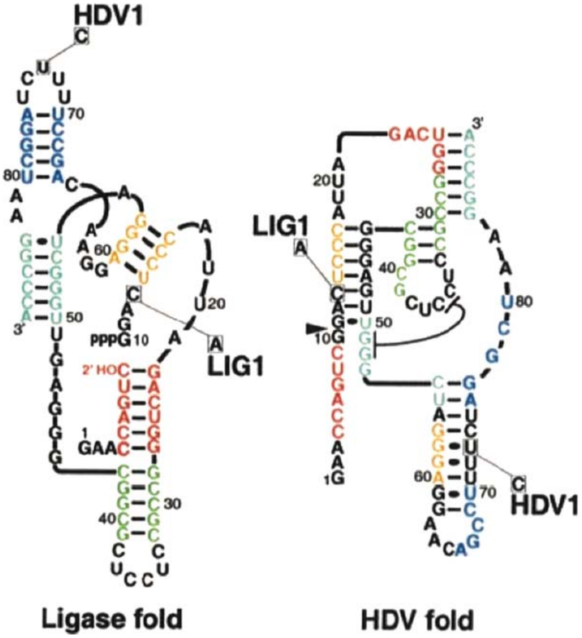


Fig. 4 The intersection sequence: a single RNA sequence accommodating the base-pairing configuration of two different ribozyme folds. Sequence position is numbered and color coded (with respect to the ligase secondary structure), demonstrating that there are no two base-pairs in common between the two folds. Two single-step substitutions are indicated (LIG1 & HDV1) that stabilize one fold over the other (from [57])

44 mutational steps from the HDV ribozyme target (40 substitutions, one deletion, and three insertions). When this sequence was synthesized in the two formats depicted in the figure, catalytic activity significantly above the uncatalyzed rates were detected for both self-ligation and site-specific self-cleavage (Fig. 5). It was shown that ligation occurred with the regiospecificity of the class III ligase (forming a 2′ linkage rather than the biological 3′ linkage), indicating that the class III ligase fold was achieved by some of the molecules. Cleavage also occurred as expected, with formation of cyclic phosphate. Although ligation and cleavage rates were lower than for the anchor and target sequences, this single sequence could assume two completely different, catalytically active folds.

In designing the intersection sequence, an unavoidable substitution from A to C (at position 13) was required, a position known to be critical to the optimal functioning of the ligase. Substituting C13 with A (creating the LIG1 construct, see Figs. 4 and 5) simultaneously restored ligase activity and introduced a G:A mismatch in the context of the HDV fold. The C13A point substitution dramatically increased the ligation rate (90 times) and lowered the cleavage rate below detection. On the other hand, the U73C substitution (creating the HDV1 construct), which is expected to stabilize the HDV fold, substantially increased site-specific cleavage (120 times), while lowering the ligation rate twofold. The substantial improvement seen with single-nucleotide substitutions suggested that the intersection sequence might be very close to the neutral networks of both ribozymes. With only two additional mutations (again one stabilizing the ligase fold, the other stabilizing the HDV fold), it was demonstrated that two ribozyme sequences (LIG2 and HDV 2), having totally

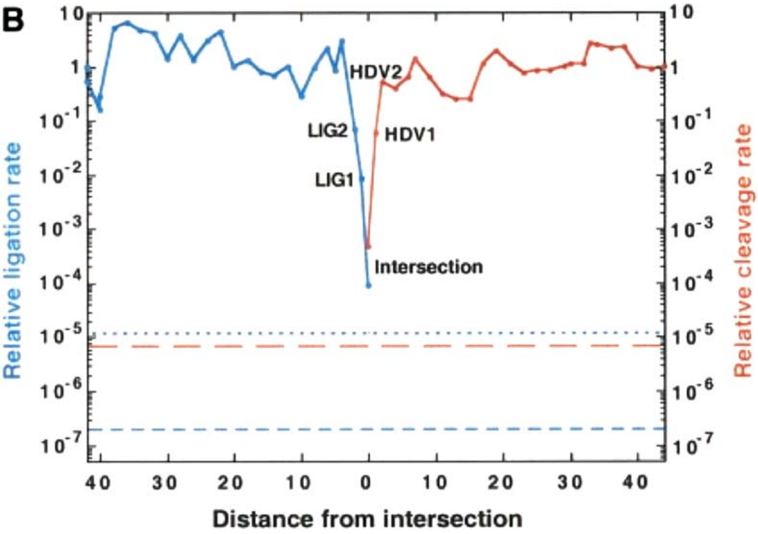


Fig. 5 The close apposition of two neutral networks in RNA sequence space. The *abscissa* indicates distance in single-step mutations from the intersection sequence (center) to the prototype ligase and HDV ribozyme sequences. The *curves* represent measured activities for ligation (*blue*) and cleavage (*red*) along the respective neutral networks. The intersection sequence demonstrated both ligation and cleavage activities (*horizontal dotted lines*). Note that LIG2 and HDV2, though separated by only 4 substitutions, are considered to be neutral for their respective functions (from [57])

different folds and near-prototype-level activities, were separated in sequence space by only four substitutions.

At this point, in order to confirm that both LIG2 and HDV2 are on the respective neutral networks of the anchor and target sequences, paths were designed in sequence space that link these minor variants of the intersection sequence to their prototype sequences. Each step along these paths changed no more than two residues, often as compensatory mutations. Very smooth paths could be designed (Fig. 5), gradually changing nearly half the ribozyme residues yet never falling from the prototype activities by more than sevenfold. The ease by which these paths were designed is consistent with the theoretical results suggesting that neutral networks are a common feature of RNA sequence space. Because the anchor and target ribozymes share no evolutionary history or structural features, it suggests that neutral networks for other pairs of ribozymes closely approach each other. Indeed, by virtue of the high-dimensionality of sequence space, it appears plausible that each ribozyme neutral network closely approaches all other ribozyme networks.

In a very different approach to LNS, Curtis and Bartel [10] employed combinatorial pools to gain insight into the proximity of different ribozyme folds in sequence space. Nucleic acid sequences having a known functionality (i.e., the anchor) can often be optimized by creating high-diversity, partially randomized, combinatorial libraries of sequence variants (often differing from the anchor sequence by 2–10%). These so-called doped pools, containing up to 10^{14} molecules, are then screened for sequences having higher levels of activity than the anchor. In this case, however, Curtis and Bartel [10] used doped pool selections not to optimize the existing functionality, but as the basis for selecting an entirely new activity unrelated to that of the anchor.

First, a previously described, 90 nucleotide, self-aminoacylating ribozyme (anchor) was partially randomized (each of 65 bases were permitted to vary to one of the three other bases with a probability of 11%). Using methods to separate and amplify any sequence that could covalently link a phosphate group to itself, 23 distinct classes of self-kinasing ribozymes were eventually isolated. The kinase ribozyme activities were radically different than that of the parent (including the use of a trigonal bipyramidal transition state in contrast to the tetrahedral transition state of the parental self-aminoacylating ribozyme). Furthermore, the folds of each of the kinase ribozymes had little, if any correspondence with the fold of the parent. Hence, it was clear that many new folds and new activities can be found in close proximity of sequence space.

However, it was also found that the new kinase ribozymes were on average significantly farther from the self-aminoacylating ribozyme in sequence space (14 mutational steps) than was expected considering the statistical distribution of sequence variants in the initial doped pool (averaging only 7.5 mutational steps). Evidently, the more closely related sequences in the local mutational neighborhood of the anchor frustrate the formation of the alternative folds necessary for the alternative kinase function. Similar results have been obtained in analogous experiments using doped pools of nucleic acid aptamers [10, 26, 27], suggesting this is a general feature of the distribution of structure and function in sequence space. Although a close apposition of neutral networks (only 4 mutational steps) was demonstrated in the LNS experiment using rational design of individual mutants discussed above, the doped pool LNS experiments suggest that this may be relatively rare (though easily designed when the competing structures are understood). Although no conventional imagery can adequately capture the complexity of these high-dimensional neutral networks, it would appear that in some sense, RNA neutral networks are less like footpaths and more like expressways, where many different lanes of traffic are juxtaposed, yet remain distinct.

Prospects for a Protein Simplex and Exploring Protein Neutral Networks

Using the RNA simplex and very limited sampling (in vitro and in vivo), it is surprising how much we could learn about RNA sequence space. The same could be done for protein sequence space, although in the case of proteins there are 20 amino acids rather than 4 nucleotides, complicating the analysis (the composition of space for proteins is a 19-dimensional analogue of a 3-dimensional tetrahedron). Furthermore, the structure of the protein polymer backbone and the kinds of interactions that drive protein folding are different from RNA and necessitate the use of different algorithms, projections, and exploitation of different symmetries in order to create maps that emphasize the most relevant features of protein folding and function. Hence, protein sequence spaces will require the development of alternative methods of representation. As with RNA, such approaches promise insights into deep problems of protein folding, structure, function, and evolution not otherwise resolvable.

As just one example, from analyses of the native folds of protein sequences found in nature, it has been estimated that the total number of protein folds may be as few as 3000 [41]. This remarkably small number of folds found in nature immediately raises a fundamental question that cannot be answered when referring to genomic data sets alone. Are these 3000 folds the limit of what sequence space has to offer or is this limited number a consequence of historically constrained genealogical lineages? As is the case with RNA, current theories of protein structural biology and evolution make no predictions about what lies beyond biological sequences or the 3000 well-defined folds identified among those sequences. Only by systematically sampling protein sequence space in silico and ultimately in vitro will it be possible to reach a definitive explanation for the observed redundancy of protein folds found in nature.

Not only does this silence about unevolved sequences and structures create an explanatory gap in understanding the distributions of biological proteins, but it also fails to account for the spectrum of complex protein behavior found in nature and the laboratory. For example, it has recently been recognized that entire proteins, or large segments of proteins lack well-structured, three dimensional folds and that the amino acid sequence of these disordered regions can be highly conserved. Some of these disordered segments have been shown to have specific function, such as folding upon binding to specific targets and providing linkers in macromolecular arrays [16]. Without a theory of protein structure extending beyond evolved sequences, it has not been possible to understand the evolutionary significance or the structural biology of these so-called intrinsically unstructured proteins. On the one hand, their disorder may be taken as models of the structures of unevolved sequences. On the other hand, their evolutionarily conserved sequences (and characteristic amino acid composition, [65]) suggests that this "disordered" state is actually evolutionarily derived and, therefore, just as "evolved" as the structures of highly ordered globular or fibrous proteins. Only by learning more about the folds of unevolved protein sequences will it be possible to formulate a coherent understanding of the entire spectrum of protein structure, from disordered homopolymers [53] and intrinsically unstructured proteins to meso-ordered molten globules [49] to insoluble "over-structured" protein aggregates such prions [69].

No Molecule Is an Island

This survey of recent theoretical and experimental findings leads to a very different conception of folding and evolution than had been assumed from the time of the earliest biochemical investigations of protein and RNA. Under the delicate clockwork hypothesis, functional molecules were thought to be rare, isolated islands of

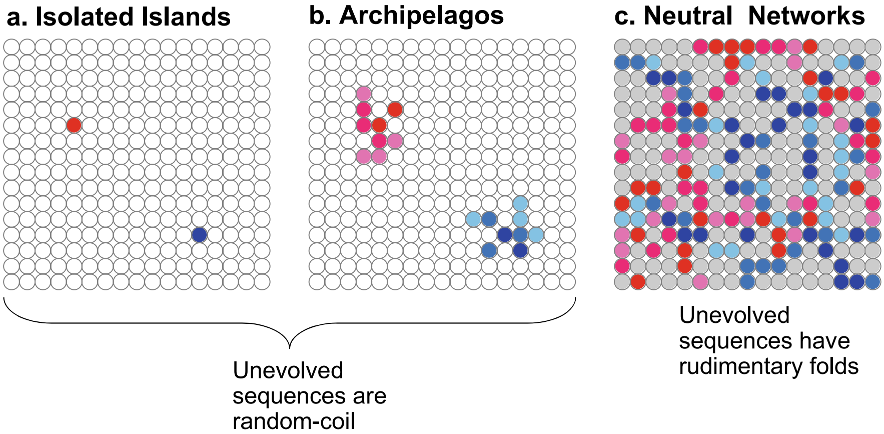

a. Isolated Islands
b. Archipelagos
c. Neutral Networks
Unevolved sequences are random-coil
Unevolved sequences have rudimentary folds

Fig. 6 Three theories of structure and function in sequence space. *Circles* represent different sequences and neighboring circles represent neighboring sequences. *Red colors* and *blue colors* represent two distinct molecular structures/functions. Different *shades of red and blue* represent slight variations in structure or function that are considered neutral. *White circles* indicate sequences without defined structures (e.g., random-coils). *Grey circles* indicate sequences with multiple competing conformations (rudimentary folds)

ordered structure in a sea of disordered random-coils (Fig. 6a). Although accumulating theory and data began to conflict with the DCH, suggesting that the distribution of function in sequence space was more like archipelagos (where isthmuses connected a small chain of islands, Fig. 6b), it remained axiomatic that the surrounding seas of unevolved sequences were disordered and irrelevant to understanding molecular structure and evolution. Since the 1990s, theoretical and experimental results suggest that RNA and protein sequence space is permeated by vast networks of sequences having neutral folds and functions. These networks span the entire sequence space, just as mountain ridges may span entire continents (Fig. 6c). As each fold is thought to correspond to a unique neutral network, these networks must be interwoven, densely "packed" in sequence space like a ball of string made of many different threads. Although it remains an open question as to whether arbitrary, unevolved sequences belong to extensive neutral networks, our experimental analyses of the structures of unevolved RNA and protein sequences suggest that a polymorphic *rudimentary* folding is prevalent among unevolved sequences throughout sequence space. We envision this rudimentary folding, for both RNA and protein, not as the absence of order, but rather as the superposition of multiple "ordered" structures, not unlike induced molten-globule states of native proteins. Indeed, the ubiquity of the molten-globule state (usually seen as a "broken" native fold), is from this point of view evidence of the intrinsic capacity of sequence space to generate ordered structure. In Fig. 6c, the rudimentary state depicted as grey circles, in contrast to the random-coil states (white circles, Figs. 6a and 6b) postulated in the DCH.

The hypothesis that unevolved sequences typically have rudimentary folds provides the coherent framework for structural biology and molecular evolution that has been missing since the 1950s. Not only does rudimentary folding provide a context in which to understand the many non-native states that have been identified in RNA and proteins, but it also begins to rationalize the origins of functional folds and how one fold could evolutionarily transform into another. Rudimentary folding implies that evolution proceeds not by building up a unique folding pathway from sequences that fold poorly, but by the elimination of competing meta-stable structures from sequences that are already significantly folded. Molecular evolution is therefore not a laborious search for rare sequences, but a problem in negative design, whereby numerous competing folds are selected against. Hence, rather than seeing the vast space of unevolved sequences as a near-perfect vacuum of biological function, the ubiquitous rudimentary fold suggests that sequence space is a nursery of nascent functionality, incubating evolutionary diversification and transformation. Due to the high-dimensionality of sequence space, any unevolved sequence having a rudimentary fold is probably close to a native fold, if not many native folds (as in Fig. 6c, where a grey circle may be near both red and blue circles). This would explain the ease by which molten-globule states can be induced from native folds by mutation or alteration in solution conditions.

Extensive neutral networks of native folding proteins or RNAs explain the diversification of sequences seen in nature, but neutral networks and rudimentary folding also explain the origin of entirely new folds. Different neutral networks that have close apposition in sequence space allow one structure to transform into another with only a small number of mutations. Rudimentary folds may facilitate this transition, acting as intersection sequences, simultaneously maintaining functional capacity for both folds until suitable mutations can complete the transition to the new neutral network. Indeed, rudimentary folding may be indistinguishable from sequences that lie at the intersections of (or between the close apposition of) neutral networks. From this point of view, the molten-globule states of perturbed biological native folds may be seen as intersection sequences bridging two (or more) neutral networks. It would be interesting to see if known molten-globules can be stabilized into alternative yet native-like structures. If so, then it is most accurate to conceive of arbitrary unevolved sequences not as evolutionary dead ends, but as a rich matrix of superimposed structure that bridge the multidimensional space between numerous neutral networks of native folds. In a sequence space supporting ubiquitous rudimentary folding, no molecule need ever be an island unto itself.

Rather than being irrelevant to our understanding of macromolecular structure and evolution, unevolved sequences are the crucial missing link between the large amount of sequence and structural data and a predictive theory of molecular evolution and its intelligent biotechnological application. Learning what we have about navigating and sampling sequence space, and seeing where this could lead, we would like to propose a framework for a more unified and concerted research effort to map protein and RNA sequence space. In analogy to genomics (a comprehensive sampling of biological sequences), we refer to this research program as sequenomics, a comprehensive sampling of sequence space.

4 Sequenomics: Mapping the Universe of Sequence Possibilities

Sequenomics is the study of sequence space. Although this includes all nucleic acid and protein sequences found in nature, the primary concern of sequenomics is the much larger space of unevolved sequences. The goal of sequenomics is to systematically and comprehensively sample nucleic acid and protein sequence spaces, in silico and in vitro, producing "maps" and "atlases" that define biological distributions, intrinsic symmetries, and physiochemical "territories" in the context of the totality of sequence possibilities. Such maps will reveal previously hidden, universal rules governing molecular folding and evolution, stimulate new approaches in the development of structure prediction algorithms, and suggest novel experiments using individual sequences, pools and microarray technology to inform the search for functional RNAs and proteins.

Before describing our vision of sequenomics, it is helpful to describe what sequenomics is *not*. Sequenomics is not cluster analysis or cladistic analysis of functional sequences. Although many theoretical [2, 19, 32], bioinformatic [38, 44], and experimental [5, 6] research efforts are otherwise complementary to the goals of sequenomics, it is the referral to informative, random samples of sequence space that most distinguishes this new approach.

The *core questions* of sequenomics, for both RNA and protein are: How do the folded conformations of unevolved sequences compare to known biological structures? What is the spectrum of ordered and disordered folding states? How common is native folding? How many different folds are there? How does the distribution of physiochemical properties and the connectivity of neutral networks vary across sequence space? How do these distributions limit or enable evolution? How can we use this knowledge to advance *de novo* design and in vitro selection? How can we best search sequence space? The *core technical challenge* of sequenomics is the development of sampling methods that are efficient and informative despite being necessarily sparse (experimentally, we need to sample on the order of a googol sequences using only thousands of sequences).

Answering these core issues will require a concerted approach of theoretical analyses and laboratory experiments, involving the development of new computational tools and experimental technologies with high-throughput capabilities. Based on our own research experience investigating unevolved RNAs and proteins, we have identified four distinct but complementary activities within sequenomics: (1) establishing a theoretical framework; (2) creating interactive visualization tools as low-dimensional windows into high-dimensional sequence spaces; (3) the assembly of a unique database of sequence and structural information combining evolved and unevolved RNAs and proteins; (4) laboratory experiments dedicated to the synthesis and structural characterization of unevolved, random-sequence RNAs and proteins.

Theoretical Sequenomics

The theoretical foundation of sequenomics is the regular graph structure of sequence space. As such, combinatorial and information theoretic tools can be immediately brought to bear in order to reveal numerous symmetries intrinsic to sequence space and correlations that emerge from the physiochemical constraints of macromolecular folding.

These theoretical descriptions will guide the formulation of sampling strategies using computer simulations, and ultimately laboratory experiments, to characterize the folding and structures of unevolved sequences. As discussed above, there are two broad classes of sampling methods in sequenomics: *Constrained Independent Random Samples* (generating sets of sequences randomly and independently under constraints, such as a specified monomer composition) and *Local Networked Samples* (generating sets of sequences that are interconnected as single-step mutational variants of an a priori specified sequence). Although LNS samples can sometimes probe local mutational neighborhoods exhaustively (i.e., all single- or double-step mutations), this method can also be used to design networks of mutational variants that extend as "transects" across the "diameter" of sequence space. Hence, despite the enormous size of sequence space, meaningful random samples of RNA and protein sequence space can be practically constructed and evaluated. As laboratory experiments tend to be more costly than computer simulations, we suspect that maps of sequence space based on samples using computer simulations will precede, and will therefore guide, subsequent experimental efforts.

Along these lines, it will be useful to develop software tools integrating various nucleic acid and protein folding/structure prediction algorithms, with the goal to evaluate and compare the performance of existing algorithms on known sequences and on samples of sequence space. Computational sampling may in some instances require the folding of hundreds of millions of sequences in order to elucidate distinct regions of sequence space and the nature of their boundaries. Some or all of this computed data may be archived into the Sequenomics Database described below. Like any map, the scale or resolution in mapping sequence space will be chosen to present certain features of the "territory" over others. Thus, fast algorithms can be used for constructing small-scale, low-resolution maps, and more demanding algorithms for detailed analysis (e.g., neutral network analyses).

Although no folding algorithm is perfect, we can still develop useful, approximate maps of sequence space using computer simulations. By comparing the structures of evolved and unevolved sequences head-to-head, we can begin to make inferences about background sequences and how they are modified in the course of evolution, even if algorithms can make only low-resolution predictions or have some error in predicting correct folds. This is because any limitation that is idiosyncratic to a particular algorithm will be held constant over individual sequences of the sample and can therefore be handled as systematic error. In addition to the final output of an algorithm, the behavior of the algorithm may also tell us much about the space. For example, for sequences that are poised at the junction of two neutral networks, folding algorithms may demonstrate anomalous run times as the sequence is undecided about its ultimate structure. Furthermore, the computational sampling of sequence space using different algorithms is also an opportunity to perform comparisons of various algorithms against each other, and evaluate their performance over large datasets that are not confounded by genealogical history or other factors that may be peculiar to biological samples. Such head-to-head comparisons will also indicate where factors like extreme monomer composition and sequence information complexity may expose the intrinsic limitations of a given algorithm.

Sequenomics Visualization

In our work with the RNA simplex described above, we discovered that visual representation was much more than a convenient communication tool. It was essential to conceptualizing RNA sequence space and understanding how the high-dimensionality of sequence information impacts the evolution of RNA molecules. Yet the RNA simplex is only one of many conceivable projections, and additional methods for RNA and novel methods for protein sequence spaces are needed. These visualization tools need to be interactive, taking advantage of space, motion, and parametric control of the projections themselves, in order to identify and communicate inherently complex, and sometimes high-dimensional patterns. The development of visualization software would interlock with the theoretical efforts (described above) and the database efforts (described below), resulting in interactive, low-dimensional computational windows to the high-dimensional regular-graphs of sequence space.

The Sequenomics Database

To create useful maps of sequence space, we will need a centralized system of cataloging molecular sequence and structural data that is much more general than any existing bioinformatics database. In particular, this central data source would need to archive data from both RNA and proteins. But even beyond the juxtaposition of nucleic acids and protein sequences, such a database would be unique in archiving, in a consistent format, structural information for four broad classes of sequences. These sequence classes are necessary as standards and controls when evaluating the structures of unevolved sequences.

(1) *Reference Sequences* (*negative controls*): These are homopolymers, 4 for RNA, 20 for proteins. These 24 sequences contain no information content and so their macromolecular properties reflect the physiochemical properties intrinsic to each of the nucleotide or amino acid monomers. Some homopolymers may be impossible to synthesize and/or experimentally analyze (see experimental section below), and may confound folding algorithms. Nonetheless, homopolymer sequences will still provide essential reference points if only in mapping zones forbidden to biological evolution.

(2) *Model Sequences* (*positive controls*): These are well behaved, artificial sequence designs that fold into simple, predictable structures, e.g., hairpins in RNA and the four-helix-bundle in proteins.
(3) *Evolved Sequences*: These are sequences isolated from nature or in vitro evolution experiments. For the purposes of sequenomics, only the highest-quality, representative data for the known ranges of molecular structures, functions, source organisms, and ecological settings will be included in this database. Although datasets comprehensive for particular functional classes (e.g., 16 S rRNA used to build universal phylogenetic trees) are important when asking questions about neutral networks, it is the inclusion of sequences from a wide range of functional classes and evolutionary lineages will make this compilation unique among bioinformatics databases, and uniquely positioned to answer the core questions of sequenomics.
(4) *Unevolved Sequences*: As samples of sequence space, the unevolved sequences are the unknowns that sequenomics wishes to understand. At first, it seems absurd to keep randomly generated sequences in a database, but the arbitrary sequences we generate and then analyze (by computational folding or by lab experiments) become valuable data that deserve to be archived for the purposes of ongoing analyses. Depending on the application, random sequences could be stored as records similar to the biological sequence data or as a seed for pseudo-random number generation.

Experimental Sequenomics

Ultimately, protein and RNA sequence space will need to be probed experimentally, using real molecules in the laboratory. As the actual synthesis, preparation and analysis of protein and RNA sequences is much more costly than its virtual analogue, the issues surrounding efficient sampling methodologies become even more important. Experimentalists, however, will be able to consult the theoretical maps of protein and RNA sequence space (and even download specific unevolved sequences from the sequenomics database) to find sets of sequences permitting specific hypotheses to be tested. There are three experimental methods by which arbitrary, unevolved sequences can be synthesized and their structures analyzed.

(1) *The synthesis of specific arbitrary sequences*: Automated DNA synthesis has made the construction of arbitrary DNA templates routine. From these templates, the synthesis of specified RNAs and proteins sequences can be implemented. Using commercial sources of DNA synthesis, it is possible for a single lab to make and process hundreds of oligonucleotide templates per year. Using high-throughput technologies, it is not inconceivable that tens of thousands of oligonucleotide and protein sequences could ultimately be made and perhaps even characterized per year. These techniques would be used to synthesize unevolved sequences, including the rational construction of putative neutral networks.
(2) *The synthesis of combinatorial pools*: Automated DNA synthesis can also be adapted to the synthesis of diverse pools of DNA templates. In this case, residue

positions along the length of the template are permitted to incorporate a mixture of the four nucleotides, simultaneously creating many, random-sequence DNA templates. Routine synthesis and preparative techniques can yield individual pools with 10^{15} unique sequences. Such pools have been used as the starting point for the selection of a wide variety of functional biopolymers. However, from the point of view of sequenomics, it is the sequences from these pools that are *not* selected that are of interest. Combinatorial pools can serve as a means to economically isolate large numbers of arbitrary sequences for direct analysis (although, unlike the synthesis of specific arbitrary sequences noted above, the monomer composition will fluctuate around the composition of the nucleotide mixture during pool synthesis). Combinatorial pools having biased compositions, although currently possible, have yet to be explored extensively as a means of focusing search in sequence space. Mathematical and experimental exploration of the role compositional focusing might play in optimizing combinatorial searches is an important goal of sequenomics. In contrast to this "shotgun" sampling of sequence space (where each molecule has uncorrelated sequence), these combinatorial methods can also be used to sample the local neighborhood of particular sequences. This "neighborhood" sampling method generates a pool of sequence variants that differ from the original sequence (on average) by some specified number of mutations (typically in the range of 2 to 10% of N). In this case, the 10^{15} molecules in the pool are highly correlated. Entire pools or cloned isolates from these pools, can be assayed for structure.

(3) *Fabrication of micoarrays*: Microarray technology for synthesis and display of protein and nucleic acid sequences has found wide application in genomic studies. Microarrays permit the quantitative analysis of large numbers of sequences (for binding or other biochemical tasks) in parallel using automated "chip" readers. As a cross between the synthesis of specific arbitrary sequences and the synthesis of combinatorial pools, microarrays will find extensive and novel uses in the sequenomics, especially in the exhaustive search of local mutational neighborhoods for neutral networks or intersecting neutral networks.

The characterization of folding and structure in unevolved sequences will be performed using any of various, standard techniques to be determined as a compromise between resolution and throughput. However, particularly useful techniques may include Nuclear Magnetic Resonance Spectroscopy (a technique that can yield precise measurements of the overall order-disorder in a protein or RNA) and Temperature Gradient Gel Electrophoresis (a technique that permits expedient measurement of thermodynamic and kinetic properties, e.g., [24, 25]. As noted above in the outline of the sequenomics database, positive structural controls will include sequences having trivial but well-behaved structures or sequences that code for complex structures that have been previously well characterized. Negative structural controls, or reference molecules, include sequences such as homopolymers, having no information content and in some cases no unique structure.

The dedication of expensive resources to the study of unevolved protein and RNA sequences seems, at first, anathema to both the spirit and practice of structural biology. Research proposals compete and funding is justified through the promise of application in biotechnology and medicine. How could unevolved sequences, having by definition, no known function, find their way to the head of long queues of excellent proposals struggling for scarce resources? We believe that once a solid theory of sequence space is developed, visualization tools are in place, and the sequenomics database is constructed, a large number of well-defined, tractable experiments will present themselves, and the space of unevolved sequences will become the obvious frontier of post-genomic and post-structural proteomic research.

References

1. Anfinsen CB (1973) Principles that govern the folding of protein chains. Science 181:223–230
2. Armstrong KA, Tidor B (2008) Computationally mapping sequence space to understand evolutionary protein engineering. Biotechnol Prog 24:62–73
3. Beadle GW, Tatum EL (1941) Genetic control of biochemical reactions in *neurospora*. Proc Natl Acad Sci 27:499–506
4. Bloomfield VA, Crothers DM, Tinoco I (2000) Nucleic acids: structures, properties, and functions. University Science Books, Sausalito
5. Breaker RR (2004) Natural and engineered nucleic acids as tool to explore biology. Nature 432:838–844
6. Carothers JM, Oestreich SC, Davis JH, Szostak JW (2004) Information complexity and functional activity of RNA structure. J Am Chem Soc 126:5130–5137
7. Chargaff E (1950) Chemical specificity of nucleic acids and mechanism of their enzymatic degradation. Experientia 6:201–209
8. Chiarabellia C et al (2001) Investigation of *de novo* totally random biosequences, part II: on the folding frequency in a totally random library of de novo proteins obtained by phage display. Chem Biodivers 3:840–859
9. Creighton TE (1993) Proteins: structures and molecular properties, 2nd edn. Freeman, New York, pp 172–173
10. Curtis EA, Bartel DP (2005) New catalytic structures from an existing ribozyme. Nat Struct Mol Biol 12:994–1000
11. Davidson AR, Sauer RT (1994) Folded proteins occur frequently in libraries of random amino acid sequences. Proc Natl Acad Sci 91:2146–2150
12. Davidson AR, Lumb KJ, Sauer RT (1995) Cooperatively folded proteins in random sequence libraries. Nat Struct Biol 2:856–864
13. Doherty EA et al (2001) A universal mode of helix packing in RNA. Nat Struct Biol 8:339–343
14. Doi N, Kakukawa K, Oishi Y, Yanagawa H (2004) High solubility of random-sequence proteins consisting of five kinds of primitive amino acids. Prot Eng Des Sel 18:279–284
15. Draper DE (1992) The RNA-folding problem. Acc Chem Res 25:201–207
16. Dyson HJ, Wright PE (2005) Intrinsically unstructured proteins and their functions. Nat Rev Mol Biol 6:197–207
17. Fontana W, Schuster P (1998) Continuity in evolution: on the nature of transitions. Science 280:1451–1455
18. Frauenfelder H, Wolynes PG (1994) Biomolecules: where the physics of complexity and simplicity meet. Phys Today 47:58–64
19. Gan HH, Pasquali S, Schlick T (2003) Exploring the repertoire of RNA secondary motifs using graph theory; implications for RNA design. Nucleic Acids Res 31:2926–2943
20. Green DW, Ingram VM, Perutz MF (1953) The structure of hemoglobin, IV: sign determination by isomorphus replacement method. Proc R Soc Lond A 255:287–307

21. Gould SJ, Lewontin RC (1979) The spandrels of San Marco and the Panglossian paradigm: a critique of the adaptationalist programme. Proc R Soc Lond B 205:581–598
22. Grüner W et al (1996a) Analysis of RNA sequence structure maps by exhaustive enumeration, I: Neutral networks. Mon Chem 127:355–374
23. Grüner W et al (1996b) Analysis of RNA sequence structure maps by exhaustive enumeration, II: Structures of neutral networks and shape space covering. Mon Chem 127:375–389
24. Guo F, Cech TR (2002) Evolution of tetrahymena ribozyme mutants with increased structural stability. Nat Struct Biol 9:855–861
25. Hecker R et al (1988) Analysis of RNA structure by temperature-gradient gel electrophoresis:viroid replication and processing. Gene 72:59–74
26. Held DM et al (2003) Evolutionary landscapes for the acquisition of new ligand recognition by RNA aptamers. J Mol Evol 57:299–308
27. Huang Z, Szostak JW (2003) Evolution of aptamers with a new specificity and new secondary structure from ATP aptamers. RNA 9:1456–1463
28. Kendrew JC, Bode G, Dintzis HM, Parrish RC, Wykoff H (1958) A three-dimensional model of the myoglobin molecule obtained by X-ray analysis. Nature 181:660–662
29. Kimura M (1968) Evolutionary rate at the molecular level. Nature 217:624–626
30. King JL, Jukes TH (1969) Non-Darwinian evolution. Science 164:788–798
31. Kauffman SA (1993) The origins of order: self-organization and selection in evolution. Oxford University Press, New York
32. Kim N, Shin JS, Elmetwaly S, Gan HH, Schlick T (2007) RAGPOOLS: RNA-as-graph-pools a web server for assisting the design of structured RNA pools for in vitro selection. Bioinformatics. doi:10.1093/bioinformatics/btm439
33. Knight R et al (2005) Abundance of correctly folded RNA motifs in sequence space, calculated on computational grids. Nucleic Acids Res 33:6671–6671
34. Lambros RJ, Mortimer JR, Forsdyke DR (2003) Optimum growth temperature and the base composition of open reading frames in prokaryotes. Extremophiles 7:443–450
35. LaBean TH, Kayffman SA (1993) Design of synthetic gene libraries encoding random sequence proteins with desired ensemble characteristics. Protein Sci 2:1249–1254
36. LaBean TH, Kauffman SA, Butt TR (1995) Libraries of random-sequence polypeptides produced with high yield as carboxy-terminal fusions with ubiquitin. Mol Divers 1:29–38
37. LaBean TH, Schultes EA, Butt TR, Kauffman SA (2009) Protein folding absent selection (submitted)
38. Leontis N et al (2006) The RNA ontology consortium: an open invitation to the RNA community. RNA 12:533–541
39. Levinthal C (1968) Are there pathways for protein folding? Extrait J Chim Phys 65:44–45
40. Levinthal C (1969) How to fold graciously. In: DeBrunner JTP, Munck E (eds) Mossbauer spectroscopy in biological systems: proceedings of a meeting held at Allerton House, Monticello, IL. University of Illinois Press, Champaign, pp 22–24
41. Liu X, Fan K, Wang W (2004) The number of protein folds and their distribution over families in nature. Proteins 54:491–499
42. Lisacek F, Diaz Y, Michel F (1994) Automatic identification of group I introns cores in genomic DNA sequences. J Mol Biol 235:1206–1217
43. Mathews DH, Sabina J, Zuker M, Turner DH (1999) Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. J Mol Biol 288:911–940
44. Meier S, Özbek S (2007) A biological cosmos of parallel universes: does protein structural plasticity facilitate evolution? BioEssays 29:1095–1104
45. Mirsky AE, Pauling L (1936) On the structure of native, denatured, and coagulated proteins. Proc Natl Acad Sci 22:439–447
46. Nissen P et al (2001) RNA tertiary interactions in the large ribosomal subunit: the A-minor motif. Proc Natl Acad Sci 98:4899–4903
47. Pinker RJ, Lin L, Rose GD, Kallenbach NR (1993) Effects of alanine substitutions in alpha-helices of sperm whale myoglobin on protein stability. Protein Sci 2:1099–1105

48. Prijambada ID et al (1996) Solubility of artificial proteins with random sequences. FEBS Lett 382:21–25
49. Ptitsyn OB (1995) Molten globule and protein folding. Adv Protein Chem 47:83–229
50. Quastler H (1964) The emergence of biological organization. Yale University Press, New Haven
51. RajBhandary UL, Kohrer C (2006) Early days of tRNA research: discovery, function, purification and sequence analysis. J Biosci 31:439–451
52. Reidys CM, Stadler PF, Schuster P (1997) Generic properties of combinatory maps: neural networks of RNA secondary structures. Bull Math Biol 59:339–397
53. Rucker AL, Creamer TP (2002) Polyproline II helical structure in protein unfolded states: lysine peptides revisited. Protein Sci 11:980–985
54. Salisbury FB (1969) Natural selection and the complexity of the gene. Nature 224:342–343
55. Sanger F (1952) The arrangement of amino acids in proteins. Adv Protein Chem 7:1–69
56. Schultes EA, Spasic A, Mohanty U, Bartel DP (2005) Compact and ordered collapse in randomly generated RNA sequences. Nat Struct Mol Biol 12:1130–1136
57. Schultes EA, Bartel DP (2000) One sequence, two ribozymes: implications for the emergence of new ribozyme folds. Science 289:448–452
58. Schultes E, Hraber PT, LaBean TH (1999a) A parameterization of RNA sequence space. Complexity 4:61–71
59. Schultes EA, Hraber PT, LaBean TH (1999b) Estimating the contributions of selection and self-organization in RNA secondary structures. J Mol Evol 49:76–83
60. Schultes E, Hraber PT, LaBean TH (1997) Global similarities in nucleotide base composition among disparate functional classes of single-stranded RNA imply adaptive evolutionary convergence. RNA 3:792–806
61. Smit S, Yarus MY, Knight R (2006) Natural selection is not required to explain universal compositional patterns in rRNA secondary structure categories. RNA-A Publ RNA Soc 12:1–14
62. Smith JM (1970) Natural selection and the concept of protein space. Nature 225:563–564
63. Sondek J, Shortle D (1990) Accommodation of single amino acid insertions by the native state of staphylococcal nuclease. Proteins 7:299–305
64. Svedberg T, Fahraeus R (1926) A new method for the determination of the molecular weights of proteins. J Am Chem Soc 48:430–438
65. Tompa P (2002) Instrinsically unstructured proteins. Trends Biochem Sci 27:527–533
66. Urfer R, Kirschner K (1992) The importance of surface loops for stabilizing an eightfold beta alpha barrel protein. Protein Sci 1:31–45
67. Uhlenbeck OC (1995) Keeping RNA happy. RNA 1:4–6
68. van Holde KE (2003) Reflections on a century of protein chemistry. Biophys Chem 100:71–79
69. Weissmann C (2004) The state of proin. Nat Rev Microbiol 2:861–871
70. Wilson DS, Szostak JW (1999) In vitro selection of functional nucleic acids. Annu Rev Biochem 68:611–647
71. Woese CR (2000) Interpreting the universal phylogenetic tree. Proc Natl Acad Sci 97:8392–8396
72. Zuker M, Stiegler P (1981) Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. Nucleic Acids Res 9:133–149
73. Zuker M (2003) mfold web server for nucleic acid folding and hybridization prediction. Nucleic Acids Res 31:3406–3415
74. Zukerkandl E, Pauling L (1965) Evolutionary divergence and convergence in proteins. In: Bryson V, Vogel H (eds) Evolving genes are proteins. Academic Press, New York

Niching Methods: Speciation Theory Applied for Multi-modal Function Optimization

Ofer M. Shir and Thomas Bäck

Abstract While contemporary Evolutionary Algorithms (EAs) excel in various types of optimizations, their generalization to speciational subpopulations is much needed upon their deployment to multi-modal landscapes, mainly due to the typical loss of population diversity. The resulting techniques, known as niching methods, are the main focus of this chapter, which will provide the motivation, pose the problem both from the biological as well as computational perspectives, and describe algorithmic solutions. Biologically inspired by organic speciation processes, and armed with real-world incentive to obtain multiple solutions for better decision making, we shall present here the application of certain bioprocesses to multi-modal function optimization, by means of a broad overview of the existing work in the field, as well as a detailed description of specific test cases.

1 Introduction

Optimal behavior of natural systems is frequently encountered at all levels of everyday life, and thus has become a major source of inspiration for various fields. The discipline of natural computing aims at developing computational techniques that mimic collective phenomena in nature that often exhibit excellent behavior in information processing. Among a long list of natural computing branches, we are particularly interested in the fascinating field of *organic evolution*, and its computational derivative, the so-called *Evolutionary Algorithms* (EAs) field. By encoding an optimization problem into an artificial biological environment, EAs mimic certain elements in the Darwinian dynamics and aim at obtaining highly-fit solutions in terms of the problem. A population of trial solutions undergo artificial variations and survive this simulation upon the criteria posed by the selection mechanism. Analogously, it is suggested that this population would evolve into highly-fit solutions of the optimization problem.

The original goal of this work was to extend specific variants of EAs, called Evolution Strategies (ES), to subpopulations of trial solutions which evolve in parallel to various solutions of the problem. This idea stems from the evolutionary concept of *organic speciation*. Essentially, the *natural computing* way of thinking is

O.M. Shir (✉)
Natural Computing Group, Leiden University, Leiden, The Netherlands
e-mail: oshir@liacs.nl
url: http://natcomp.liacs.nl

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_35, © Springer-Verlag Berlin Heidelberg 2009

required here to further deepen into evolutionary biology theory, and attain creative solutions for the artificial population in light of the desired speciation effect. The so-called *Niching* techniques are the extension of EAs to speciational subpopulations. They have been investigated since the early days of EAs, mainly within the popular variants of Genetic Algorithms (GAs). In addition to the theoretical challenge to design such techniques, which is well supported by the biological-inspired motivation, there is a *real-world incentive* for this effort. The discipline of *decision making*, which makes direct benefit out of the advent of the global optimization field, poses the demand for the multiplicity of different optimal solutions. Ideally, those multiple solutions, as obtained by the optimization routine, would have high diversity among each other, and represent different *conceptual designs*.

The remainder of this chapter is organized as follows. Section 2 will formally provide the reader with the motivation for this study. Section 3 will then outline the relevant biological components upon which this work is based, with clear emphasis on *speciation* and *organic diversity*. The algorithmic framework, the so-called derandomized evolution strategies, will be described in Sect. 4. The linkage between organic speciation to evolutionary algorithms, namely *niching techniques*, will be discussed in Sect. 5. In Sect. 6, we shall introduce in detail our proposed niching framework, and in Sect. 7, we will report on our experimental observation. Finally, Sect. 8 will conclude this chapter and propose directions for future research in this domain.

2 Motivation: Speciation Theory vs. Conceptual Designs

Evolutionary algorithms have the tendency to lose diversity within their population of feasible solutions and to converge into a single solution [1–3], even if the search landscape has multiple globally optimal solutions.

Niching methods, the extension of EAs to finding multiple optima in multi-modal optimization within one population, address this issue by maintaining the diversity of certain properties within the population. Thus, they aim at obtaining parallel convergence into multiple attraction basins in the multi-modal landscape within a single run.

The study of niching is challenging both from the *theoretical* point of view and from the *practical* point of view. The theoretical challenge is two-fold—maintaining the diversity within a population-based stochastic algorithm from the computational perspective, but also having an insight into *speciation* theory or *population genetics* from the evolutionary biology perspective. The practical aspect provides a real-world incentive for this problem—there is an increasing interest of the *applied optimization community* in providing the decision maker with multiple solutions which ideally represent different conceptual designs, for single-criterion or multi-criteria search spaces [4, 5]. The concept of "*going optimal*" is often extended now into the aim for "*going multi-optimal*", so to speak: *obtaining optimal results but also providing the decision maker with different choices*. On this particular note, it is worth mentioning the so-called 2*nd Toyota Paradox* [6]:

> "*Delaying decisions, communicating ambiguously, and pursuing an excessive number of prototypes, can produce better cars faster and cheaper.*"

Niching methods have been studied in the past 35 years, mostly in the context of genetic algorithms, and the focus has been mainly on the theoretical aspect. As will be discussed here, niching methods have been mostly a by-product of studying *population diversity*, and were hardly ever at the front of the EC research.

3 From DNA to Organic Diversity

In this section, we introduce the *biological* elementary concepts that correspond to the core of niching methods: *population diversity*. This section is mainly based on [7].

A Preliminary Note on Terminology

A species is defined as the *smallest evolutionary independent unit*. The term *niche*, however, stems from ecology, and it has several different definitions. It is sometimes referred to as the collective environmental components which are favored by a specific species, but could also be considered as the ecosystem itself which hosts individuals of *various species*. Most definitions would typically also consider the *hosting capacity* of the niche, which refers to the limited available resources for sustaining life in its domain.

In the context of function optimization, *niche* is associated with a *peak*, or a basin of attraction, whereas a *species* corresponds to the subpopulation of individuals occupying that *niche*.

3.1 Genetic Drift

Organic evolution can be broken down into four defining fundamental mechanisms: *natural selection*, *mutation*, *migration*, or *gene flow*, and *genetic drift*. The latter, which essentially refers to *sampling errors in finite populations*, was overlooked by Darwin, who had not been familiar with Mendelian genetics, and thus did not discuss this effect in his "Origin of Species" [8].

In short, *genetic drift* is a stochastic process in which the diversity is lost in finite populations. A distribution of genetic properties is transferred to the next generation in a limited manner, due to the finite number of generated offspring, or equivalently the limited statistical sampling of the distribution. As a result, the distribution is likely to approach an *equilibrium distribution*, e.g., fixation of specific alleles when subject to equal fitness. This is why *genetic drift* is often considered as a *neutral effect*. The smaller the population, the faster and stronger this effect occurs. An analogy is occasionally drawn between genetic drift to *Brownian motion* of particles in mechanics.

In order to demonstrate the genetic drift effect, we conducted simulations[1] on the following basic model of population genetics: The evolution of random-mating populations with *two alleles*, namely, **A** and **a**, equal fitnesses of the *three genotypes* (i.e., no preferences for **AA**, **Aa**, nor **aa**), no mutations, no migration between the replicate populations, and finite population size N. We simulated ten simultaneously evolving populations, for three test-cases of population sizes: $N_1 = 10$, $N_2 = 100$, and $N_3 = 1000$. Figure 1 offers an illustration for the three different simulations. It is easy to observe a clear trend in this simple experiment: Alleles' loss/fixation is very likely to occur in small population sizes, and is not likely to occur in large population sizes.

The *genetic drift* effect had been originally recognized by Fisher [9] (referred to as *random survival*), and was explicitly mentioned by Wright when studying Mendelian populations [10]. It was, however, revisited and given a new interpretation in the *Neutral Theory of Molecular Evolution* of Kimura [11]. The *neutral theory* suggested that the *random genetic drift* effect is the main driving force within molecular evolution, rather than the *non-random natural selection* mechanism. The combination of *natural selection* and *genetic drift* is considered nowadays, by the contemporary evolutionary biology community, as the driving force of organic evolution. Moreover, the importance of the *Neutral Theory* is essentially in its being a *null hypothesis model* for the *Natural Selection Theory* by definition.

3.2 Organic Diversity

Diversity among individuals or populations in nature can be attributed to different evolutionary processes which occur at different levels. We distinguish here between variations that are observed within a single species to a *speciation* process, during which a new species arises, and review shortly both of them.

Variations Within a Species

Diversity of organisms within a single species stems from variance at the genotypic level, referred to as *genetic diversity*, or from the existence of spectrum of phenotypic realizations to a specific genotype. These effects are quantified and are usually associated with *genotypic variance* and *phenotypic variance*, respectively. Several hypotheses explaining *genetic diversity* have been proposed within the discipline of *population genetics*, including the *neutral evolution theory*. It should be noted that genetic diversity is typically considered to be advantageous for survival, as it may

[1] Simulations were conducted with the *PopG Genetic Simulation Program*, version 3.1.

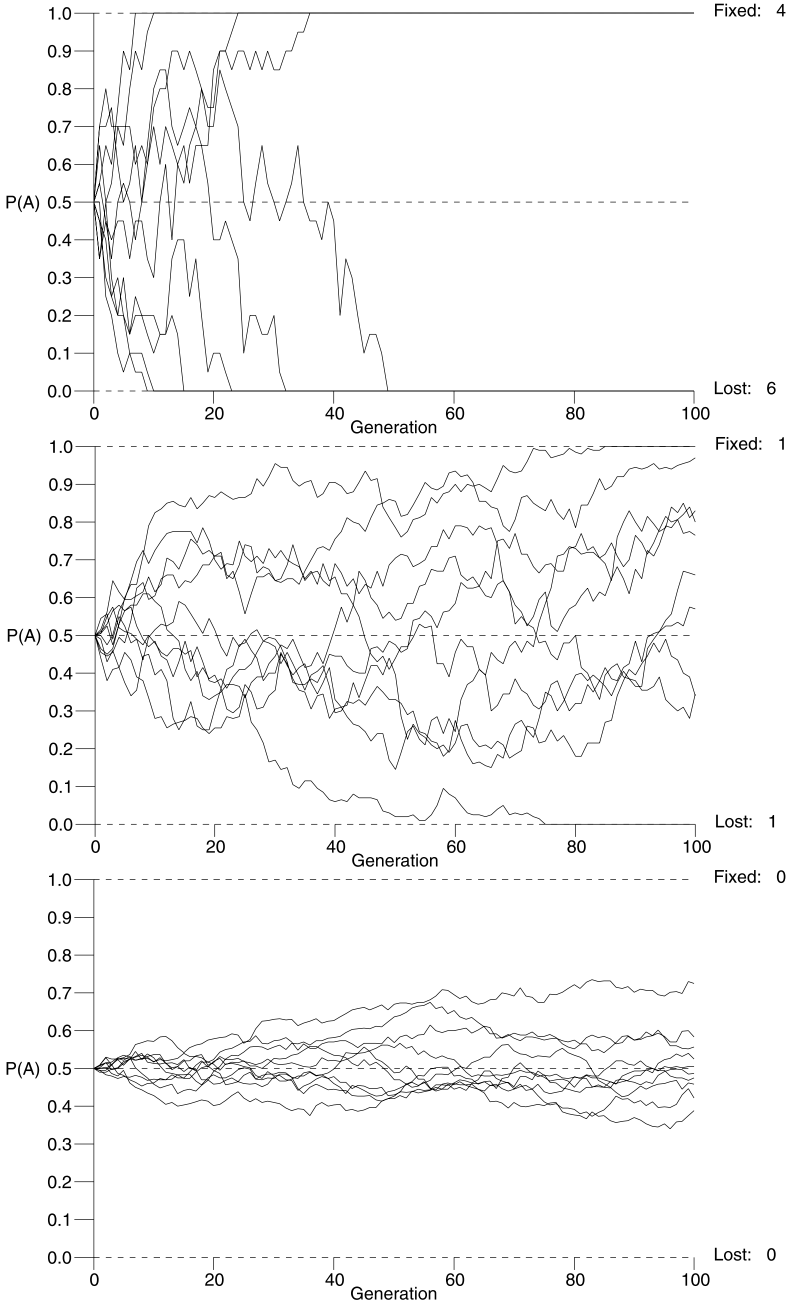


Fig. 1 Ten simultaneously evolving populations, for three test-cases of population sizes: $N_1 = 10$ [*top*], $N_2 = 100$ [*center*], and $N_3 = 1000$ [*bottom*]. The vertical axis corresponds to the allele frequency of **A** in the population, as a function of generations, indicated on the *horizontal axis*

allow better adaptation of the population to environmental changes, such as climate variations, diseases, etc.

Phenotypic variance is measured on a continuous spectrum, also known as quantitative variation. Roughly speaking, the main sources of quantitative variations [7, 12] are outlined here:

1. Genes have *multiple loci*, and hence are mapped into a large set of phenotypes.
2. *Environmental effects* have direct influence on natural selection; fitness is time-dependent, and thus phenotypic variations in the outcome of selection are expected.
3. *Phenotypic plasticity* is the amount in which the *genotypic expression* vary in different environments,[2] and it is a direct source of variation at the phenotypic level.
4. The plastic response of the genotype to the environment, i.e., the joint effect of genetic and environmental elements, also affects the selection of a specific phenotype, and thus can lead to variations. This effect is known as *Genotype-Environment Interaction* ("G-by-E").

Thus, *quantitative variations* are mainly caused by genotypic and phenotypic realizations and their interaction with the environment. The ratio between *genetic variance* to total *phenotypic variance* is defined as *heritability* [10].

Speciation

The essence of the speciation process is *lack of gene flow*, where physical isolation often plays the role of the barrier to gene flow. Lack of gene flow is only one of the necessary conditions for speciation. Another necessary condition for speciation to occur is that the reduction of gene flow will be followed by a phase of *genetic divergence*, by means of *mutation*, *selection*, and *drift*. Finally, the completion or elimination of divergence can be assessed via the so-called *secondary contact* phase: interbreeding between the parental populations would possibly fail (offspring is less fit), succeed (offspring is fitter) or have a neutral outcome (offspring has the same fitness). This would correspond respectively to increasing, decreasing, or stabilizing the differentiation between the two arising species. Note that the speciation can occur de facto, without the actual secondary contact taking place; the latter is for observational assessment purposes.

In organic evolution, four different levels of *speciation* are considered, corresponding to four levels physical linkage between the subpopulations:

1. *Allopatric speciation* The split in the population occurs only due to complete geographical separation, e.g., migration or mountain building. It results in two geographically isolated populations.

[2] Bradshaw [13] gave the following qualitative definition to phenotypic plasticity: "The amount by which the expressions of individual characteristics of a genotype are changed by different environments is a measure of the plasticity of these characters".

2. *Peripatric speciation* Species arise in small populations which are not geographically separated but rather isolated in practice; the effect occurs mainly due to the *genetic drift* effect.
3. *Parapatric speciation* The geographical separation is limited, with a physical overlap between the two zones where the populations split from each other.
4. *Sympatric speciation* The two diverging populations co-exist in the same zone, and thus the speciation is strictly non-geographical. This is observed in nature in parasite populations, that are located in the same zone, but associated with different plant or animal hosts [14].

These four modes of speciation correspond to four levels of geographically decreasing linkages. Roughly speaking, *statistical association* of genetic components in nature, such as *loci*, typically results from *physical linkage*. In this case, we claim that statistical disassociation, which is the trigger to speciation, originates from gradually decreasing physical linkage.

In summary, speciation typically occurs throughout three steps:

1. Geographic isolation or reduction of gene flow.
2. Genetic divergence (mutation, selection, drift).
3. Secondary contact (observation/assessment).

4 Derandomized Evolution Strategies (DES)

The paradigm of *Evolutionary Computation* (EC), which is gleaned from the model of *organic evolution*, studies populations of candidate solutions undergoing variations and selection, and aims at benefiting from the collective phenomena of their generational behavior. The term Evolutionary Algorithms (EAs) essentially refers to the collection of such generic methods, inspired by the theory of *natural evolution*, that encode complex problems into an artificial biological environment, define its genetic operators, and simulate its propagation in time. Motivated by the basic principles of the Darwinian theory, it is suggested that such simulation would yield an optimal solution for the given problem.

Evolutionary algorithms [1] have three main streams, rooted either in the United States and in Germany, during the 1960s: *Evolutionary Programming* (EP), founded by Fogel in San Diego [15], *Genetic Algorithms* (GAs) founded by Holland in Ann Arbor [16, 17], and *Evolution Strategies* (ES), founded by Bienert, Schwefel and Rechenberg, three students to that time at the Technical University of Berlin (see, e.g., [18–20]).

Derandomized evolution strategies for *global parameter optimization*, the general framework of this study, is reviewed in this section.

4.1 Evolutionary Algorithms

Whereas ES and EP are similar algorithms and share many basic characteristics [21], the principal difference between them and GAs is the *encoding of the genetic*

Algorithm 1 An Evolutionary Algorithm

```
$t \leftarrow 0$
$P_t \leftarrow$ init() $\{P_t \in \mathcal{S}^{\mu}$: Set of solutions$\}$
evaluate($P_t$)
**while** $t < t_{\max}$ **do**
    $G_t \leftarrow$ generate($P_t$) {Generate $\lambda$ variations}
    evaluate($G_t$)
    $P_{t+1} \leftarrow$ select($G_t \cup P_t$) {Rank and select $\mu$ best}
    $t \leftarrow t + 1$
**end while**
```

information. Traditional GAs encode the genome with discrete values (as in nature), whereas ES as well as EP do that with continuous real-values. Moreover, ES and EP focused more on development of mutation operators, while in classical GA research the recombination operator received most attention. Today, GA, ES, and EP subsume under the term Evolutionary Algorithms (EAs).

Here, we offer an introductory generic description of an EA. The latter considers a *population* (i.e., set) of *individuals* (i.e., trial solutions), and models its collective learning process. Each individual in the population is initialized according to an algorithm-dependent procedure, and may carry not only a specific search point in the landscape, but also some environmental information concerning the search. A combination of stochastic as well as deterministic processes such as *mutation*, *recombination*, and *selection*, dictate the propagation in time toward successively *better* individuals, corresponding to better regimes of the landscape. The quality of an individual or alternatively the merit of a trial solution are determined by a so-called *fitness function*, which is typically the objective function or its rescaling. Thus, certain individuals are favored over others during the selection phase, which is based upon the fitness evaluation of the population. The selected individuals become the candidate solutions of the next generation, while the others die out.

More explicitly, an EA starts with initializing the *generation* counter t. After generating the initial population with μ individuals in $\mathcal{S}$, a set G_t of λ new solutions are generated by means of *mutation* and possibly *recombination*. The new candidate solutions are evaluated and ranked in terms of their quality (*fitness* value). The μ best solutions in $G_t \cup P_t$ are selected to form the new *parent* population P_{t+1}.

A generalized EA pseudo-code is outlined in Algorithm 1.

4.2 Canonical Evolution Strategies

Evolution strategies were originally developed at the Technical University of Berlin as a procedure for automated experimental design optimization, rather than a global optimizer for continuous landscapes. Following a sequence of successful applications (e.g., shape optimization of a bended pipe, drag minimization of a joint plate, and hardware design of a two-phase flashing nozzle), a diploma thesis [22], and a dissertation [23] laid out the solid foundations for ES as an optimization methodology. There has been extensive work on ES analysis and algorithmic design since then [20, 24, 25]. We shall review here the basic components of a canonical evolution strategy.

General

Evolution strategies consider a population of candidate solutions of the given problem. This population undergoes stochastic as well as deterministic variations, with the so-called *mutation* operator, and possibly with the *recombination* operator. The mutation operator is typically equivalent to sampling a random variation from a *normal distribution*. Due to the continuous nature of the parameter space, the biological term *mutation rate* can be associated here with the actual size of the mutation step in the decision space, also referred to as the *mutation strength*.

Representation

An individual is represented by a tuple of continuous real-values, sometimes referred to as a *chromosome*, which comprises the decision parameters to be optimized, $\mathbf{x}$, their fitness value, $f(\mathbf{x})$, as well as a set of *endogenous* (i.e., evolvable) strategy parameters, $\mathbf{s} \in \mathbb{R}^m$:

The kth individual of the population is thus denoted by

$$\mathbf{a}_k = \big(\mathbf{x}_k, \mathbf{s}_k, f(\mathbf{x}_k)\big).$$

The dimension m of the strategy parameter space is subject to the desired parameter control approach, to be discussed shortly. The endogenous parameters are a unique concept for ES, in particular in the context of the mutation operator, and they play a crucial role in the so-called *self-adaptation principle*.

Next, we outline the two essential ES operators, while omitting the description of others (e.g., recombination).

Mutation

The ES mutation operator considers stochastic continuous variations, which are based on the *multi-variate normal distribution* with zero mean value.. Given a normally-distributed random vector, denoted by $\mathbf{z} = (z_1, z_2, \ldots, z_n)^T$, the mutation operator is then defined as follows:

$$\mathbf{x}^{\text{NEW}} = \mathbf{x}^{\text{OLD}} + \mathbf{z}. \tag{1}$$

Essentially, the $(n \cdot (n+1))/2$ independent elements of the covariance matrix, which uniquely defines the normal distribution, are the endogenous strategy parameters that evolve along with the individual:

$$\mathbf{s} \leftarrow \mathbf{C},$$

i.e., the strategy parameter vector **s** represents the covariance matrix **C** in this case. We choose to omit the details concerning the update rule for the strategy parameters, and refer the reader to [1].

Selection

Natural selection is the driving force of organic evolution: clearing-out an old generation, and allowing its individuals with the fitness advantage to increase their representation in the genetic pool of future generations. As dramatic as it might sound, *death* is an essential part in this process.

Evolution strategies adopt this principle, and apply *deterministic* operators in order to select the best μ individuals with the highest fitness, e.g. minimal objective function values, to be transferred into the next generation. Two selection operators are introduced in the standard ES using an elegant notation due to Schwefel. The notation characterizes the selection mechanism, as well as the number of parents and offspring involved:

- $(\mu + \lambda)$-selection: the next generation of parents will be the best μ individuals selected out of the *union* of current parents and λ offspring.
- (μ, λ)-selection: the next generation of parents will be the best μ individuals selected out of the current λ offspring.

4.3 Derandomization

Due to the crucial role that the mutation operator plays within evolution strategies, its mutative step-size control (MSC) was investigated intensively. The latter tends to work well in the Standard-ES for the adaptation of a single global step-size, but tends to fail when it comes to the individual step-sizes or arbitrary normal mutations (elements of the *covariance matrix*). In particular, the disruptive effects to which the MSC is subject, were studied at several levels [25, 26]. The so-called *derandomized mutative step-size control* aims to treat those disruptive effects, regardless of the problem dimensionality, population size, etc.

The concept of derandomized evolution strategies has been originally introduced by scholars at the Technical University of Berlin in the beginning of the 1990s. It was followed by the release of a new generation of successful ES variants by Hansen, Ostermeier, and Gawelczyk [27–30].

The first versions of *derandomized ES algorithms* introduced a controlled global step-size in order to monitor the individual step-sizes by decreasing the stochastic effects of the probabilistic sampling. The selection disturbance was completely removed with later versions by omitting the adaptation of strategy parameters by means of probabilistic sampling. This was combined with individual information from the last generation (the successful mutations, i.e. of selected offspring), and then adjusted to *correlated mutations*. Later on, the concept of *adaptation by accumulated information* was introduced, aiming to use wisely the past information for the purpose of step-size adaptation: instead of using the information from the last generation only, it was successfully generalized to a weighted average of the previous generations.

Note that the different derandomized-ES variants strictly follow a $(1,\lambda)$ strategy, postponing the treatment of recombination or plus-strategies for later stages. In this way, the question how to update the strategy parameters when an offspring does not improve its ancestor is not relevant here.

Moreover, the different variants hold different numbers of strategy parameters to be adapted, and this is a factor in the learning speed of the optimization routine. The different algorithms hold a number of strategy parameters in either $\mathcal{O}(n)$ or $\mathcal{O}(n^2)$ in terms of the dimensionality n of the search problem.

In our work, we consider 5 specific DES variants: **DR1** [27], **DR2** [28], **DR3** [29], the CMA comma-strategy [25, 30], and its elitist sibling, the CMA plus-strategy [31]. The first two possess a first-order learning mechanism (a vector of variations), while the other three possess a second-order learning mechanism (full covariance matrix).

5 Introduction to Niching

5.1 Population Diversity in Evolution Strategies

Subject to the complex dynamics of the various forces within an evolutionary algorithm, population diversity is typically lost, and the search is likely to converge into a single basin of attraction in the landscape.

Population diversity loss within the population of solutions is the fundamental effect which niching methods aim to treat. In fact, from the historical perspective, the quest for diversity-promoting-techniques was the main goal within the EC community for some time, and niching methods were merely obtained as *by-products*, so to speak, of that effort. As will be argued here, population diversity is an important component in a population-based search.

Next, we describe the effect of *diversity loss* within evolution strategies. This will be followed by some conclusions drawn by the GA research concerning diversity loss with GAs, as a point of reference to ES.

5.2 Diversity Loss in Evolution Strategies

The defining mechanism of ES is strongly dictated by the mutation operator as well as by the deterministic selection operator. As defining operators, they have a direct influence on the diversity property of the population. The recombination operator, nevertheless, does not play a critical role in the ES mechanism. In practice, especially in the context of derandomized ES, it is not an essential component.

We attribute two main components to the *population diversity loss* within ES: fast *take-over*, which is associated with the *selection* operator, and *genetic drift* (or *neutrality* effect), which is associated both with the *selection* and the *recombination* operators, respectively.

Selective Pressure: Fast Take-over

Evolution strategies have a strictly deterministic, rank-based approach, to selection. In the two traditional approaches, (μ, λ) and $(\mu + \lambda)$, the best individuals are selected—implying, rather intuitively, high *selective pressure*. Due to the crucial role of the selection operator within the evolution process, its impact within the ES field has been widely investigated.

Furthermore, Goldberg and Deb introduced the important concept of *takeover time* [32], which gives a quantitative description of selective pressure *with respect only to the selection operator*:

Definition 1 The *takeover time* τ^* is the minimal number of generations until repeated application of the selection operator yields a uniform population filled with copies of the best individual.

The selective pressure has been further investigated by Bäck [2], who analyzed all the ES selection mechanisms also with respect to takeover times. It concluded that upon employing the typical selection mechanisms, very short *takeover times* are yielded. This result implies that ES are typically subject to high *selective pressures*.

ES Genetic Drift

We consider two different ES neutral effects, that could be together ascribed as a general ES genetic drift: *recombination drift*, and *selection drift*. We argue that these two components are directly responsible to the loss of population diversity in ES.

Recombination Drift

Beyer explored extensively the so-called *mutation-induced speciation by recombination* (MISR) principle (see, e.g. [33]). According to this important principle, repeated application of the mutation operator, subject to a dominant recombination operator, would lead to a stable distribution of the population, which resembles a species or a cloud of individuals. When fitness-based selection is applied, this cloud is likely to move together towards fitter regions of the landscape. Furthermore, Beyer managed to prove analytically [33] that the MISR principle is indeed universal when finite populations are employed, subject to sampling-based recombination. The latter was achieved by analyzing the ES dynamics without fitness-based selection, deriving the expected population variance, and showing that it is reduced with random sampling in finite populations. This result was also corroborated by numerical simulations. That study provides us with an analytical result that a sampling-based recombination is subject to genetic drift, and leads to loss of population diversity.

Selection Drift

On the other hand, a recent study on the extinction of subpopulations on a simple *bi-modal equi-fitness* model investigated the drift effect of the selection operator [34]. It considered the application of *selection* on finite populations, when the fitness values of the different attractors were equal (i.e. eliminating the possibility of a *take-over effect*), and argued that a neutral effect (*drift*) would occur, pushing the population into a single attractor. The latter study indeed demonstrated this effect of *selection drift* in ES, which resulted in a convergence to an equilibrium distribution around a single attractor. It was also shown that the time of extinction increases proportionally with μ. The analysis was conducted by means of Markov chain models supported by statistical simulations.

Corollary 1 *Evolution strategies that employ finite populations are typically underposed to several effects that are responsible for the loss of population diversity. It has been shown that the standard selection mechanisms may lead to a fast take-over effect. In addition, we argued that both the* recombination *and the* selection *operators experience their own* drift *effects that lead to population diversity loss. We conclude that an evolution strategy with a* small population *is likely to encounter a rapid effect of diversity loss.*

5.3 Neutrality in ES Variations: Mutation Drift

The mutation operator, the defining operator of evolution strategies, applies normally-distributed variations of finite sample sizes, and thus is expected to experience sampling errors as the sample sizes decrease. These sampling errors lead to an undirected movement of the population center of mass, with speed which depends on the population size. We shall call this effect *mutation drift*.

Simulations

In order to demonstrate and analyze this *mutation drift* effect, we conducted simulations on the following basic ES model: The parallel evolution of several populations in an n-dimensional space, based on sequential normally-distributed variations (with a fixed identity matrix as the covariance of the distribution), without selection nor recombination. The ES variation can be then considered as a *continuous random walk* of μ individuals in an n-dimensional space. Essentially, this corresponds to *mutation-only ES* of multiple populations.

We simulated 10 simultaneously evolving populations, for three test-cases of population sizes: $\mu_1 = 10$, $\mu_2 = 100$, and $\mu_3 = 1000$, subject to three space dimensions $n_1 = 1$, $n_2 = 10$, and $n_3 = 1000$. For each simulation, we measured the distance of the *population mean*, or *center of mass*, to the starting point, as a function of generational steps. More precisely, we measured the location of the *population mean* for n_1, and the *Euclidean distance from the origin* for $\{n_2, n_3\}$. Figure 2 presents the outcome of these calculations.

It is easy to observe in those simulations a similar trend in the equivalent simulations of Sect. 3.1: The population center of mass is strongly drifted away from the origin when it is small, and has the contrary effect when the population is large. We therefore conclude that mutation drift is very likely to occur in small population sizes, and is not likely to occur in large population sizes. This is the essence of the distribution's *sampling error*, which we chose to define here as *mutation drift*.

We thus demonstrated here that the *center of mass* of a small ES population is subject to a so-called *mutation drift*. This is an equivalent effect to the genetic drift of alleles, as described in Sect. 3.1. We claim that it allows for easy translation of small populations from one location to another, having the potential to boost fast and efficient *speciation*. Therefore, we argue that drift in this context can be a blessing for the fast formation of species in niching.

Since small populations are typically employed by evolution strategies, and especially by the derandomized variants, we consider this effect of *mutation drift* as a positive potential component for niching with ES. This result provides us with further motivation to introduce DES with small populations into the niching framework.

5.4 Niching Methods

Despite the fact that the motivation for multi-modal optimization is beyond doubt, and the biological inspiration is real, there is no unique definition of the goal statement for *niching techniques*. There have been several attempts to provide a proper definition and functional specification for niching; we review some of them here:

1. Mahfoud [3] chose to put emphasis on locating as well as maintaining good optima, and formulated the following:

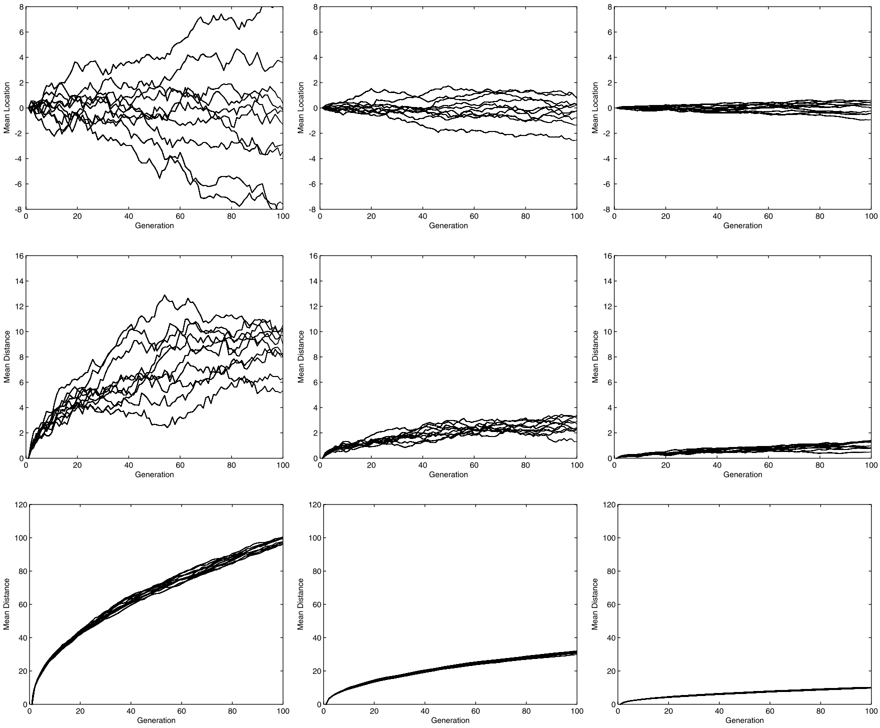


Fig. 2 Illustration of the *mutation drift* effect in ES, for 10 simultaneously evolving populations, as a function of population size [$\mu_1 = 10$ (*left*), $\mu_2 = 100$ (*center*), and $\mu_3 = 1000$ (*right*)] and landscape dimensionality [$n_1 = 1$ (*top*), $n_2 = 10$ (*center*), and $n_3 = 1000$ (*bottom*)]. The vertical axes correspond to the location of the center of mass of the population (for $n_1 = 1$, *top row*) or distance from the origin to the center of mass of the population (for $n_2 = 10$ or $n_3 = 1000$, in the *center* or *bottom rows*, respectively). The *horizontal axis* corresponds to the generational step of the calculation

> The litmus test for a niching method, therefore, will be whether it possesses the capability to find multiple, final solutions within a reasonable amount of time, and to maintain them for an extended period of time.

2. Beyer et al. [35] put forward also the actual maintenance of population diversity:

> *Niching*: process of separation of individuals according to their states in the search space or maintenance of diversity by appropriate techniques, e.g. local population models, fitness sharing, or distributed EA.

3. Preuss [36] considered the two definitions mentioned above, and proposed a third:

> Niching in EAs is a two-step procedure that (a) concurrently or subsequently distributed individuals onto distinct basins of attraction and (b) facilitates approximation of the corresponding (local) optimizers.

We choose to adopt Preuss' mission statement, and define the challenge in niching as follows:

> Attaining the optimal interplay between partitioning the search space into niches occupied by stable subpopulations, by means of population diversity preservation, versus exploiting the search in each niche by means of a highly efficient optimizer with local-search capabilities.

GA Niching Methods

Niching methods within genetic algorithms have been studied during the past few decades (see, e.g. [3] and [37]), initially triggered by the necessity to promote *population diversity* within EAs. The research has yielded a variety of different methods, which are the vast majority of existing work on niching in general. Especially, the important concept of *fitness sharing* should be outlined.

The *sharing* concept was one of the pioneering niching approaches. It was first introduced by Holland in 1975 [17], and later implemented as a niching technique by Goldberg and Richardson [38]. This strong approach of *considering the fitness as a shared resource* has essentially become an important concept in the broad field of evolutionary algorithms, as well as laid the foundations for various successful niching techniques for multi-modal function optimization, mainly within GAs. The basic idea of *fitness sharing* is to consider the fitness of the landscape as a resource to be shared among the individuals, in order to decrease redundancy in the population. In practice, the individual raw fitness is scaled by its niche count, which is an indicator for the number of other individuals in its proximity. Thus, an individual is penalized in the selection phase for being in a crowded region with other individuals. This concept follows to some degree the *hosting capacity* concept of an ecological niche, as discussed previously.

Niching in Evolution Strategies

Researchers in the field of evolution strategies initially showed no particular interest in the field of niching, leaving it essentially for genetic algorithms. An exception would be the employment of island models. Roughly speaking, classical niching mechanisms such as *fitness sharing*, which redefine the selection mechanism, are likely to interfere with the core of evolution strategies—the *self-adaptation mechanism*—and thus doomed to fail in a straightforward implementation. Any manipulation of the fitness value is usually not suitable for evolution strategies, as in the case of constraints handling: death-penalty is typically the chosen approach for a violation of a constraint in ES, rather than a continuous punishment as used in other EAs, in order to avoid the introduction of disruptive effects to the self-adaptation mechanism (see, e.g. [39, 40]). Therefore, niching with evolution strategies would have to be addressed from a different direction. Moreover, the different nature of the ES dynamics, throughout the *deterministic selection* and the *mutation operator*, suggests as well that a different treatment is required here.

There are several, relatively new, niching methods that have been proposed within ES, mostly clustering-based [34, 41, 42]. A different approach, which preceded this work, was presented in [43–45].

6 Proposed Framework: Niching with DES Kernels

Following our *mission statement*, as presented in the previous section, we were aiming at constructing a generic niching framework which offers the combination of population diversity preservation and local-search capabilities. We considered derandomized evolution strategies as the best choice for that purpose, as an EA with local search characteristics. Furthermore, DES typically employ small populations, which was shown to be a potential advantage for a niching technique, as it can boost the speciation effect (Sect. 5.3).

6.1 The Proposed Algorithm

The advent of derandomized evolution strategies allows successful global optimization with minimal requirements concerning exogenous parameters, mostly without recombination, and with a low number of function evaluations. In particular, consider the $(1 \overset{+}{,} \lambda)$ derandomized ES variants presented earlier. In the context of niching, this generation of modern ES variants allows the construction of fairly simple and elegant niching algorithms. Next, we outline our proposed method.

Our niching technique is based upon interacting search processes, which simultaneously perform a derandomized $(1, \lambda)$ or $(1 + \lambda)$ search in different locations of the space. In case of multi-modal landscapes, these search processes are meant to explore different attractor basins of local optima.

An important point in our approach is to strictly enforce the fixed allocation of the population resources, i.e. number of offspring per niche. The idea is thus to prevent a scenario of a take-over, where a subpopulation located at a fitter optimum can generate more offspring. The biological idea behind this fixed allocation of resources lies in the concept of limited *hosting capacities* of given ecological niches, as previously discussed.

The *speciation interaction* occurs every generation when all the offspring are considered together to become niches' representatives for the next iteration, or simply the next search points, based on the rank of their fitness and their location with respect to higher-ranked individuals. We focus in a simple framework without recombination ($\mu = 1$).

6.2 *Niching with* $(1 \overset{+}{,} \lambda)$ *DES Kernels*

Given q, the estimated/expected number of peaks, $q + p$ "D-sets" are initialized, where a D-set is defined as the collection of all the dynamically adapted strategy as

Algorithm 2 Dynamic Peak Identification (DPI)

```
input: population Pop, number of niches q, niche radius $\rho$
Sort Pop in decreasing fitness order
$i := 1$
$NumPeaks := 0$
$DPS := \emptyset$ {Set of peak elements in population}
while $NumPeaks \neq q$ and $i \leq popSize$ do
    if $Pop[i]$ is not within sphere of radius $\rho$ around peak in $DPS$ then
        $DPS := DPS \cup \{Pop[i]\}$
        $NumPeaks := NumPeaks + 1$
    end if
    $i := i + 1$
end while
output: DPS
```

well as decision parameters of the derandomized algorithm, which uniquely define the search at a given point of time. These parameters are the current search point, the mutation vector/covariance matrix, the step-size, as well as other auxiliary parameters. At every point in time, the algorithm stores exactly $q + p$ D-sets, which are associated with $q + p$ search points: q for the peaks and p for the "non-peaks domain". The $(q + 1)^{th}...(q + p)^{th}$ D-sets are individuals which are randomly regenerated every *epoch*, i.e. a cycle of κ generations, as potential candidates for niche formation. This is basically a *quasi-restart* mechanism, which allows new niches to form dynamically. We stress that the total number of function evaluations allocated for a run should depend on the number of desired peaks, q, and not on p. Setting the value of p should reflect the dilemma between applying a wide restart approach for exploring further the search space and exploiting computational resources for the existing niches. In any case, due to the *curse of dimensionality*, p loses its significance as the dimension of the problem increases.

Until the stopping criterion is met, the following procedure takes place. Each search point samples λ offspring, based on its evolving D-set. After the fitness evaluation of the new $\lambda \cdot (q + p)$ individuals, the classification into niches of the entire population is obtained in a *greedy* manner, by means of the DPI routine [46] (Algorithm 2). The latter based on the fixed niche radius ρ. The peaks then become the new search points, while their D-sets are inherited from their parents and updated, respectively.

Algorithm Dynamics

We would like to point out the dynamic nature of the subpopulations dynamics. Due to the *greedy* classification to niches, which is carried out every generation, some niches can merge in principle, while all the individuals, except for the *peak individual*, die out in practice. Following our principle of fixed resources per niche,

Algorithm 3 $(1 \overset{+}{,} \lambda)$-CMA-ES Niching with Fixed Niche Radius

```
for $i = 1 \ldots (q + p)$ search points do
    Generate $\lambda$ samples based on the D-set of $i$
end for
Evaluate fitness of the population
Compute the Dynamic Peak Set with the DPI Algorithm
for all elements of $DPS$ do
    Set peak as a search point
    Inherit the D-set and update it respectively
end for
if $N_{DPS}$ = size of $DPS < q$ then
    Generate $q - N_{DPS}$ new search points, reset D-sets
end if
if $gen \bmod \kappa \equiv 0$ then
    Reset the $(q + 1)^{th} \ldots (q + p)^{th}$ search points
end if
```

only the peak individual will be sampled λ times in the following generations. In biological terms, the peak individual could be associated with an *alpha-male*, which wins the local competition and gets all the sexual resources of its ecological niche.

A pseudo-code for the *niching routine* is presented as Algorithm 3.

Sizing the Population

We follow the recommended population size for $(1, \lambda)$ derandomized ES (see, e.g. [26]), and set $\lambda = 10$. On this note, we would like to mention a theoretical work on sizing the population in a derandomized $(1, \lambda)$ ES with respect to the local progress [47]. The latter work obtained theoretical results showing that the local serial progress is maximized when the expected progress of the second best individual vanishes. These results allowed in the construction of a population size adaptation scheme, which sets the value of λ as a function of the fitness difference of the second fittest offspring and its parent. This adaptation scheme was shown to perform well on a set of simple theoretical landscapes [47].

6.3 *Niche Radius Calculation*

The original formula for ρ for *phenotypic sharing* in GAs was derived by Deb and Goldberg [32]. Analogously, by considering the decision parameters as the decoded parameter space of the GA, the same formula can be applied, using the Euclidean metric. Given q, the number of peaks in the solution space, every niche is considered to be surrounded by an n-dimensional hyper-sphere with radius ρ which occupies

$\frac{1}{q}$ of the entire volume of the space. The volume of the hyper-sphere which contains the entire space is

$$V = cr^n \tag{2}$$

where c is a constant, given explicitly by

$$c = \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)}, \tag{3}$$

with $\Gamma(n)$ as the Gamma function. Given lower and upper bound values $x_{k,\min}$, $x_{k,\max}$ of each coordinate in the decision parameters space, r is defined as follows:

$$r = \frac{1}{2}\sqrt{\sum_{k=1}^{n}(x_{k,\max} - x_{k,\min})^2}. \tag{4}$$

If we divide the volume into q parts, we may write

$$c\rho^n = \frac{1}{q}cr^n \tag{5}$$

which yields

$$\rho = \frac{r}{\sqrt[n]{q}}. \tag{6}$$

Hence, by applying this niche radius approach, two assumptions are made:

1. The expected/desired number of peaks, q, is given or can be estimated.
2. All peaks are at least in distance 2ρ from each other, where ρ is the fixed radius of every niche.

7 Experimental Observation

The proposed niching framework has been successfully applied to a suite of artificial multi-modal high-dimensional continuous landscapes, as reported in [48]. It was observed to perform well, both with $(1, 10)$ and $(1 + 10)$ DES kernels (the latter corresponds only to the elitist CMA), while typically obtaining most of the desired basins of attraction of the various landscapes at different dimensions. Furthermore, upon carrying out behavioral analysis of the simulations, some characteristic patterns for the different algorithmic kernels were revealed. The elitist CMA was observed to perform better on average. We choose to omit here specific details concerning the numerical simulations, and refer the reader to [48].

As was furthermore reported in [48], the proposed niching framework was successfully applied to a real-world landscape from the field of quantum control, namely the dynamic molecular alignment problem (for a review, see [49]). This challenging application required the definition of a tailored-made diversity measure, due to certain invariance properties of the control function. It was shown to perform well, and to obtain different pulse-shapes of high-quality, representing different conceptual designs. Also in this case, the elitist-CMA kernel was observed to perform best.

7.1 Extending the Framework: Learning Niche-Shapes

The relaxation of assumptions and limitations concerning the hypothetical landscape is much needed if niching is to be valid in a broader range of applications. Another study, as reported in [50], addressed the so-called *niche radius problem* by introducing the Mahalanobis distance into the niching framework with the CMA kernel, for obtaining niches with more complex geometrical shapes. The proposed approach was tested on high-dimensional artificial landscapes at several levels of difficulty, and was shown to be robust and to achieve satisfying results. Figure 3 presents a snapshot gallery of the niching algorithm with the CMA-plus kernel, employing the Mahalanobis distance, performing on the Fletcher–Powell landscape.

8 Summary and Outlook

We have introduced Niching as an algorithmic framework following the biological process of *speciation*. Upon providing the practical motivation for such a framework, in terms of conceptual designs for better decision making, we outlined in detail the essential biological background, as well as the algorithmic arena of derandomized evolution strategies. We then proposed a specific niching algorithm, which employs DES kernels, subject to a fixed niche-radius approach.

We discussed briefly experimental observations of the proposed niching algorithm, both on artificial as well as real-world high-dimensional multi-modal landscapes, as reported previously in [48]. The conclusion was that the proposed algorithm performed well on multi-modal landscapes. Furthermore, given some invariance properties in the decision space, a tailored-made diversity measure was reported to obtain satisfactory results. Niching with the elitist-CMA kernel was observed to perform best on average. We suggest an explanation for the advantage of a plus strategy for niching. The niching problem can be considered as an optimization task with constraints, i.e. the formation of niches that restricts competing niches and their optimization routine of exploring the search space freely. It has been suggested in previous studies (see, e.g. [40]) that ES self-adaptation in constrained problems will tend to fail with a comma-strategy, and thus a plus-strategy is preferable for such problems. We might link this argumentation to the observation of our numerical results here, and suggest that a plus-strategy is preferable for niching.

Here are some directions for future study in this domain:

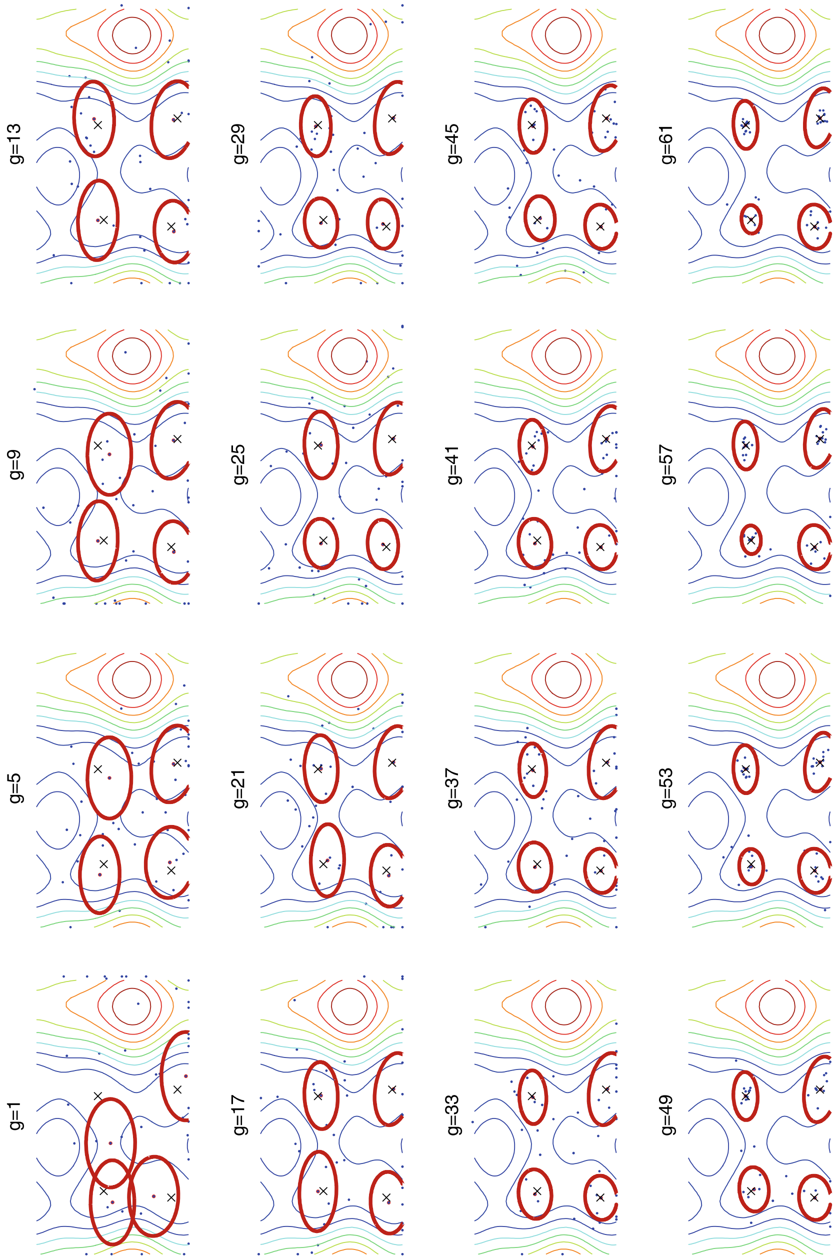


Fig. 3 A snapshot gallery: the adaptation of the *classification-ellipses*, subject to the Mahalanobis metric with the updating covariance matrix, in the CMA+ strategy for a $2D$ Fletcher–Powell problem. Images are taken in the box $[-\pi, \pi]^2$. *Contours* of the landscape are given as the background, where the X's indicate the real optima, the *dots* are the evolving individuals, and the *ellipses* are plotted centered about the peak individual. A snapshot is taken every 4 generations (i.e. every 160 function evaluations), as indicated by the counter

- Transferring this generation of niching algorithms into additional real-world applications.
- Developing elitist versions for all the DES kernels, for employment in the proposed niching framework. Following the experimental observation reported here for the elitist-CMA, such kernels can perform well in the proposed niching algorithm.
- Proceeding with the effort to tackle the niche radius problem, in order to develop state-of-the-art DES niching techniques which are not subject to the niche radius assumptions.

References

1. Bäck T (1996) Evolutionary algorithms in theory and practice. Oxford University Press, New York
2. Bäck T (1994) Selective pressure in evolutionary algorithms: a characterization of selection mechanisms. In: Michalewicz Z, Schaffer JD, Schwefel HP, Fogel DB, Kitano H (eds) Proceedings of the first IEEE conference on evolutionary computation, ICEC'94, Orlando, FL. IEEE Press, Piscataway, pp 57–62
3. Mahfoud S (1995) Niching methods for genetic algorithms. PhD thesis, University of Illinois at Urbana Champaign
4. Avigad G, Moshaiov A, Brauner N (2004) Concept-based interactive brainstorming in engineering design. J Adv Comput Intell Intell Inform 8(5):454–459
5. Avigad G, Moshaiov A, Brauner N (2005) Interactive concept-based search using MOEA: the hierarchical preferences case. Int J Comput Intell 2(3):182–191
6. Cristiano JJ, White CC, Liker JK (2001) Application of multiattribute decision analysis to quality function deployment for target setting. IEEE Trans Syst Man Cybern: Part C 31(3):366–382
7. Freeman S, Herron JC (2003) Evolutionary analysis, 3rd edn. Benjamin Cummings, Redwood City
8. Darwin CR (1999) The origin of species: by means of natural selection or the preservation of favoured races in the struggle for life. Bantam Classics, New York
9. Fisher RA (1922) Darwinian evolution of mutations. Eugen Rev 14:31–34
10. Wright S (1931) Evolution in Mendelian populations. Genetics 16:97–159
11. Kimura M (1983) The neutral theory of molecular evolution. Cambridge University Press, Cambridge
12. Scheiner SM, Goodnight CJ (1984) The comparison of phenotypic plasticity and genetic variation in populations of the grass danthonia spicata. Evolution 38(4):845–855
13. Bradshaw A (1965) Evolutionary significance of phenotypic plasticity in plants. Adv Genet 13:115–155
14. McPheron BA, Smith DC, Berlocher SH (1988) Genetic differences between host races of rhagoletis pomonella. Nature 336:64–66
15. Fogel LJ (1966) Artificial intelligence through simulated evolution. Wiley, New York
16. Holland JH (1962) Outline for a logical theory of adaptive systems. J ACM 9(3):297–314
17. Holland JH (1975) Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor
18. Schwefel HP (1995) Evolution and optimum seeking. Wiley, New York
19. Rechenberg I (1973) Evolutionsstrategies: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart
20. Beyer HG, Schwefel HP (2002) Evolution strategies a comprehensive introduction. Nat Comput Int J 1(1):3–52

21. Bäck T, Rudolph G, Schwefel HP (1993) Evolutionary programming and evolution strategies: similarities and differences. In: Proceedings of the second annual conference on evolutionary programming. Evolutionary Programming Society, La Jolla, pp 11–22
22. Schwefel HP (1965) Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik. Master's thesis, Technical University of Berlin
23. Rechenberg I (1971) Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. PhD thesis, Technical University of Berlin
24. Beyer HG (2001) The theory of evolution strategies. Springer, Heidelberg
25. Hansen N, Ostermeier A (2001) Completely derandomized self-adaptation in evolution strategies. Evol Comput 9(2):159–195
26. Ostermeier A, Gawelczyk A, Hansen N (1994) A derandomized approach to self adaptation of evolution strategies. Evol Comput 2(4):369–380
27. Ostermeier A, Gawelczyk A, Hansen N (1993) A derandomized approach to self adaptation of evolution strategies. Technical report, TU Berlin
28. Ostermeier A, Gawelczyk A, Hansen N (1994) Step-size adaptation based on non-local use of selection information. In: Parallel problem solving from nature (PPSN III). Lecture notes in computer science, vol 866. Springer, Berlin, pp 189–198
29. Hansen N, Ostermeier A, Gawelczyk A (1995) On the adaptation of arbitrary normal mutation distributions in evolution strategies: the generating set adaptation. In: Proceedings of the sixth international conference on genetic algorithms, ICGA6. Morgan Kaufmann, San Francisco, pp 57–64
30. Hansen N, Ostermeier A (1996) Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In: Proceedings of the 1996 IEEE international conference on evolutionary computation. IEEE, Piscataway, pp 312–317
31. Igel C, Suttorp T, Hansen N (2006) A computational efficient covariance matrix update and a $(1+1)$-CMA for evolution strategies. In: Proceedings of the genetic and evolutionary computation conference, GECCO, 2006. ACM Press, New York, pp 453–460
32. Deb K, Goldberg DE (1989) An investigation of niche and species formation in genetic function optimization. In: Proceedings of the third international conference on genetic algorithms. Morgan Kaufmann, San Francisco, pp 42–50
33. Beyer HG (1999) On the dynamics of GAs without selection. In: Banzhaf W, Reeves C (eds) Foundations of genetic algorithms 5. Morgan Kaufmann, San Francisco, pp 5–26
34. Schönemann L, Emmerich M, Preuss M (2004) On the extiction of sub-populations on multimodal landscapes. In: Proceedings of the international conference on bioinspired optimization methods and their applications, BIOMA 2004. Jožef Stefan Institute, Slovenia, pp 31–40
35. Beyer H, Brucherseifer E, Jakob W, Pohlheim H, Sendhoff B, To TB (2002) Evolutionary algorithms—terms and definitions. http://ls11-www.cs.uni-dortmund.de/people/beyer/EA-glossary/
36. Preuss M (2006) Niching prospects. In: Proceedings of the international conference on bioinspired optimization methods and their applications, BIOMA 2006. Jožef Stefan Institute, Slovenia, pp 25–34
37. Singh G, Deb K (2006) Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In: Proceedings of the 2006 annual conference on genetic and evolutionary computation, GECCO 2006. ACM Press, New York, pp 1305–1312
38. Goldberg DE, Richardson J (1987) Genetic algorithms with sharing for multimodal function optimization. In: Proceedings of the second international conference on genetic algorithms and their application. Lawrence Erlbaum Associates, Mahwah, pp 41–49
39. Coello Coello CA (1999) A survey of constraint handling techniques used with evolutionary algorithms. Technical Report Lania-RI-99-04, Laboratorio Nacional de Informática Avanzada, Xalapa, Veracruz, México
40. Kramer O, Schwefel HP (2006) On three new approaches to handle constraints within evolution strategies. Nat Comput Int J 5(4):363–385
41. Aichholzer O, Aurenhammer F, Brandtstätter B, Ebner T, Krasser H, Magele C (2000) Niching evolution strategy with cluster algorithms. In: Proceedings of the 9th biennial IEEE conference on electromagnetic field computations. IEEE Press, New York

42. Streichert F, Stein G, Ulmer H, Zell A (2003) A clustering based niching EA for multimodal search spaces. In: Proceedings of the international conference evolution artificielle. Lecture notes in computer science, vol 2936. Springer, Berlin, pp 293–304
43. Shir OM, Bäck T (2005) Niching in evolution strategies. Technical Report TR-2005-01, LIACS, Leiden University
44. Shir OM, Bäck T (2005) Niching in evolution strategies. In: Proceedings of the genetic and evolutionary computation conference, GECCO-2005. ACM Press, New York, pp 915–916
45. Shir OM, Bäck T (2005) Dynamic niching in evolution strategies with covariance matrix adaptation. In: Proceedings of the 2005 congress on evolutionary computation, CEC-2005. IEEE Press, Piscataway, pp 2584–2591
46. Miller B, Shaw M (1996) Genetic algorithms with dynamic niche sharing for multimodal function optimization. In: Proceedings of the 1996 IEEE international conference on evolutionary computation, ICEC'96, New York, NY, USA, pp 786–791
47. Hansen N, Gawelczyk A, Ostermeier A (1995) Sizing the population with respect to the local progress in $(1, \lambda)$-evolution strategies—a theoretical analysis. In: Proceedings of the 1995 IEEE international conference on evolutionary computation. IEEE, New York, pp 312–317
48. Shir OM, Bäck T (2009) Niching with derandomized evolution strategies in artificial and real-world landscapes. Nat Comput 8(1):171–196
49. Siedschlag C, Shir OM, Bäck T, Vrakking MJJ (2006) Evolutionary algorithms in the optimization of dynamic molecular alignment. Opt Commun 264:511–518
50. Shir OM, Emmerich M, Bäck T (2007) Self-adaptive niching CMA-ES with Mahalanobis metric. In: 2007 IEEE congress on evolutionary computation, CEC. IEEE Computational Intelligence Society, Los Alamitos, pp 820–827

On the Concept of *Cis*-regulatory Information: From Sequence Motifs to Logic Functions

Ryan Tarpine and Sorin Istrail

Abstract The regulatory genome is about the "system level organization of the core genomic regulatory apparatus, and how this is the locus of causality underlying the twin phenomena of animal development and animal evolution" (E.H. Davidson. The Regulatory Genome: Gene Regulatory Networks in Development and Evolution, Academic Press, 2006). Information processing in the regulatory genome is done through regulatory states, defined as sets of transcription factors (sequence-specific DNA binding proteins which determine gene expression) that are expressed and active at the same time. The core information processing machinery consists of modular DNA sequence elements, called *cis*-modules, that interact with transcription factors. The *cis*-modules "read" the information contained in the regulatory state of the cell through transcription factor binding, "process" it, and directly or indirectly communicate with the basal transcription apparatus to determine gene expression. This endowment of each gene with the information-receiving capacity through their *cis*-regulatory modules is essential for the response to every possible regulatory state to which it might be exposed during all phases of the life cycle and in all cell types. We present here a set of challenges addressed by our CYRENE research project aimed at studying the *cis*-regulatory code of the regulatory genome. The CYRENE Project is devoted to (1) the construction of a database, the *cis*-Lexicon, containing comprehensive information across species about experimentally validated *cis*-regulatory modules; and (2) the software development of a next-generation genome browser, the *cis*-Browser, specialized for the regulatory genome. The presentation is anchored on three main computational challenges: the *Gene Naming Problem*, the *Consensus Sequence Bottleneck Problem*, and the *Logic Function Inference Problem*.

1 Introduction

Gene expression is regulated largely by the binding of transcription factors to genomic sequence. Once bound, these factors interact with the transcription apparatus to activate or repress the gene. Unlike a restriction enzyme, which recognizes a single sequence or clearly defined set of sequences, a transcription factor binds to a

R. Tarpine (✉)
Department of Computer Science and Center for Computational Molecular Biology,
Brown University, 115 Waterman Street, Box 1910, Providence, RI 02912, USA
e-mail: ryan@cs.brown.edu

A. Condon et al. (eds.), *Algorithmic Bioprocesses,* Natural Computing Series,
DOI 10.1007/978-3-540-88869-7_36, © Springer-Verlag Berlin Heidelberg 2009

family of similar sequences with varying strengths and effects. While the similarity between sequences bound by a single factor is usually obvious at a glance, there is as yet no reliable sequence-only method for determining functional binding sites. Even the seemingly conservative method of looking for exact sequence matches to known binding sites yields mostly false positives—since binding sites are small, usually less than 15 bases long, by chance alone millions of sequence matches are scattered throughout any genome. Very few of these have any influence on gene regulation. The activity of a putative binding site depends on more than the site sequence, and it is hoped that the site context (i.e., the surrounding sequence) contains the necessary information. To predict new binding sites, we must understand the biology of gene regulation and incorporate the missing sources of regulatory information into our model.

To this end, we have created a database for storing the location and function of all experimentally found and validated binding sites, the *cis*-Lexicon. The *cis*-Browser, a powerful visual environment for integrating many types of known genomic information and experimental data, together with the *cis*-Lexicon make up the core components of the CYRENE project: a software suite, associated tools, and set of resources for regulatory genomics. Only by mining a large, high-quality collection of functional binding sites can we uncover the missing clues into what makes sites functional.

We have come across many fundamental issues as the CYRENE project has developed. We crystallized three of these under the names the *Gene Naming Problem*, the *Consensus Sequence Bottleneck Problem*, and the *Logic Function Inference Problem*.

2 A Case Study

A gene with one of the best-understood *cis*-regulatory regions is *endo16* of the sea urchin *Strongylocentrotus purpuratus* [6]. A 2.3 kb sequence directly upstream of the transcription start site has been shown to contain binding sites for many transcription factors, clustered into six *cis*-regulatory modules [26], as seen in Fig. 1. Each module has its own independent function: Modules A and B drive expression in different periods of development, Modules DC, E, and F cause repression in certain regions of the embryo, and Module G is a general expression booster at all times. Each module has a similar organization: one or two binding sites for unique factors which do not bind anywhere else in the regulatory region, surrounded by several sites for factors which are found in other modules as well.

Module A has been described as a "hardwired logic processor," whose output in terms of gene expression depends entirely on the binding of its inputs. The factor which binds uniquely to Module A, Otx, is the "driver" of the module, meaning that the change in the output of the module over time is a result of the change in the concentration of Otx [24]. The other factors which bind to Module A, while absolutely necessary for correct behavior, are ubiquitous and typically do not play a role in its output changing. The precise structure of a regulatory region is visualized in a

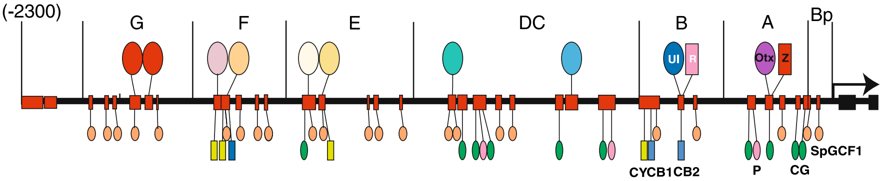


Fig. 1 The structure of *endo16*'s regulatory region (from [25])

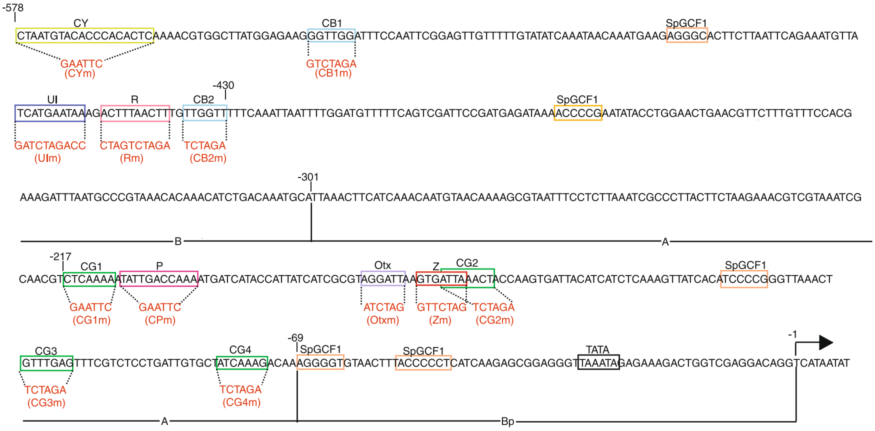

-578
CY
CB1
SpGCF1
CTAATGTACACCCACACTCAAAACGTGGCTTATGGAGAAGGGTTGGATTTCCAATTCGGAGTTGTTTTTGTATATCAAATAACAAATGAAGAGGGCACTTCTTAATTCAGAAATGTTA
GAATTC
(CYm)
GTCTAGA
(CB1m)
-430
UI
R
CB2
SpGCF1
TCATGAATAAAGACTTTAACTTTGTTGGTTTTTCAAATTAATTTTGGATGTTTTTCAGTCGATTCCGATGAGATAAAACCCCGAATATACCTGGAACTGAACGTTCTTTGTTTCCACG
GATCTAGACC
(UIm)
CTAGTCTAGA
(Rm)
TCTAGA
(CB2m)
-301
AAAGATTTAATGCCCGTAAACACAAACATCTGACAAATGCATTAAACTTCATCAAACAATGTAACAAAAGCGTAATTTCCTCTTAAATCGCCCTTACTTCTAAGAAACGTCGTAAATCG
B
A
-217
CG1
P
Otx
Z
CG2
SpGCF1
CAACGTCTCAAAAATATTGACCAAAATGATCATACCATTATCATCGCGTAGGATTAAGTGATTAAACTACCAAGTGATTACATCATCTCAAAGTTATCACATCCCCGGGTTAAACT
GAATTC
(CG1m)
GAATTC
(CPm)
ATCTAG
(Otxm)
GTTCTAG
(Zm)
TCTAGA
(CG2m)
CG3
CG4
-69
SpGCF1
SpGCF1
TATA
-1
GTTTGAGTTTCGTCTCCTGATTGTGCTATCAAAGACAAAGGGGTGTAACTTTACCCCCTCATCAAGAGCGGAGGGTTAAATAGAGAAAGACTGGTCGAGGACAGGTCATAATAT
TCTAGA
(CG3m)
TCTAGA
(CG4m)
A
Bp

Fig. 2 Quintessential diagram (from [25])

"quintessential diagram," as seen in Fig. 2. We call such an image the quintessential diagram because it displays at a glance the most fundamental knowledge we have of the organization of a gene's regulatory region. When searching for papers with information relevant to our database, we found that no useful paper lacks this diagram.

The sites within Module A carry out a complex logic program. Sites CG_1 and P (highlighted green in Fig. 3) communicate the effects of Module B to the transcription apparatus—if either of them is disabled (by mutating the sequence), the gene expression is as if Module B were not present at all. Site Z (highlighted red) communicates the effects of Modules DC, E, and F. Sites CG_1, CG_2, and CG_3 (highlighted blue) together boost the final output of the module. If even one of them is disabled, then the boost is completely removed.

Module B was shown to have a similar complexity [25]. In fact, all *cis*-regulatory modules execute information processing of their inputs, and the sum total of the computations performed by the regulatory regions of individual genes, when considered within the complex network established by the dependencies among genes, forms a "genomic computer" [13].

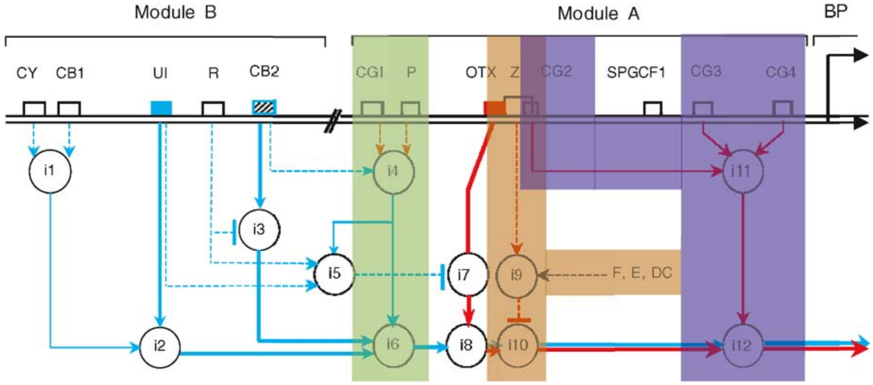


Fig. 3 Computational logic model for Modules A and B of *endo16* (from [25])

3 The Gene Naming Problem

"It all comes, I believe, from trying to define the indefinable."
– Charles Darwin [29]

"Scientists would rather share a toothbrush than use the same name for the same gene."
– Anonymous

As one tries to compile information from the literature into a database for automatic analysis, one of the issues that immediately crops up is that of naming. If information was inserted verbatim, then in order to find something one would need to use the exact terminology of the author. It would also be impossible to bring related information together automatically, unless large "translation" tables were compiled manually, which would suffer from problems with both completeness and precision. Completeness, because the addition of a new term in the database would require a new entry in the translation table, which could easily be left out by accident. Precision, since many terms are ambiguous when taken out of context even if they were completely obvious in the original setting. Errors in completeness lead to missed relationships, while errors in precision lead to spurious ones. Therefore, it is clear that a standard set of terms must be developed that all of the descriptions given in the literature can be translated to at the time the information is added to the database. This does lead to some loss of information, as the translation will always be imperfect, but the set of terms can always be expanded if systematic problems are noticed. A prime example of such a controlled vocabulary is the Gene Ontology, which describes gene and gene product attributes [2]. In the course of developing the *cis*-Lexicon, we composed standard terms for describing regulatory function, such as activation, repression, and *cis*-regulatory module linkage, called the *Cis*-Regulatory Ontology. We are in the process of defining a similar vocabulary for describing the functional interactions among transcription factor binding sites.

In all of the above cases, the translation of terms to a standard vocabulary is relatively straightforward because the properties described are clearly distinct and understood, regardless of the terminology—if one paper says a transcription factor

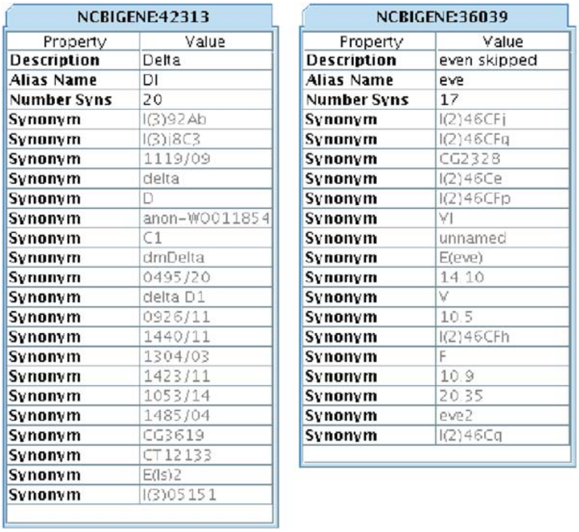


Fig. 4 Synonyms for *dl* and *eve*

NCBIGENE:42313

Property	Value
Description	Delta
Alias Name	Dl
Number Syns	20
Synonym	l(3)92Ab
Synonym	l(3)i8C3
Synonym	1119/09
Synonym	delta
Synonym	D
Synonym	anon-WO011854
Synonym	C1
Synonym	dmDelta
Synonym	0495/20
Synonym	delta D1
Synonym	0926/11
Synonym	1440/11
Synonym	1304/03
Synonym	1423/11
Synonym	1053/14
Synonym	1485/04
Synonym	CG3619
Synonym	CT12133
Synonym	E(ls)2
Synonym	l(3)05151

NCBIGENE:36039

Property	Value
Description	even skipped
Alias Name	eve
Number Syns	17
Synonym	l(2)46CFi
Synonym	l(2)46CFq
Synonym	CG2328
Synonym	l(2)46Ce
Synonym	l(2)46CFp
Synonym	VI
Synonym	unnamed
Synonym	E(eve)
Synonym	14.10
Synonym	V
Synonym	10.5
Synonym	l(2)46CFh
Synonym	F
Synonym	10.9
Synonym	20.35
Synonym	eve2
Synonym	l(2)46Cq

"increases expression" and another says a factor "boosts expression," it is very clear that both phrases refer to our canonical term "activation." A much more difficult problem is the wide variety of names given to a single gene in the literature. A peek into GenBank for a well-studied gene often yields more than 15 aliases; see Fig. 4 to see the synonyms for *dl* and *eve* of *D. melanogaster* displayed in the *cis*-Browser. To attempt to use tables of gene synonyms to determine which gene is really "meant" by a name can only lead to the troubles outlined in our discussion of translation tables earlier.

The *Gene Naming Problem* arises from the many techniques biologists use to study genes: examining a gene from the phenotype of a mutation, linkage map, or similarity to other known genes all yield different views, and consequently different names. Given the genomic sequence, the obvious unifier would be the gene's locus, but this is not always known. In addition, even the definition of a gene and its locus is in flux [11].

When a gene is named according to its similarity with other known genes, different conventions lead to similar but separate names, such as *oct-3* versus *oct3*, which can both be found in the literature. One method which is less widespread is to prefix a homology with an indicator of the new species, such as *spgcm* being a homology of *gcm* in *D. melanogaster*. This method can be impaired by the difficulty in determining what qualifies as a new species—at least 26 definitions have been published [29].

Occasionally, different transcription factors can bind to the exact same sequence. This is especially likely for factors in the same family which share an evolutionary history—the DNA binding motif may not have diverged much between them. In this case, when a sequence is found to be a functional site, for example, by a gel shift assay or perturbation analysis, it is not clear what factor is actually binding there. It's also possible for two putative binding sites to overlap, where it is not clear whether only one or both play a role in gene regulation. This can play havoc in determining the function of individual sites.

When biologists attempt to determine which factor is binding to a site by finding the molecular weights of proteins isolated by gel shifts or other similar techniques, often several weights are found. This can be caused by measuring different subunits of a single protein, alternative splicings of a single gene, different translation products from a single mRNA molecule, or even completely distinct factors. For example, when the structure of *endo16* was first mapped, 8 of the 14 inputs tentatively identified were found with more than one molecular weight [26]. Only by cloning the cDNAs encoding the factors can it be determined whether the multiple weights are from the same gene or not. The molecular weight can be thought of as a type of hash code for a protein: its value depends on the protein's sequence, but contains less information. If two proteins have a slightly different weight, they can still be from totally unrelated genes and their sequences can be very different. Analogously, even protein products from the same gene can have very different weights, if they differ in terms of splicing (entire exons added or subtracted which might not affect the final function a great deal). Biologists depend, however, on the assumption that proteins from different genes will have different weights. This may be true with very high probability, but it cannot be universally true, especially since our methods of measurement are always imperfect. This is similar to how unique identifiers are created and used in computer science: IDs are given by random number generators from a large enough range (e.g., 64-bit numbers) that the probability of a clash is almost zero.

4 The Consensus Sequence Bottleneck Problem

> *"[E]ssentially all predicted transcription-factor (TF) binding sites that are generated with models for the binding of individual TFs will have no functional role."*
> – Wasserman and Sandelin [23]

Given the wide variety of sequences that a transcription factor binds to, there is no straightforward yet accurate model. The earliest model, consensus sequence, while still preferred by most biologists, suffers from being forced to choose either selectivity (matching only known site sequences) or sensitivity (matching all known site sequences) [21]. As almost any pair of binding sites for a single factor differ in at least one base (see [19] for a study where out of seven binding sites for the same factor, only two sites have identical sequence and only three match the known consensus in all positions), the consensus sequence model allows for differences in two ways: (1) permitting a range of bases in certain positions through symbols such as Y (allowing C, T, or U); and (2) permitting a set number of global mismatches, such as allowing one or two bases which do not match the consensus at all. Both methods greatly increase the number of hits to random sequence: replacing an A with an R or V increases the probability of a match by chance by a factor of 2 or 3, respectively. Allowing for 1 or 2 mismatches anywhere in a consensus sequence increases it by a factor of up to 4 or 16.

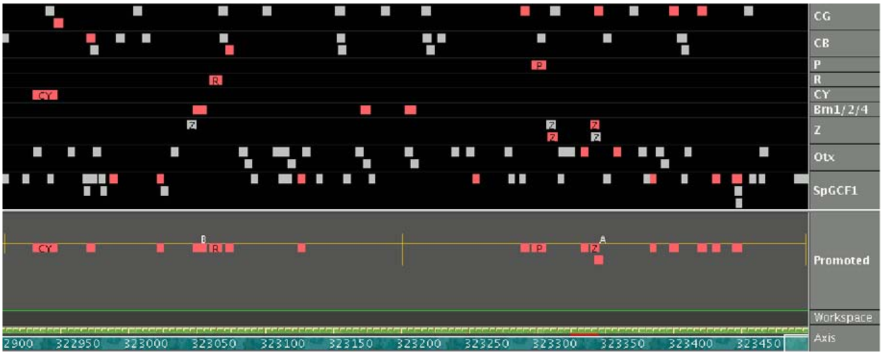


Fig. 5 Matches for known binding site sequence in *endo16*

Imagining the set of all DNA sequences of a single length to be a coordinate space, these two methods of allowing for differences act to create a bubble whose contents are the sequences matched by the model. The more differences allowed, the larger the bubble. The assumption is that since the set of sequences bound by a transcription factor are evidently similar in some unclear way, they should be clustered in this coordinate space and we should be able to find a "bubble" which contains this cluster tightly—it should contain all the sequences in the set, and none outside it. In practice, however, this is not the case. Stormo showed that given just 6 sites in *E. coli*, in order to match all of them, the "consensus" sequence must be so vague that a match is found by chance every 30 bp [21]. It is clear that at such levels of generality, the sets accepted by consensus sequence models begin to overlap as well—the distinction between binding sites for individual factors effectively disappears.

To demonstrate the futility of searching for individual sites using known binding site sequences, we examined the gene *endo16* from *S. purpuratus*. Two *cis*-regulatory modules of this gene have been studied, yielding 13 unique transcription factor inputs binding to 17 sites [24, 25]. We searched for additional sequences which look like binding sites for these same 13 factors within the two modules by searching for sequences like those we have recorded, except allowing for one single mismatch. For one of the inputs, *otx*, we knew of 4 other sites in other genes (*blimp1/krox* [16] and *otx* itself [28]), and we included those in our search, yielding 3 unique site sequences total for *otx*. For another input, *brn1/2/4*, we knew of one other site in *blimp1/krox* [16], so we included its sequence as well. The result of our search as visualized in the *cis*-Browser can be seen in Fig. 5. Exact matches are highlighted in red, and matches with one mismatch are drawn in gray.

The second major model of transcription factor binding sites is the position weight matrix (PWM). By incorporating not only the bases known to appear at each position but also their probabilities of occurrence, a much more precise model is made. While the independence/additivity assumptions are imperfect, they are a good approximation of reality, especially for the simplicity of the model—while there are a few notable exceptions, most factors are described relatively well by a PWM [3]. The logarithm of the observed base frequencies has been shown to be proportional to the binding energy contribution of the bases [4], so there is clear biological significance to using these values as the weights of the matrix. But a matrix of coefficients is not sufficient to predict binding sites—there is still the question of the best cut-off score. Unlike a consensus sequence, a PWM assigns a *score* to every sequence, and a cut-off must be chosen as the minimum score to expect a sequence to be a functional binding site. Even with a cut-off chosen to best match the experimentally known sites, it is unclear how "good" a site is if it is barely over the limit, or how nonfunctional a sequence is if it is just under the cut-off. As true binding sites are not merely "on" or "off," but have an effect whose degree may depend on the strength of the bind, the concept of a cut-off may be incorrect.

4.1 Motif Finding

None of the existing methods of representing a binding site can predict which sites are functional. Unlike restriction enzymes which have an effect by themselves, transcription factors only work by affecting the transcription machinery. Some factors do this directly, while others only communicate via intermediaries. Even a short sequence will contain what looks like many sites for many different transcription factors, and it is difficult to determine which actually determine gene expression.

One method to bypass this problem is to look at a set of genes that appear to be coregulated, i.e., they are expressed at the same time in the same location. It is very likely that the same transcription factor regulates these genes, either directly or indirectly. Therefore, many of the promoter regions of these genes should contain a binding site for that factor. By simply searching for a short sequence which is found to be overrepresented (i.e., more common than expected by chance), in principle we should be able to find such a binding site.

Unfortunately, the binding sites will probably not be identical. Some type of tolerance for mismatches must be added to the search algorithm, which complicates things considerably (otherwise a simple count of the number of occurrences of, e.g., every 8-mer would suffice). Some algorithms model the motif they are looking for combinatorially as a consensus string with a maximum number of mismatches [18], while others use a probabilistic or information-theoretic framework [14].

Without being able to discern de novo the regulatory regions of genes, we know that they should be conserved between closely-related species. Like the protein-coding sequence, regulatory sequence has a functional purpose and most mutations to it will cause harm to an organism. Therefore, few offspring who have any changes to the regulatory sequence will survive, in contrast to those who have changes to sequence outside the regulatory and coding regions, which should have no difficulty. Over generations, while a few minor changes occur within functional regions, large changes will accumulate in the rest. By examining the sequence of species at the right evolutionary distance, we should see clear conservation only where the sequence has a specific function. We can exclude the protein-coding sequence from our analysis, either by using predicted gene models (e.g., [10]) or by transcriptome analysis (e.g., [20]), and only look at the conserved patches of unknown function, which are likely to contain regulatory sequence (see, for example, [1, 17, 19, 27]). For the highest accuracy, several species can be compared simultaneously [5].

4.2 Evaluations of Motif-Finding

Existing motif algorithms perform reasonably well for yeast, but not for more complex organisms [7]. Several evaluations of the many proposed methods have been attempted, but the use of real genomic promotor sequences is hampered by the simple fact that "no one knows the complete 'correct' answer" [15, 22]. For an overview of the algorithms and the models they are based on, see [7].

5 The Logic Function Inference Problem

> *"Development of Western Science is based on two great achievements—the invention of the formal logical system (in Euclidean geometry) by the Greek philosophers, and the discovery of the possibility to find out causal relationships by systematic experiment (during the Renaissance)."*
>
> – Albert Einstein

The observed expression of a gene in space and time is not determined by a single transcription factor binding site, but by a collection of sites whose effects are combined in nontrivial ways. Istrail and Davidson catalogued the known site interactions into 4 categories: transcriptional activation operators, BTA control operators, combinatorial logic operators, and external control operators [12]. To fully understand the regulation of any gene, we must (1) identify all binding sites, (2) understand the function of each site, and (3) understand the rules for combining their functions to infer the overall *cis*-regulatory output [8].

As discussed above, step 1 alone is an especially difficult problem, since knowledge of transcription factors is largely biased toward one category: the "drivers." The drivers are those transcription factors whose expression varies in space and time, and thus determine the expression of the genes they regulate. Other factors are more or less ubiquitous—they do not vary, but instead are always present, which makes their presence harder to detect. Only thorough studies such as perturbation analysis can expose them.

Perturbation analysis is currently the only method for determining the functional interactions among modules and sites. Biologists take the known regulatory sequence of a gene and modify it to determine the function of each part. When several modules are known to exist, a high level understanding of the function of each module and how they interact together can be found by observing the expression caused by different subsets of the full array of modules. For example, to discover the function of the modules of *endo16*, Yuh and Davidson tried each of the six modules individually and in many different combinations (A, B, DC, E, F, FA, EA, DCA, FB, EB, DCB, etc.) [26].

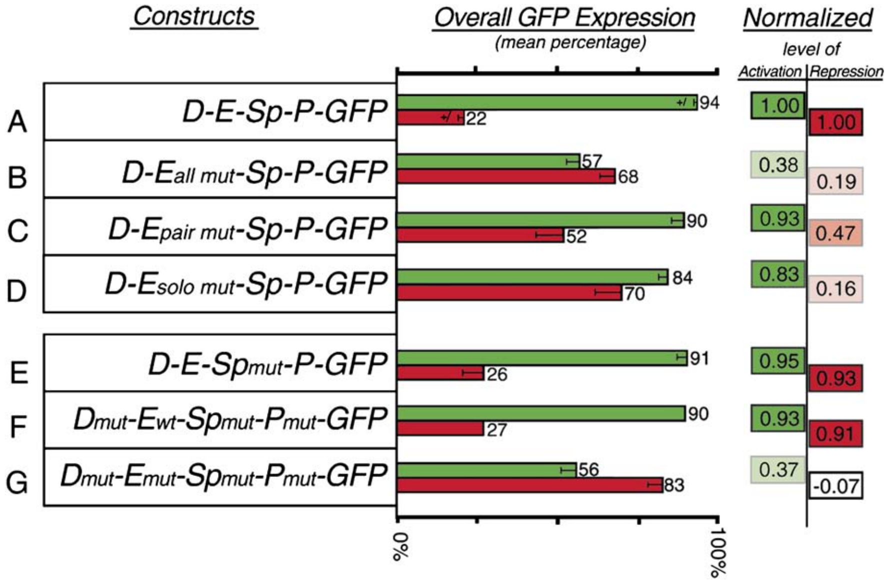


Fig. 6 Quintessential graph (from [19])

When putative sites are identified, biologists can tease out the complex hidden logic underlying even the simplest regulatory region by mutating both individual sites and groups of sites, each in independent experiments. This "regulatory archaeology" is performed one experiment ("dig") at a time—after each experiment, possible models are postulated, and the next perturbation is chosen based on the results. When the *S. purpuratus* gene *spgcm* was studied by Ransick and Davidson [19], they identified three putative binding sites for Su(H) to investigate. They recognized that two of the sites were arranged in a manner similar to a pattern familiar from studies in *Drosophila* and mouse, so they treated the two as a single feature. To determine the precise function of the sites, they observed the expression resulting from mutating all three, only the pair, and only the third single site. Even with only two elements to examine, they found a nontrivial logic behind their function: having either element yielded nearly the full correct activation, while one element in particular (the solo site, not the pair) was clearly necessary for correct repression (mutating the pair resulted in impaired repression but did not remove it entirely).

There are a two obvious strategies for choosing the next construct when in the midst of a series of experiments: pick the perturbation whose results can rule out the greatest number of putative models (i.e., binary search); or pick the perturbation whose expected results would confirm the experimenter's current hunch (i.e., an expert heuristic). The better the algorithm, the fewer the experiments that need to be performed to uncover the regulatory logic—and that means less time and less cost.

The intermediate results of perturbation analysis is visualized as a "quintessential graph": with one row for each reporter construct tested, bars are drawn to show the expression and/or repression of each construct relative to the control, which contains the full regulatory region sequence. See Fig. 6 for one example.

It can be difficult even to test whether a model is correct or not—the measured output, gene expression, is not a simple value. It varies from organism to organism, and it cannot be characterized as a single number, but at best as a distribution with variance. This variance can cause the ranges of expression for perturbation constructs to overlap even when they are significantly different. More experiments would yield tighter distributions, but the cost can sometimes not be afforded. At times, biologists must choose the next construct to test and move on without achieving *statistically* significant results, although the results they achieved might have been significant to them in view of their domain knowledge.

6 Conclusion

The concept of *cis*-regulatory information abstracts the wealth of types of data required for designing algorithms for computationally predicting the most likely organization of a piece of regulatory DNA into *cis*-regulatory modules. In this paper, we presented three problems that highlight the computational difficulties encountered in extracting such information. Two other sources of *cis*-regulatory information of fundamental importance are: (1) gene regulatory networks architecture (e.g., the endomesoderm network [9]), and (2) genomic data from species at the right evolutionary distance that preserve only the *cis*-regulatory modules (e.g., *S. purpuratus* and *L. variegatus*). The challenge of extracting *cis*-regulatory information is much amplified by the relatively scarce data sets and the depth of the analysis required to unveil such rich information sources.

Acknowledgements We thank Eric Davidson for many inspiring discussions and other contributions to this project. This research was supported by National Science Foundation Award 0645955.

References

1. Amore G, Davidson EH (2006) Cis-regulatory control of cyclophilin, a member of the ets-dri skeletogenic gene battery in the sea urchin embryo. Dev Biol 293(2):555–64
2. Ashburner M, Ball CA, Blake JA, Botstein D, Butler H, Cherry JM, Davis AP, Dolinski K, Dwight SS, Eppig JT et al (2000) Gene ontology: tool for the unification of biology. The gene ontology consortium. Nat Genet 25(1):25–29
3. Benos PV, Bulyk ML, Stormo GD (2002) Additivity in protein–DNA interactions: how good an approximation is it?. Nucleic Acids Res 30(20):4442–4451
4. Berg OG, von Hippel PH (1987) Selection of DNA binding sites by regulatory proteins. Statistical-mechanical theory and application to operators and promoters. J Mol Biol 193(4):723–750
5. Blanchette M, Tompa M (2002) Discovery of regulatory elements by a computational method for phylogenetic footprinting. Genome Res 12(5):739
6. Sodergren E, Weinstock GM, Davidson EH, Cameron RA, Gibbs RA, Angerer RC, Angerer LM, Arnone MI, Burgess DR et al (Sea Urchin Genome Sequencing Consortium) (2006) The genome of the sea urchin strongylocentrotus purpuratus. Science 314(5801):941–952

7. Das MK, Dai HK (2007) A survey of DNA motif finding algorithms. Feedback
8. Davidson EH (2006) The regulatory genome: gene regulatory networks in development and evolution. Academic Press, New York
9. Davidson EH, Rast JP, Oliveri P, Ransick A, Calestani C, Yuh CH, Minokawa T, Amore G, Hinman V, Arenas-Mena C et al (2002) A genomic regulatory network for development. Science 295(5560):1678, 1669
10. Elsik C, Mackey A, Reese J, Milshina N, Roos D, Weinstock G (2007) Creating a honey bee consensus gene set. Genome Biol 8(1):R13
11. Gerstein MB, Bruce C, Rozowsky JS, Zheng D, Du J, Korbel JO, Emanuelsson O, Zhang ZD, Weissman S, Snyder M (2007) What is a gene, post-encode? History and updated definition. Genome Res 17(6):669–681
12. Istrail S, Davidson EH (2005) Gene regulatory networks special feature: logic functions of the genomic cis-regulatory code. Proc Natl Acad Sci 102(14):4954
13. Istrail S, Ben-Tabou De-Leon S, Davidson EH (2007) The regulatory genome and the computer. Dev Biol 310(2):187–195
14. Lawrence CE, Altschul SF, Boguski MS, Liu JS, Neuwald AF, Wootton JC (1993) Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. Science 262(5131):208–214
15. Li N, Tompa M (2006) Analysis of computational approaches for motif discovery. Algorithms Mol Biol 1(8)
16. Livi CB, Davidson EH (2007) Regulation of spblimp1/krox1a, an alternatively transcribed isoform expressed in midgut and hindgut of the sea urchin gastrula. Gene Expr Patterns 7 (1–2):1–7
17. Minokawa T, Wikramanayake AH, Davidson EH (2005) Cis-regulatory inputs of the wnt8 gene in the sea urchin endomesoderm network. Dev Biology 288(2):545–558
18. Pevzner PA, Sze SH (2000) Combinatorial approaches to finding subtle signals in DNA sequences. In: Proceedings of the eighth international conference on intelligent systems for molecular biology, vol 8, pp 269–278
19. Ransick A, Davidson EH (2006) Cis-regulatory processing of Notch signaling input to the sea urchin glial cells missing gene during mesoderm specification. Dev Biol 297(2):587–602
20. Samanta MP, Tongprasit W, Istrail S, Cameron RA, Tu Q, Davidson EH, Stolc V (2006) The transcriptome of the sea urchin embryo. Science 314(5801):960–962
21. Stormo GD (2000) DNA binding sites: representation and discovery. Bioinformatics 16(1):16–23
22. Tompa M, Li N, Bailey TL, Church GM, De Moor B, Eskin E, Favorov AV, Frith MC, Fu Y, Kent WJ et al (2005) Assessing computational tools for the discovery of transcription factor binding sites. Nat Biotechnol 23:137–144
23. Wasserman WW, Sandelin A (2004) Applied bioinformatics for the identification of regulatory elements. Nat Rev Genet 5(4):276–287
24. Yuh CH, Bolouri H, Davidson EH (1998) Genomic cis-regulatory logic: experimental and computational analysis of a sea urchin gene. Science 279(5358):1896–1902
25. Yuh CH, Bolouri H, Davidson EH (2001) Cis-regulatory logic in the endo16 gene: switching from a specification to a differentiation mode of control. Development 128(5):617–629
26. Yuh CH, Davidson EH (1996) Modular cis-regulatory organization of endo16, a gut-specific gene of the sea urchin embryo. Development 122(4):1069–1082
27. Yuh C-H, Titus Brown C, Livi CB, Rowen L, Clarke PJC, Davidson EH (2002) Patchy interspecific sequence similarities efficiently identify positive cis-regulatory elements in the sea urchin. Dev Biol 246(1):148–161
28. Yuh C-H, Dorman ER, Howard ML, Davidson EH (2004) An otx cis-regulatory module: a key node in the sea urchin endomesoderm gene regulatory network. Dev Biol 269(2):536–551
29. Zimmer C (2008) What is a specie? Sci Am Mag 298(6):72–79