Methods in Molecular Biology 1244

Springer Protocols

Mario Andrea Marchisio *Editor*

Computational Methods in Synthetic Biology

Humana Press

METHODS IN MOLECULAR BIOLOGY

Series Editor
John M. Walker
School of Life Sciences
University of Hertfordshire
Hatfield, Hertfordshire, AL10 9AB, UK

Computational Methods in Synthetic Biology

Edited by

Mario Andrea Marchisio

School of Life Science and Technology, Harbin Institute of Technology, Harbin, China

Humana Press

Editor
Mario Andrea Marchisio
School of Life Science and Technology
Harbin Institute of Technology
Harbin, China

ISSN 1064-3745 ISSN 1940-6029 (electronic)
ISBN 978-1-4939-1877-5 ISBN 978-1-4939-1878-2 (eBook)
DOI 10.1007/978-1-4939-1878-2
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2014954440

Printed on acid-free paper

Humana Press is a brand of Springer
Springer is part of Springer Science+Business Media (www.springer.com)

Preface

In every engineering discipline, the construction of a complex artifact is driven by computational design and simulations. Electronics, for instance, deals with circuits made of components that obey the laws of electromagnetism. These well-established physical foundations allow the derivation of models, which predict circuit behavior faithfully, and permit the development of software for circuit's computer-aided design (CAD).

Giving precise mathematical pictures of biological systems is a challenging task. They are often characterized by a high number of interacting species and a non-negligible intrinsic noise (stochastic effects) that causes the failure of deterministic representations. However, in order to become an engineering science, Synthetic Biology needs to lie on a solid theoretical ground.

Ideally, well-characterized basic biological components (Standard Parts) should be retrieved from shared databases and wired together, into gene circuits, on the computer screen. Computational analysis and simulations would, then, lead to improvements on circuit's structure and consequent performance optimization. Finally, circuit's schemes would be sent to a robot for their actual implementation.

Following this path, the book has been organized into four parts. The first one provides an overview of the computational methods and algorithms for the design of bio-components, such as DNA Parts, RNA devices, and proteins that act as signal carriers inside a synthetic network. The second section gives an insight into CAD programs, languages, and methods for gene circuit in silico implementation, whereas the third section presents analysis techniques that are adopted to refine circuit layouts. The last section is devoted to distributed systems—aimed at fostering Part standardization—and DNA assembly protocols for automatic gene circuit wet-lab realization.

Overall, this book gives a complete coverage of the computational approaches currently used in Synthetic Biology and can be regarded as a guide to plan in silico the in vivo or in vitro construction of a variety of synthetic bio-circuits.

Harbin, China *Mario Andrea Marchisio*

Contents

Contributors

EBBE SLOTH ANDERSEN • *Department of Molecular Biology and Genetics, Center for DNA Nanotechnology, Interdisciplinary Nanoscience Center, Aarhus University, Aarhus, Denmark*

ERIC BATCHELOR • *Laboratory of Pathology, Center for Cancer Research, National Cancer Institute, National Institutes of Health, Bethesda, MD, USA*

GREGORY BATT • *INRIA Paris-Rocquencourt, Le Chesnay, France*

SAMUEL BOTTANI • *Laboratoire Matière et Systèmes Complexes, UMR 7057 CNRS and Université Paris Diderot, Paris, France*

PABLO CARBONELL • *Institute of Systems and Synthetic Biology (iSSB), CNRS; Université d'Evry val d'Essonne, Évry, France*

JAMES M. CAROTHERS • *Department of Chemical Engineering, University of Washington, Seattle, WA, USA; Department of Bioengineering, University of Washington, Seattle, WA, USA; Molecular Engineering and Sciences Institute, University of Washington, Seattle, WA, USA; Center for Synthetic Biology, University of Washington, Seattle, WA, USA*

GILLES CHARVIN • *Institut de Biologie Moléculaire et Cellulaire, Illkirch, France*

KEVIN CLANCY • *Synthetic Biology Unit, Life Technologies, Carlsbad, CA, USA*

MICHAEL T. COOLING • *Auckland Bioengineering Institute, University of Auckland, Auckland, New Zealand*

VINCENT DANOS • *School of Informatics, University of Edinburgh, Edinburgh, Scotland, UK*

THIERRY DELAVEAU • *Laboratoire de Génomique des Microorganismes, UMR 7238 CNRS and Université Pierre et Marie Curie, Paris, France*

FRANÇOIS FAGES • *INRIA Paris-Rocquencourt, Le Chesnay, France*

ELISA FRANCO • *Department of Mechanical Engineering, University of California at Riverside, Riverside, CA, USA*

KATE E. GALLOWAY • *Department of Stem Cell Biology and Regenerative Medicine, Keck School of Medicine, University of Southern California, Los Angeles, CA, USA*

PASCAL HERSEN • *Laboratoire Matière et Systèmes Complexes, UMR 7057 CNRS and Université Paris Diderot, Paris, France; The Mechanobiology Institute, National University of Singapore, Singapore, Singapore*

RICARDO HONORATO-ZIMMER • *School of Informatics, University of Edinburgh, Edinburgh, Scotland, UK*

ALFONSO JARAMILLO • *Institute of Systems and Synthetic Biology (iSSB), CNRS; Université d'Evry val d'Essonne, Évry, France; School of Life Sciences, University of Warwick, Coventry, England, UK*

YIANNIS N. KAZNESSIS • *Department of Chemical Engineering and Materials Science, University of Minnesota, Minneapolis, MN, USA*

KYUNG HYUK KIM • *Department of Bioengineering, University of Washington, Seattle, WA, USA*

THOMAS E. LANDRAIN • *Institute of Systems and Synthetic Biology (iSSB), CNRS; Université d'Evry val d'Essonne, Évry, France*

CURTIS MADSEN • *School of Computing, University of Utah, Salt Lake City, UT, USA*

MARIO ANDREA MARCHISIO • *School of Life Science and Technology, Harbin Institute of Technology, Harbin, P.R. China; D-BSSE, ETH Zurich, Basel, Switzerland*

AGNÈS MIERMONT • *Laboratoire Matière et Systèmes Complexes, UMR 7057 CNRS and Université Paris Diderot, Paris, France*
GOKSEL MISIRLI • *School of Computing Science, Newcastle University, Newcastle upon Tyne, England, UK*
CHRIS MYERS • *Department of Electrical and Computer Engineering, University of Utah, Salt Lake City, UT, USA*
ERNST OBERORTNER • *Department of Electrical and Computer Engineering, Boston University, Boston, MA, USA*
MICHAEL PEDERSEN • *Department of Plant Sciences, Cambridge University, Cambridge, England, UK; Biological Computation Group, Microsoft Research, Cambridge, England, UK*
MATTHEW POCOCK • *School of Computing Science, Newcastle University, Newcastle upon Tyne, England, UK*
JOSHUA R. PORTER • *Laboratory of Pathology, Center for Cancer Research, National Cancer Institute, National Institutes of Health, Bethesda, MD, USA*
JACQUELINE QUINN • *Bio Nano Programmable Matter, Autodesk Research, San Francisco, CA, USA*
OFIR RAZ • *Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel*
GUILLERMO RODRIGO • *Institute of Systems and Synthetic Biology (iSSB), CNRS; Université d'Evry val d'Essonne, Évry, France*
NICHOLAS ROEHNER • *Department of Bioengineering, University of Utah, Salt Lake City, UT, USA*
WILLIAM ROSTAIN • *Institute of Systems and Synthetic Biology (iSSB), CNRS; Université d'Evry val d'Essonne, Évry, France*
CASIM A. SARKAR • *Department of Biomedical Engineering, College of Science and Engineering, University of Minnesota, Minneapolis, MN, USA*
HERBERT M. SAURO • *Department of Bioengineering, University of Washington, Seattle, WA, USA*
DENIS SELNIHHIN • *Department of Molecular Biology and Genetics, Center for DNA Nanotechnology, Interdisciplinary Nanoscience Center, Aarhus University, Aarhus, Denmark*
NAJAF A. SHAH • *Genomics and Computational Biology Graduate Group, Perelman School of Medicine, University of Pennsylvania, Philadelphia, PA, USA; Applied Mathematics and Modeling, Informatics IT, Merck Research Labs, West Point, PA, USA*
PATRICK SMADBECK • *Department of Chemical Engineering and Materials Science, University of Minnesota, Minneapolis, MN, USA*
TIM THIMMAIAH • *Department of Chemical Engineering, University of Washington, Seattle, WA, USA; Department of Bioengineering, University of Washington, Seattle, WA, USA; Molecular Engineering and Sciences Institute, University of Washington, Seattle, WA, USA; Center for Synthetic Biology, University of Washington, Seattle, WA, USA*
TY THOMSON • *Plectix Biosystems, Somerville, MA, USA*
JEAN-YVES TROSSET • *SUP'Biotech, Villejuif, France*
JANNIS UHLENDORF • *INRIA Paris-Rocquencourt, Le Chesnay, France; Laboratoire Matière et Systèmes Complexes, UMR 7057 CNRS and Université Paris Diderot, Paris, France*

WILLIAM E. VOJE JR. • *Department of Chemical Engineering, University of Washington, Seattle, WA, USA; Department of Bioengineering, University of Washington, Seattle, WA, USA; Molecular Engineering and Sciences Institute, University of Washington, Seattle, WA, USA; Center for Synthetic Biology, University of Washington, Seattle, WA, USA*

JOHN WILSON-KANAMORI • *School of Informatics, University of Edinburgh, Edinburgh, Scotland, UK*

CHRIS WINSTEAD • *Department of Electrical and Computer Engineering, Utah State University, Logan, UT, USA*

TUVAL BEN YEHEZKEL • *Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel; SynVaccine, Tel-Aviv, Israel*

BOYAN YORDANOV • *Department of Plant Sciences, Cambridge University, Cambridge, England, UK; Biological Computation Group, Microsoft Research, Cambridge, England, UK*

TOMMY YU • *Auckland Bioengineering Institute, University of Auckland, Auckland, New Zealand*

ZHEN ZHANG • *Department of Electrical and Computer Engineering, University of Utah, Salt Lake City, UT, USA*

Part I

Component Design

Chapter 1

Computational Protein Design Methods for Synthetic Biology

Pablo Carbonell and Jean-Yves Trosset

Abstract

Computational protein design, a process that searches for mutants with desired improved properties, plays a central role in the conception of many synthetic biology devices including biosensors, bioproduction, or regulation circuits. To that end, a rational workflow for computational protein design is described here consisting of (a) searching in the sequence, structure or chemical spaces for the desired function and associated protein templates; (b) finding the list of potential hot regions to mutate in the parent proteins; and (c) performing in silico screening of mutants with predicted improved properties.

Key words Synthetic biology, Protein design, Protein engineering, Biosensor

1 Introduction

Protein engineering is a process that searches for variants (mutants) of a parent protein showing improved properties. The process involves finding substitutions through engineering strategies, such as rational design (mutagenesis) or random selection (directed evolution), and evaluating the desired properties in the protein variants through screening and selection [1, 2]. Even though high-throughput screening in combination with directed evolution techniques could substantially increase the explored sequence space, the development of cost-effective procedures for hit identification remains still a challenging experimental task for protein engineering.

In order to increase the rate of success in protein engineering, computational protein design methods provide in silico protocols that combine physicochemical knowledge from biological databases with scoring functions from bioinformatics modeling to optimally select candidate solutions. Computational methods involve a lower level of human intervention, accelerating the discovery process and overcoming some of the practical limitations of the experimental screening and selection, such as detectability and stability of proteins. Wet lab engineering techniques such as directed evolution and high-throughput screening methods are, thus, complementary to computational protein design methods, as the latter can provide to the former functional mutant libraries with high diversity [3]. Therefore, it is not possible to think of a computational protein design that is fully detached of the protein engineering process, but rather computational design and experimental protein engineering are parts of an engineering cycle involving modeling, design, implementation, and validation, which typically needs to be performed in several rounds before achieving the desired performance.

In a similar fashion, the engineering approach for the design of biological systems is followed in the emerging discipline of synthetic biology to create modular parts such as artificial feedback loops [4], oscillators [5], and toggle switches [6] involving signaling, regulatory, and transcriptional networks. Notably, computational protein design is used on building these constructs as a molecular tool to engineer genetic networks [7], playing a central role in the development of novel approaches providing optimized solutions in applications of synthetic biology such as in metabolic engineering [8–10] and biosensors [11, 12]. Many promising emerging applications involve synthetic protein circuits, often embedded into synthetic scaffold proteins [13], producing a variety of novel cellular responses [14, 15]. For such innovative applications, computational protein design provides valuable methods amenable to the engineering toolbox of synthetic biology.

1.1 The Computational Protein Design Workflow

The strategy presented here for computational protein design is based on the principle of efficiently coding and reusing any protein information available about protein regions that are functionally linked to the desired activity in order to model it. To that end, the computational protein design workflow can be divided into three steps:

(a) Similarity search for proteins using either the sequence, the structure, the interacting partner, the ligand, or the reaction

(b) Finding the list of potential mutants that could improve the property to be optimized

(c) Combinatorial in silico screen of the mutants to select the optimal cocktail of mutations

The identification of functional regions or hot spots in the protein has been extensively studied in the literature (*see* **Note 1**) and several authors have proposed methods that perform appropriately in many cases. We will describe some of them here but others might as well be used and the reader is invited to check them in the literature [16–18]. It should be taken into account that no unique method would always provide the best hints about functional regions, being the best choice the method that works more appropriately for each particular case. We provide in this method paper advice about typical applications and limitations for some of the most popular methods.

Regarding the third goal, large-scale screening, it is strongly linked to available computational resources, an issue that has dramatically become more affordable with the advent of clustering computing. Therefore, even though several search reduction strategies are possible in virtual protein screening such as working with a reduced list of amino acids, we will assume here that computational resources are not a major limiting factor and that screening libraries comprising several hundreds of millions of mutants can be performed in a significant lower amount of time in comparison with wet-lab experimental procedures (for instance, in no more than 24 h) (*see* **Note 2**).

2 Materials

The following list provides a survey of the main tools that are used in this chapter at each step of the computational protein design:

1. Databases: Proteins: Uniprot, PROSITE, Pfam, PDB, GenBank; Metabolic: MetaCyc and KEGG; Enzymatic: databases such as ENZYME, IntEnz, BRENDA.
2. Protein conservation, homology and function site identification: BLAST, CD-HIT, T-Coffee, SWISS modeler, Consurf, EvolTrace, Concavity, Ghecom, Q-Site Finder.
3. Molecular visualization and structural analysis: Chimera, PyMol.
4. Statistical package *R* with machine learning libraries: kernlab, pls, randomForest, caret.
5. Chemical analysis: Openbabel, Stereo signature molecular descriptor (*see* **Note 3**).
6. FLO-QXP for combinatorial sampling of side chains in presence of ligands.
7. Enzyme design server: RosettaDesign.

3 Methods

3.1 A Motivating Example for Protein Design in Synthetic Biology

In this chapter we illustrate the various concepts and protocols of computational protein design using the xanthine phosphoribosyltransferase or XPRTase (2.4.2.22). This enzyme catalyzes the transfer of the phosphoribosyl group from 5′ribosyl phosphate (XMP or GMP) into xanthine or guanine, respectively. In our example, the goal is to increase the selectivity of the enzyme towards XMP.

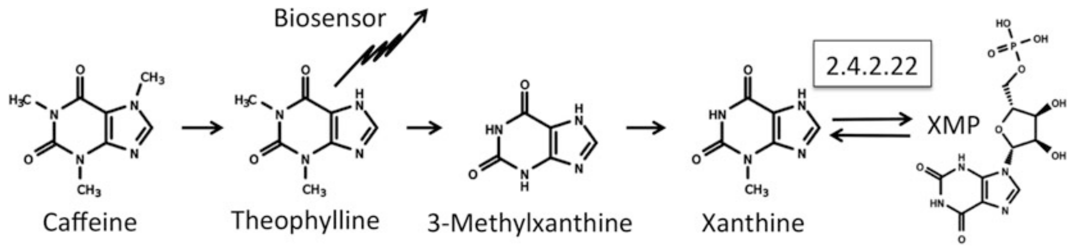


Fig. 1 Catabolism of caffeine in fungi takes place by theophylline route and is similar to the pathway operating in plants. Theophylline, a metabolite that can be used to develop biosensors, is further metabolized into xanthine, a purine base whose phosphoribosyl group is reversibly transferred from XMP by enzyme EC 2.4.2.22. Modulating the efficiency of that enzyme towards XMP, thus, could be used to tune the biosensor circuit

Such example illustrates the use of protein design in a potential synthetic biology application. Xanthine, one of the chemical entities catalyzed by XPRTases, belongs to the natural pathway of caffeine metabolism (Fig. 1) and it has been shown to be able to modulate theophylline [19], which is the first formed intermediate in this pathway. Interestingly enough, theophylline-dependent riboswitches for the conditional control of gene expression have been demonstrated in yeast and bacteria [20, 21], opening the possibility of developing biosensor circuits through the regulation of the presence or absence of theophylline.

As a template for the protein, we take 2FXV from the PDB database, which corresponds to the crystal structure of the *Bacillus subtilis* XPRTase (gene name BSU22070) in complex with GMP. Our goal, thus, will be to search for mutations in that protein increasing the affinity towards XMP through sequence and structure-based methods.

3.2 Similarity Search for Proteins

The goal here is to obtain information about the mechanisms related to the function of the target protein based on the existing data of related proteins. Depending on the type of property we want to optimize, we can relate the property with the protein descriptors that are either based on the sequence, the structure, the ligand, or the reaction in the case of an enzyme.

3.2.1 Sequence-Based Search

The minimal information of the protein is its sequence. If it does not inform on the involved mechanism, we can still relate a change of sequence with an observed change of enzymatic activity, binding free energy, temperature of folding, etc. Even in the absence of property data for the protein, amino acids conservation from sequence alignment can already guide us towards the possible mutations observed in nature. Comparing substrates of the related enzymes can also give a hint on the mutation that could optimize the property for our enzyme. In our example, the sequence alignment of xanthine and hypoxanthine phosphoribosyl transferases informs on the conserved difference of sequence between the two families. Relating this difference of sequence to the difference of substrate, xanthine and hypoxanthine, we can infer potential mutations that could favor the enzyme activity towards hypoxanthine-like substrate. Multi-sequence alignment within proteins of the same family is therefore a first source of information for selecting mutant candidates.

For our guided example, enzyme EC 2.4.2.22, we should first look into its sequence annotations in protein databases. Either a homology search based on BLAST or a similarity measure based on motif or pattern content will help us to identify closer sequences. For instance, ENZYME database from ExPASy provides multiple links to protein information regarding the sequences for EC 2.4.2.22. One of such sources of information is IntEnz, which provides the link to annotated sequences in UniProt for EC 2.4.2.22 (300 sequences), being reduced to 22 clusters under a 50 % filtering of redundancy (a tool like CD-HIT [22] can conveniently perform such task for a sequence list). PROSITE, in turn, provides the link to proteins containing the PDOC00096 pattern or signature (purine/pyrimidine phosphoribosyl transferases), which is common to multiple substrates of phosphoribosyltransferases such as adenine, hypoxanthine-guanine, orotate, amido, or xanthine-guanine (*see* **Note 4**). A multiple alignment can then provide information about residues where mutations will bring specificity towards each substrate. To illustrate here this method, we selected two sequences for each substrate from the list of non-redundant sequences containing the motif (total 259 sequences). After performing the alignment with T-Coffee [23], we observed two highly conserved regions: ALA42 to ARG80 and ASP116 to LYS155 (according to 2FXV numbering) followed by regions that are specific to each substrate. Figure 2 shows part of the alignment (region ILE139-ARG171) for the second conserved region followed by a region of high polymorphism. Interestingly, this region, as we will see in next sections, does not directly interact

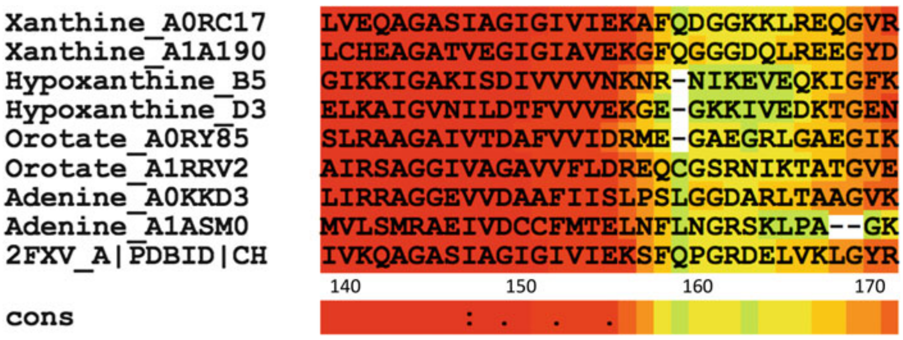

Xanthine_A0RC17 LVEQAGASIAGIGIVIEKAFQDGGKKLREQGVR
Xanthine_A1A190 LCHEAGATVEGIGIAVEKGFQGGGDQLREEGYD
Hypoxanthine_B5 GIKKIGAKISDIVVVVNKNR-NIKEVEQKIGFK
Hypoxanthine_D3 ELKAIGVNILDTFVVVEKGE-GKKIVEDKTGEN
Orotate_A0RY85 SLRAAGAIVTDAFVVIDRME-GAEGRLGAEGIK
Orotate_A1RRV2 AIRSAGGIVAGAVVFLDREQCGSRNIKTATGVE
Adenine_A0KKD3 LIRRAGGEVVDAAFIISLPSLGGDARLTAAGVK
Adenine_A1ASM0 MVLSMRAEIVDCCFMTELNFLNGRSKLPA--GK
2FXV_A|PDBID|CH IVKQAGASIAGIGIVIEKSFQPGRDELVKLGYR
140
150
160
170
cons
: . . .

Fig. 2 Conservation based on a multiple alignment for phosphoribosyltransferases accepting different substrates. The figure shows a conserved region and another that is highly variable depending on the substrate corresponding to sequence region ILE139 to ARG171

with the ligand. This fact shows how the sequence-based conservation approach can identify potential mutations that are not easily detected by a rational structure-based approach. More in general, conserved regions are typically related to some structural role, while regions showing polymorphism in a way concerted with the type of substrate might be related to positions determining substrate affinity.

3.2.2 Structure-Based Search

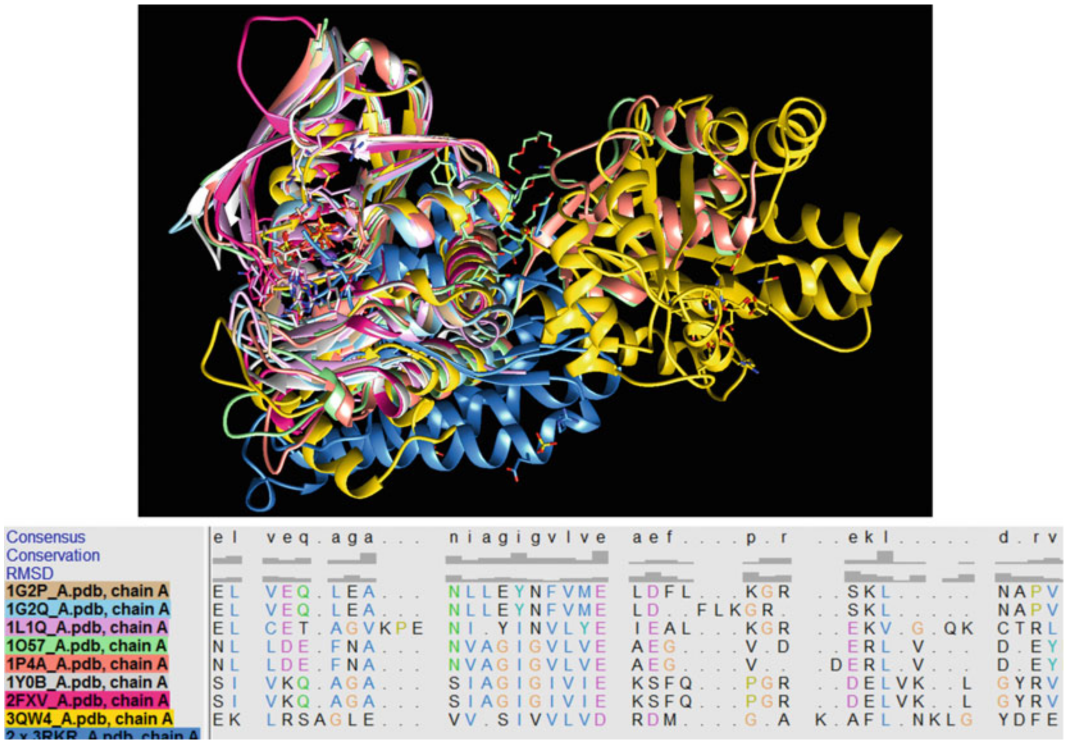
The second source of information is 3D protein structures. With the increasing number of protein structures in the Protein Data Bank (PDB), there is an increasing probability of finding a 3D protein structure for a given related protein sequence. As structure is more conserved than sequence, the multi-sequence alignment deduced from structure superposition is often more precise. Visual inspection of these polymorphisms around the region of interest (catalytic site or protein-protein binding interface or in the core of the protein) could provide valuable clues for possible mutants that could be further tested with energy-based methods (*see* section below). Note that the full 3D superposition of two proteins is only possible if there is enough structural similarity between the two proteins. Note also that full structure superposition is based on the structure similarity of Cα chains (*see* **Note 5**).

In our example, we use the 2FXV sequences to do a sequence search on the Protein Data Bank using BLAST. From the 29 hits, only nine display enough sequence covering (>50 %) for the overall structure similarity of the protein to be recognized by a 3D superposition program such as Chimera [24]. The protein structure superposition and the corresponding multi-sequence alignment are shown in Fig. 3.

If the full protein structure superposition could help at prioritizing mutants according to their position in structural elements, for example discriminating buried residues versus those that are solvent accessible or residues that are in close contact with the region of interest of the protein, this approach is however limited to proteins that have the same fold. Further similarity can be obtained if we focus the search on a specific region of the protein, e.g., the binding site.

3.2.3 Pharmacophore-Based Search

More specific functional similarities can be observed through a smaller scale superposition of a particular region of the protein, e.g., binding site or protein-protein interface. As structure is more conserved than sequence, functional residues are even more conserved in the structural alignment and structural similarity can therefore be observed on those important residues even for far-related proteins. These structural or functional protein interfamily similarities can be highlighted using a chemical pharmacophore description on the protein residues [25]. Local protein similarity can be observed even for proteins of different fold.

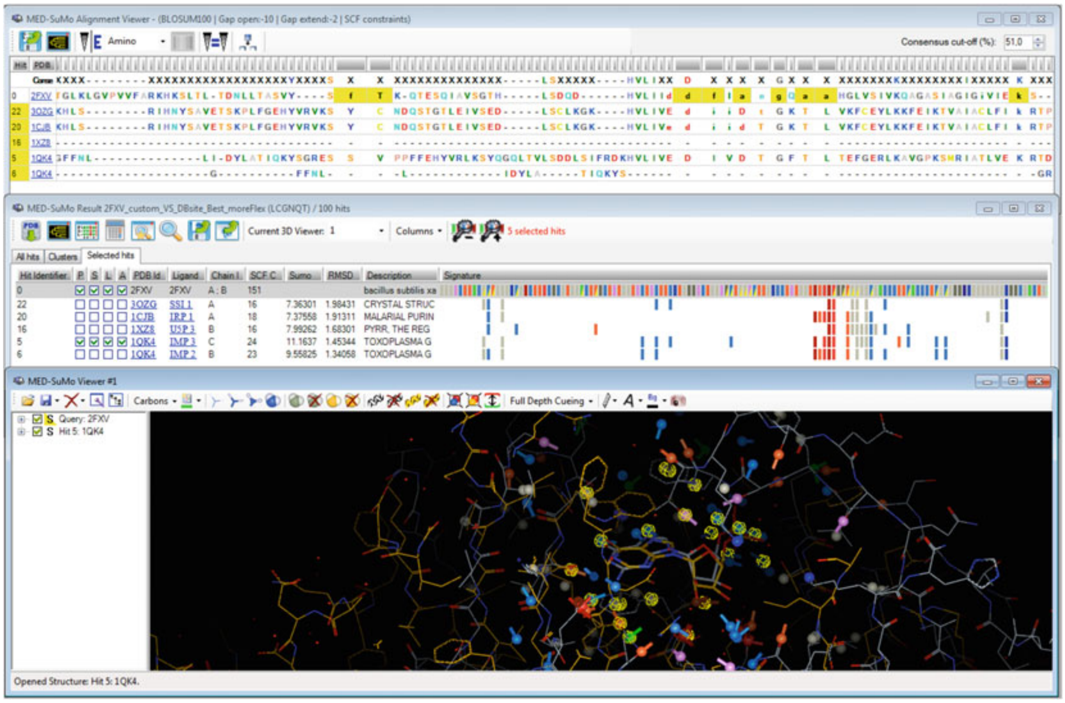


```
Consensus                e l  v e q . a g a . . .  n i a g i g v l v e  a e f . . . . p . r    . . e k l . . . . .  d . r v
Conservation
RMSD
1G2P_A.pdb, chain A      E L  V E Q . L E A . . .  N L L E Y N F V M E  L D F L . . . K G R    . . S K L . . . . .  N A P V
1G2Q_A.pdb, chain A      E L  V E Q . L E A . . .  N L L E Y N F V M E  L D . . F L K G R .    . . S K L . . . . .  N A P V
1L1Q_A.pdb, chain A      E L  C E T . A G V K P E  N I . Y I N V L Y E  I E A L . . . K G R    . . E K V . G . Q K  C T R L
1O57_A.pdb, chain A      N L  L D E . F N A . . .  N V A G I G V L V E  A E G . . . . V . D    . . E R L . V . . .  D . E Y
1P4A_A.pdb, chain A      N L  L D E . F N A . . .  N V A G I G V L V E  A E G . . . . V . .    . D E R L . V . . .  D . E Y
1Y0B_A.pdb, chain A      S I  V K Q . A G A . . .  S I A G I G I V I E  K S F Q . . . P G R    . . D E L V K . . L  G Y R V
2FXV_A.pdb, chain A      S I  V K Q . A G A . . .  S I A G I G I V I E  K S F Q . . . P G R    . . D E L V K . . L  G Y R V
3QW4_A.pdb, chain A      E K  L R S A G L E . . .  V V . S I V V L V D  R D M . . . . G . A    K . A F L . N K L G  Y D F E
2 x 3RKR_A.pdb, chain A
```

Fig. 3 3D alignment of XPRTases using Chimera. *Top panel*: Superposition of nine structures. *Bottom panel*: Close-up of the multi-sequence alignment based on the 3D superposition in the similar region of Fig. 1, i.e., the region 139I to 171R of 2FXV. The quality of the 3D superposition is higher in the regions of the protein that are well superimposed

Using our example, we query XPRTase (2FXV) over the protein data bank (PDB) using the MedSumo Software from MEDIT [25]. The hits from MedSumo query were then filtered to conserve protein with enough 3D similarity (Sumo score > 6.0 and RMSD less than 2.0 Å for distance differences between the pharmacophore features). The hits are then clustered according to their pharmacophore similarity and for each cluster a multiple sequence alignment can be calculated by using the pharmacophore superposition as constraints.

Figure 4 shows a selected cluster that includes enzymes of the same Phosphoribosyl transferase pfam family (PF00156) as our query. The pharmacophores of the query protein that are close to the ligand are marked in the structure (bottom panel), in the multi-alignment (top panel) and in the bar code of the query (middle panel of Fig. 4). This bar code is a color-code pharmacophore representation that can be used for protein structure alignment. The larger the number of aligned pharmacophore with the query, the higher the sumo score is. The nine pharmacophores that are on the residues PHE99, TYR100, ASP125, PHE126, ALA128, GLY130, ALA132, ALA133, and LYS156 of the query protein


MED-SuMo Alignment Viewer - (BLOSUM100 | Gap open:-10 | Gap extend:-2 | SCF constraints)

Consensus cut-off (%): 51.0

MED-SuMo Result 2FXV_custom_VS_DBsite_Best_moreFlex (LCGNQT) / 100 hits

Current 3D Viewer: 1 · Columns · 5 selected hits

All hits | Clusters | Selected hits

Hit Identifier	P	S	L	A	PDB Id	Ligand	Chain I.	SCF C.	Sumo	RMSD	Description
0	✓	✓	✓	✓	2FXV	2FXV	A : B	151			bacillus subtilis xa
22	☐	☐	☐	☐	3OZG	SS1 1	A	16	7.36301	1.98431	CRYSTAL STRUC
20	☐	☐	☐	☐	1CJB	IRP 1	A	18	7.37558	1.91311	MALARIAL PURIN
16	☐	☐	☐	☐	1XZ8	U5P 3	B	16	7.99262	1.68301	PYRR, THE REG
5	✓	✓	✓	✓	1QK4	IMP 3	C	24	11.1637	1.45344	TOXOPLASMA G
6	☐	☐	☐	☐	1QK4	IMP 2	B	23	9.55825	1.34058	TOXOPLASMA G

MED-SuMo Viewer #1

Carbons · Full Depth Cueing

Query: 2FXV
Hit 5: 1QK4

Opened Structure: Hit 5: 1QK4

Fig. 4 3D structural alignment of the pharmacophore of the binding region of a selected cluster of XPRTases related proteins using Med-Sumo server. *Bottom panel*: 3D superposition of 2FXV and 1QK4. The residue side chain pharmacophores highlighted in *yellow* belong to 2FXV. They are close to the guanine part of the ligand GMP. *Middle panel*: The pharmacophores of each protein of the cluster are represented in *color code*. The *marked triangles* in the color code of the query correspond to the marked pharmacophores in the *bottom panel*. *Top panel*: Sequence alignment using the pharmacophore superposition as constraints. *Marked* residues correspond to the highlighted pharmacophores of *bottom panel*. A position residue numbering is shown at the *top of the bottom panel*. Number 211 at the *left* corresponds to an empty position in 2FXV followed by ILE38 (position 212) and GLY39 (position 213)

2FXV are those that have direct interaction with the guanine part of GMP. Potential mutants can be deduced from top panel of Fig. 4 for the corresponding highlighted amino acids. The selection of mutants is focused on a specific part of the protein with a structure-based rational approach that is closely related to the energy-based approach described later in the text.

Another advantage of the pharmacophore based protein similarity is that it is not limited to proteins with the same fold. Triads of catalytic residues having the same position in space could still be observed for proteins that have very different fold as it is the case for example between trypsin and subtilisin [26].

3.2.4 Ligand-Based Search

Finally, related proteins can be searched through similarity of their interacting partners. In the case of enzymes, those partners correspond to the chemical entities involved in the reaction. By searching for similar reactants and/or products of two enzymatic reactions it is possible not only to capture enzymes with similar catalytic function but also to capture apparently unrelated proteins with similar sub-pockets. The similarity of sub-pockets corresponds to a similarity of fragment of the substrates. Study of amino acid polymorphisms in these substrate-binding sub-pockets can inform on possible mutations that could increase or decrease the interaction a particular part of the substrate (substructure or fragment).

In the ligand-based search, we mine protein databases for sequences associated with similar activities through the encoding of their associated ligands. In the case of an enzyme, we can go further and encode the entire reaction: substrate and product. Similarity in the chemical space relates to similar biochemical transformations. In the same way, this principle can be applied in the case of other type of activities: searching for similar protein-ligand interactions can be done by searching for similar chemical ligand and searching for similar protein-protein interactions by searching for proteins with similar peptide 3D or sequence motif.

Ligand or reaction similarity is based on chemical descriptors such as those used in graph similarity techniques. Many ligand or reaction similarity measures can be defined [27], although most of them are based on the Jacquard coefficient $J = A \cap B / A \cap B$. In our example, XPRTases (EC 2.4.2.22) are defined for the reaction shown in Fig. 1. We could measure, for instance, the similarity between xanthine, hypoxanthine, and guanine by counting the number of common fragments divided by the full fragment content of the substrates (*see* **Note 6**). We can also first interrogate the chemical space in databases in order to determine the reaction cluster containing similar reactions. Taking reactions fingerprints in MetaCyc, as defined by their stereo extended connectivity fingerprint (ECFP) of even diameters up to 6 [28] and computing reaction similarity, we have that from a total of 4,392 reactions, 3 have a similarity higher than 0.5 and 22 reactions show a similarity higher than 0.25. In KEGG, these values are in turn of 64 and 73 out of 6,747 reactions. These reactions are sorted in the database in decreasing order, and non-redundant sequences annotated for such reactions can be then selected as templates. Focusing on our example for XPRTases in MetaCyc, a cutoff similarity of 0.13 is needed in order to include in our positive set a minimum number of 100 sequences, which should be considered a good trade-off between sequence diversity (provided non-redundancy has been removed) and proximity to the target reaction.

3.3 Identification of Potential Mutants

The previous section aims at providing a multi-sequence alignment of all related proteins wherever the alignment comes from sequence information alone, 3D structure superposition, or similarity of the chemical reaction patterns. Selection of residues can be then done either by inspecting the sequence alignment alone or by introducing some additional information from databases such as Brenda, PubChem Bioassay, or PPI databases.

Conserved residues in the alignment usually relate to an important functional or scaffolding role within the protein. Polymorphism over these amino acid positions inform on the possible variation observed in nature. If a 3D structure of a related sequence exists for our investigated protein then, it is easy to assign a region of interest in the protein to the corresponding section of the amino acid alignment. We can focus our selection on mutants of a particular region of the protein: binding or catalytic site if property concerns the enzyme activity, or protein surface if property concerns the protein-protein interactions. On the other hand, some property like thermostability might not be related to a defined region of the protein. Single amino acid mutation can indeed have a large effect on the protein property even if such mutation is far from the region of interest. Such mutations correspond in general to charged residues as electrostatic energy can have long-range effect.

In order to identify potential regions containing candidate mutants, many bioinformatics tools are available allowing the identification of functional sites of the protein from sequence or structure. A selection of some of the most important tools is listed in Table 1. Some are based on conservation of structures or sequences. Many others look for pockets or cavities in the protein through the analysis of structural and geometric properties. For protein-protein interactions, several tools have been built based on patterns or profiles through machine learning. In interactions between proteins and small molecules, methods have been typically developed based on alignment and structural similarity between binding sites.

We illustrate in our example for the XPRTase (PDB 2FXV) how the use of multiple servers can help in order to decide through a consensus score the site of functional residues that can lead to candidate mutants. To that end, we computed the scores from two prediction servers in Table 1 based on residue conservation (Consurf and EvolutionaryTrace) and the scores of two prediction servers based on pocket identification (Concavity and Ghecom). Scores obtained from the prediction servers were normalized into z scores (*see* **Note 7**) in order to allow the comparison between these values. We first verified that each group of scores agreed in their predictions by looking at the correlation between the conservation scores, which was found of 0.91 and the correlation between the pocket scores, which was of 0.86. Secondly, correlation between each group of scores was found weak ($r < 0.5$), suggesting that the information provided by each group of predictors is independent of the other. Therefore, in order to obtain a consensus score that put at the top of predicted residues the ones that were ranked high for both types of scores, we computed the average of their z-scores, as

Table 1
Bioinformatics tools to identify functional sites of the protein through several methods such as conservation, identification of pockets and cavities, protein-protein interaction sites, or ligand-binding site

Server	Method	Technique
Consurf	Conservation	Identification of functional regions in proteins based on conservation of PDB structures [http://consurf.tau.ac.il]
SCORECONS		Score conservation in a multiple sequence alignment [http://www.ebi.ac.uk/thornton-srv/databases/cgi-bin/valdar/scorecons_server.pl]
EvolutionaryTrace		Ranks amino acids in protein sequence by their relative evolutionary importance [http://mordred.bioc.cam.ac.uk/~jiye/evoltrace/]
CAVER	Pocket finder	Searches for tunnels and channels in protein structures [http://caver.cz/]
fpocket		Detection of packets based on Voronoi tessellation [http://fpocket.sourceforge.net/]
GHECOM		Using mathematical morphology [http://strcomp.protein.osaka-u.ac.jp/ghecom/]
DoGSiteScorer		Based on size, shape, and chemical features [http://dogsite.zbh.uni-hamburg.de/]
PocketDepth		Geometry-based identification of functional sites [http://proline.physics.iisc.ernet.in/pocketdepth/]
SiteHound-web		Compute interactions between a chemical probe and a protein structure [http://scbx.mssm.edu/sitehound/sitehound-web/Input.html]
SplitPocket		Geometric approach [http://pocket.med.wayne.edu/patch/]
ConCavity		Combines evolutionary sequence conservation and 3D structure [http://compbio.cs.princeton.edu/concavity/]
CASTp		Searches for pockets and cavities [http://sts-fw.bioengr.uic.edu/castp/calculation.php]
Q-SiteFinder		Binds hydrophobic (CH3) probes to the protein, and finding clusters of probes with the most favorable binding energy [http://www.modelling.leeds.ac.uk/qsitefinder/]
Cons-PPISP	PPI site	Consensus PPI interaction predictor based on a neural network trained with sequence profiles and solvent accessibilities [http://pipe.scs.fsu.edu/ppisp.html]
Promate		A naive Bayesian method based on protein properties [http://bioinfo.weizmann.ac.il/promate/promate.html]
SPPIDER		A neural-network method that includes predicted solvent accessibility as input [http://sppider.cchmc.org/]
Meta-PPISP		A meta web server that combines cons-PPISP, Promate, and PINUP through linear regression [http://pipe.scs.fsu.edu/meta-ppisp.html]
MED-SuMo	Binding site similarity	Pharmacophore-based superposition from PDB database [http://medit-pharma.com/index.php?page=med-sumo]
FINDSITE		Threading-based binding site prediction in a set of evolutionarily related proteins [http://cssb.biology.gatech.edu/findsite]
3DLigandSite		Homologous structures with bound ligands [http://www.sbg.bio.ic.ac.uk/~3dligandsite/]
ProBis		Detects similar protein-binding sites from a database of protein structures [http://probis.cmm.ki.si/]

(continued)

Table 1 (continued)

Server	Method	Technique
PredictProtein	Prediction suite	Integrates multiple predictors of protein structure and function [http://www.predictprotein.org/]
WHAT IF		Integrates multiple predictors of protein structure and function [http://swift.cmbi.ru.nl/whatif/]
DYNAMO	QQ/MM	Library in FORTRAN [http://www.chem.ac.ru/Chemistry/Soft/DYNAMO.en.html]
QSITE		Commercial software from Schrödinger [http://www.schrodinger.com/productpage/14/19/]
AMBER		Works with ROAR and Gaussian [http://nf.nci.org.au/~vvv900/monash/amber-gaussian/index.html]
CHARMM		CHARMM works with MOPAC, GAMESS-US, GAMESS-UK, CADPAC, DeFT [http://www.charmm.org/documentation/c33b2/qchem.html]
GROMACS		A package for performing molecular dynamics [http://www.gromacs.org]

shown in Table 2. Top residues (THR100, SER98, ALA132, ASP125, PHE126, LYS101, LYS81, SER60, PHE99, ASN27, ASP124, GLU58, SER59, ALA128, VAL96) are depicted in Fig. 5. They are located around the substrate pocket suggesting that candidate mutants for these residues may modulate substrate affinity, as desired.

3.4 Combinatorial Screening of Candidate Mutants

There are two ways of screening all the possible combinations of mutants in the functional positions selected in the previous section. The first one involves machine-learning approach, whereas the second is based on an energetics evaluation of the mutants.

3.4.1 Machine-Learning Approach

A machine-learning approach aims at learning the effect of mutants on the desired protein property using a supervised or unsupervised strategy. In the first one, external information is used to split a set of similar protein into "good" and "bad," those sequences that improve or not the chosen property. For example, Km on enzyme reaction can be searched for each sequence in the BRENDA database [29]. Sequences corresponding to low Km will be in the "Good" set and the others sequence in the "Bad" set. Then, a naïve Bayesian approach can be carried out by assigning to each amino acid of the sequence the value of the corresponding Km. A "Km score" for each position of the alignment can be calculated by averaging the score for each amino-acid in the alignment column. Columns with highest score correspond to potential hot spots.

Table 2
Top-ranked 15 residues of 2fxv based on a consensus score from Consurf, EvolutionaryTrace, Concavity, and Ghecom

Rank	Residue	Sequence	Score	Consurf	EvolTrace	Concavity	Ghecom
1	100	THR	2.87	0.95	1.24	4.22	5.07
2	98	SER	2.26	0.95	1.24	3.35	3.48
3	132	ALA	2.22	0.94	1.24	3.02	3.69
4	125	ASP	2.09	0.94	1.24	3.69	2.48
5	126	PHE	2.06	0.91	1.24	2.58	3.52
6	101	LYS	2.01	0.93	0.99	3.75	2.37
7	81	LYS	1.96	0.93	1.24	3.41	2.25
8	60	SER	1.92	0.90	1.24	2.90	2.66
9	99	PHE	1.86	0.48	0.94	2.68	3.34
10	27	ASN	1.78	0.89	1.04	2.14	3.05
11	124	ASP	1.76	0.94	1.24	2.97	1.90
12	58	GLU	1.74	0.93	1.24	3.58	1.19
13	59	SER	1.62	0.57	0.85	3.09	1.98
14	128	ALA	1.49	0.94	1.24	1.29	2.47
15	96	VAL	1.25	0.83	0.89	1.94	1.36

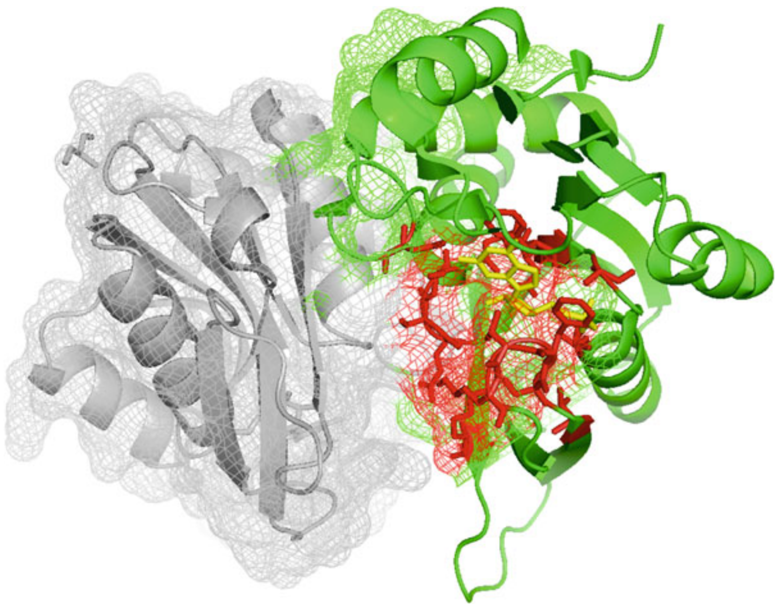


Fig. 5 Functional site highlighted in the chain A of 2FXV, as defined by the consensus score of Table 2. Residues appear located around the pocket where the substrate is positioned

If polymorphism is observed for those positions, we can consider these amino acid variants as potential mutant candidates to improve the protein property. In the unsupervised approach, we do not use external information. The “bad and “good” sets are created from different criteria (*see* **Note 8**).

Through the application of any of the numerous machine-learning techniques available, this training set is used in order to develop a predictive model (*see* **Note 9**). A training set can be either described by a vector of descriptors or by similarity matrices. In both cases, similarity information from sequences can be coupled with similarity of the interacting partner such as a ligand or a substrate. Multiple machine-learning techniques are available from open-source projects (*see* **Note 10**) and standard procedures to evaluate performance, such a cross-validation, need to be put into place in order to validate the model. In order to ensure convergence of the predictor, it might be necessary to limit the size of the training set.

In our example, sequences catalyzing reactions included in the positive set are removed from the database of sequences, performing for the rest of sequences a Monte-Carlo sampling to define a negative set. Each time this procedure is performed, a different negative set is constructed. Therefore, in order to ensure reproducibility of the data, several instances of the negative set should be generated, rather than a unique reference negative set. A kernel-based predictor for XPRTases (EC 2.4.2.22) based on MetaCyc information was constructed by using a balanced training set formed by 100 positive and 100 negative sequences and the *kernlab* package [30]. The predictor works as a classifier that assigns the sequence to either the positive or the negative class. The obtained performance in a tenfold cross-validation was of an accuracy of 94 %, an ROC area of 87 %, precision of 94 %, sensitivity of 95 %, and specificity of 93 %, showing the good performance that often can be achieved by a machine-learning approach, although such results should always be considered in terms of domain of applicability, i.e., to the actual extent of the sequence space where the predictor has been validated (*see* **Note 11**).

3.4.2 Energy-Based Approach

Energy-based methods aim at predicting the effect of mutations from first physics-based principles from precise quantum mechanical (QM) description to classical molecular mechanics (MM). The difficulty inherent to the energy-based methods is to have both accuracy and sampling. The later controls the quality of the statistical mechanical variables such as (full or partial) free energies. The hybrid QM/MM (quantum mechanics/molecular mechanics) approach combines the strength of both QM (accuracy) and MM (speed) calculations. This approach is mostly used in computational protein design to calculate the conformation and the barrier energy of transition state of a ligand inside the binding pocket.

The available codes are listed in Table 1. Conformational sampling of the protein conformation variable is usually done using MM force field to reduce computer time. Such force fields are implemented into Monte Carlo (MC) or molecular dynamics procedures. To estimate the influence of a mutant side chain at the protein binding interface or at a ligand binding site, it is possible to restrict the number of degrees of freedom to the region of interest to limit the statistical errors due to insufficient conformational sampling (*see* **Note 12**).

For a given choice of side chains at chosen positions, we search for the most favorable side chain configuration in terms of energetic fitness among the allowed interacting positions of the substrate on the enzyme in order to model the transition state of substrate-enzyme. The cluster of substrate conformers with the lowest fitness energy is estimated. Fitness considers internal, solvent and surface energies. The interface is then computed in order to determine residues that belong to the catalytic site.

One of the most successful energy-based approaches for computational protein design is Rosetta [31], a computational algorithm that has been experimentally validated to stabilize naturally occurring proteins and in the de novo design of protein structures. In the Rosetta method, protein backbone conformation is specified as a list of backbone torsion angles, while side chains are restricted to discrete conformations from a backbone-dependent rotamer library. Many Rosetta-based computational protocols for protein design are available. For instance, RosettaDesign is a Rosetta-derived server that performs Monte Carlo optimization with simulated annealing to search for amino acids that pack well on the target structure and satisfy hydrogen bonding potential. Other protocols using the Rosetta program are available as well, as for instance a protocol for enzyme design that models the reaction transition state using quantum chemistry methods in order to define side chain and backbone constrains associated with positions optimal for catalysts [32].

The structure of chain A of 2FXV was submitted to the RosettaDesign server without any substrate, and in complex with GMP and XMP. The former was in the pose determined in the crystal structure, while the pose of the latter was determined by docking using CombiDOCK [33]. Best mutations for each case and their binding energy are given in Table 3. The negative energy values are relative to wild type. Therefore, negative scores correspond to stabilized mutants. Best candidates for improving substrate affinity towards XMP are given for those positions where the negative increase in energy for mutations improving XMP affinity are significantly higher than those for GMP, as it is the case for ASP125THR, SER60THR, ASN27GLU, and SER59PHE.

Table 3
Most beneficial mutations for top-scored positions of the 2FXV structure according to RosettaDesign for the single chain, and for the chain in complex with the GMP and XMP substrates

Rank	Residue	Seq	Score	Rosetta	Rosetta	Rosetta GMP	Rosetta GMP	Rosetta XMP	Rosetta XMP
1	100	T	2.87	S	−1.41	S	−1.57	R	−1.05
2	98	S	2.26	S	−2.08	S	−2.11	A	−1.87
3	132	A	2.22	H	−0.53	L	−1.70	I	−1.72
4	125	D	2.09	E	−1.46	S	−1.13	T	−1.96
5	126	F	2.06	H	−1.54	H	−0.68	H	−1.01
6	101	K	2.01	D	−1.66	D	−1.66	Y	−1.28
7	81	K	1.96	E	−1.18	L	−1.25	E	−1.12
8	60	S	1.92	E	−0.72	E	−0.85	T	−1.37
9	99	F	1.86	T	−1.31	T	−1.25	T	−1.30
10	27	N	1.78	Q	−2.09	Q	−2.24	E	−2.43
11	124	D	1.76	H	−1.25	T	−1.35	E	−1.58
12	58	E	1.74	A	0.32	I	0.92	E	1.13
13	59	S	1.62	S	0.60	S	0.50	F	1.69
14	128	A	1.49	T	−0.83	T	−0.82	T	−0.66
15	96	V	1.25	I	−3.19	I	−3.08	I	−3.29

4 Notes

1. Hot spots are residues in protein interaction interfaces that play an important role modulating protein activity. Generally, such hot spots will disrupt the interaction when mutated (negative hot spots), although they can also have the ability of increasing it (positive hot spots). Hot spots are both defined in the context of protein-protein interactions and in enzyme-substrate or receptor-drug interactions. Effects of positive and negative cooperativity among hot spots are often observed, i.e., the simultaneous mutation of two hot spots has a significant higher (or lower) effect than each one individually.
2. In a virtual screening procedure, computation times are determined by the scoring function. Therefore, this function needs to be minimized and sequence-based methods should be preferred to structure-based. In general, methods at atomic level involving molecular dynamics should be carefully used. A possibility is the use of a multi-level screening, at first using coarse

grain-based methods such as knowledge-based potentials, in order to identify hot spots, leaving mutagenesis and docking methods involving molecular dynamics for a second step involving a smaller number of mutations.

3. Typically, ECFP descriptors consider fragments up to some given diameter, or distance to the root atom, and several graph canonicalization operations are performed in order to guarantee uniqueness of the molecule- or reaction-coding representation for proper comparison between fragments.
4. The fact that the activity in the protein appears promiscuously can be due to several factors, for instance in a multi-domain protein where each domain might have become specialized in one type of activity. Therefore, when considering the similarity of a candidate to the positive set, domain architecture of the protein should be preferably used rather than protein full sequence.
5. The advantage of the structural alignment is the often increase in quality from a multiple sequence alignment for a given protein structural class. Furthermore, this alignment could further be used as a template for homology modeling of a protein with unknown structure. The quality of the 3D superposition is given by the average of RMSD, root mean square deviation between all protein pairs in the set. The errors are usually small in the area corresponding to the function, i.e., the catalytic residues and to the core of the protein fold. Larger errors are observed in protein loops.
6. It should be taken into account that similarity matrices can always be computed between vectors of descriptors in the same fashion as computing distances between vectors. The main advantage of using similarity matrices arrives in cases when the descriptor space is considerable large (larger than the number of points in the training set). The interpretation of the results, however, is often harder when working with similarity matrices.
7. *Z* scores, i.e., scores that are normalized by subtraction of their mean values and divided by their standard deviation, allow scores comparison from different predictors and can serve to identify extreme values.
8. One of the main difficulties in the machine learning approach is the selection of the negative set or "bad" set in the training set. Ideally, a negative set has to contain a diverse-enough list of instances of pair protein-ligand or substrate. In principle, this goal can be accomplished by selecting randomly sequences that are annotated for ligands or substrates that are the most dissimilar to the desired target interacting partner.

9. In the same way as here the training set has been formed by pairs (sequence, reaction), training sets can be formed based on pairs (protein, ligand), (protein, inhibitor), (target, drug), (protein, protein), etc.
10. An excellent source for machine-learning computational methods are the *R* packages available at the CRAN repository, like *kernlab*, *randomForest*, *pls*, and *caret*.
11. Although often kernel-based predictors show a remarkably high performance, several considerations need to be taken into account. First, convergence is not always guaranteed in such type of predictors; that is, if the available sequence information is scarce and reactions in the cluster are too different, the predictor will likely not converge. Secondly, the fact that sequence information containing diversity is needed to build the predictor can often lead to a poor performance in terms of specificity. Even if the predictor is highly specific for the reaction cluster, such reaction cluster can happen to contain many diverse reactions apart from the target one.
12. Combinatorial sampling of the set of mutants can be optimized during the conformational sampling as implemented for example in CombiDOCK in the QXP/FLO package [33]. However, most energy-based methods do not sample the type of side chain and therefore can only be used on a chosen set of mutant combinations.

Acknowledgements

This work was funded by Genopole®, UniverSud Paris, and Agence Nationale de la Recherche (ANR Chaire d'excellence). UPFellows program with the support of the Marie Curie COFUND program.

References

1. Kazlauskas RJ, Bornscheuer UT (2009) Finding better protein engineering strategies. Nat Chem Biol 5:526–529
2. Cobb RE, Sun N, Zhao H (2012) Directed evolution as a powerful synthetic biology tool. Methods. doi:10.1016/j.ymeth.2012.03.009
3. Hayes RJ, Bentzien J, Ary ML et al (2002) Combining computational and experimental screening for rapid optimization of protein properties. Proc Natl Acad Sci U S A 99:15926–15931
4. Becskei A, Serrano L (2000) Engineering stability in gene networks by autoregulation. Nature 405:590–593
5. Elowitz MB, Leibler S (2000) A synthetic oscillatory network of transcriptional regulators. Nature 403:335–338
6. Gardner TS, Cantor CR, Collins JJ (2000) Construction of a genetic toggle switch in Escherichia coli. Nature 403:339–342
7. Van der Sloot AM, Kiel C, Serrano L, Stricher F (2009) Protein design in biological networks: from manipulating the input to modifying the output. Protein Eng Des Sel 22:537–542
8. Chang MC, Keasling JD (2006) Production of isoprenoid pharmaceuticals by engineered microbes. Nat Chem Biol 2:674–681

9. Carbonell P, Planson AG, Fichera D, Faulon JL (2011) A retrosynthetic biology approach to metabolic pathway design for therapeutic production. BMC Syst Biol 5:122
10. Grünberg R, Serrano L (2010) Strategies for protein synthetic biology. Nucleic Acids Res 38:2663–2675
11. Looger LL, Dwyer MA, Smith JJ, Hellinga HW (2003) Computational design of receptor and sensor proteins with novel functions. Nature 423:185–190
12. Schmidt M, de Lorenzo V (2012) Synthetic constructs in/for the environment: managing the interplay between natural and engineered Biology. FEBS Lett 586:2199–2206
13. Dueber JE, Wu GC, Malmirchegini GR et al (2009) Synthetic protein scaffolds provide modular control over metabolic flux. Nat Biotechnol 27:753–759
14. Foo JL, Ching CB, Chang MW, Leong SS (2011) The imminent role of protein engineering in synthetic biology. Biotechnol Adv. doi:10.1016/j.biotechadv.2011.09.008
15. Pleiss J (2011) Protein design in metabolic engineering and synthetic biology. Curr Opin Biotechnol 22:611–617
16. Lippow SM, Tidor B (2007) Progress in computational protein design. Curr Opin Biotechnol 18:305–311
17. Li X, Zhang Z, Song J (2012) Computational protein design approaches with significant biological outcomes: progress and challenges. Comp Struct Biotechnol J. doi:10.5936/csbj.201209007
18. Tiwari M, Singh R, Singh R et al (2012) Computational approaches for rational design of proteins with novel functionalities. Comp Struct Biotechnol J. doi:10.5936/csbj.201209002
19. Tsai M, Wu JT, Gunawardhana L, Naik H (2012) The effects of xanthine oxidase inhibition by febuxostat on the pharmacokinetics of theophylline. Int J Clin Pharmcol Ther 50:331–337
20. Rudolph MM, Vockenhuber MP, Suess B (2013) Synthetic riboswitches for the conditional control of gene expression in Streptomyces coelicolor. Microbiology. doi:10.1099/mic.0.067322-0
21. Michener JK, Smolke CD (2012) High-throughput enzyme evolution in Saccharomyces cerevisiae using a synthetic RNA switch. Metab Eng 14:306–316
22. Li W, Godzik A (2006) Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. Bioinformatics 22:1658–1659
23. Taly JF, Magis C, Bussotti G et al (2011) Using the T-Coffee package to build multiple sequence alignments of protein, RNA, DNA sequences and 3D structures. Nat Protoc 6:1669–1682
24. Pettersen EF, Goddard TD, Huang CC et al (2004) UCSF Chimera—a visualization system for exploratory research and analysis. J Comput Chem 25:1605–1612
25. Doppelt-Azeroual O, Moriaud F, Adcock SA, Delfaud F (2009) A review of MED-SuMo applications. Infect Disord Drug Targets 9:344–357
26. Jambon M, Andrieu O, Combet C et al (2005) The SuMo server: 3D search for protein functional sites. Bioinformatics 21:3929–3930
27. Maggiora GM, Shanmugasundaram V (2011) Molecular similarity measures. Methods Mol Biol 672:39–100
28. Carbonell P, Carlsson L, Faulon JL (2013) Stereo signature molecular descriptor. J Chem Inf Model 53:887–897
29. Chang A, Scheer M, Grote A et al (2009) BRENDA, AMENDA and FRENDA the enzyme information system: new content and tools in 2009. Nucleic Acids Res 37: D588–D592
30. Karatzoglou A, Smola A, Hornik K, Zeileis A (2004) Kernlab—an S4 package for Kernel methods in R. J Stat Software 11:1–20
31. Liu Y, Kuhlman B (2006) RosettaDesign server for protein design. Nucleic Acids Res 34:W235–W238
32. Richter F, Leaver-Fay A, Khare SD et al (2011) De novo enzyme design using Rosetta3. PLoS One 6:e19230
33. Zhou JZ (2008) Structure-directed combinatorial library design. Curr Opin Chem Biol 12:379–385

Chapter 2

Computer-Aided Design of DNA Origami Structures

Denis Selnihhin and Ebbe Sloth Andersen

Abstract

The DNA origami method enables the creation of complex nanoscale objects that can be used to organize molecular components and to function as reconfigurable mechanical devices. Of relevance to synthetic biology, DNA origami structures can be delivered to cells where they can perform complicated sense-and-act tasks, and can be used as scaffolds to organize enzymes for enhanced synthesis. The design of DNA origami structures is a complicated matter and is most efficiently done using dedicated software packages. This chapter describes a procedure for designing DNA origami structures using a combination of state-of-the-art software tools. First, we introduce the basic method for calculating crossover positions between DNA helices and the standard crossover patterns for flat, square, and honeycomb DNA origami lattices. Second, we provide a step-by-step tutorial for the design of a simple DNA origami biosensor device, from schematic idea to blueprint creation and to 3D modeling and animation, and explain how careful modeling can facilitate later experimentation in the laboratory.

Key words DNA, Nanotechnology, Origami, Biosensor, CAD, Software

1 Introduction

The methods for designing DNA nanostructures have been developing over the last three decades with the objective of creating well-ordered DNA lattices to organize and control matter at the nanoscale [1, 2]. When the DNA origami method was introduced in 2006 it provided many new features that were not previously achievable such as programmable overall shape, sequence-specific addressability on a large structure, and high self-assembly yield [3]. The trick to obtain these features was to fold a long DNA "scaffold" strand into a desired shape using several smaller DNA "staple" strands. The scaffold strand was obtained from a natural source, the M13mp18 bacteriophage single-stranded DNA genome with a length of 7,249 nucleotides, and the staple strands were obtained by chemical synthesis—usually about 230 DNA strands of 32 nucleotides each. The self-assembly was done by mixing all DNA strands in a standard buffer with high magnesium concentration followed by heat-annealing of the strands whereby

Mario Andrea Marchisio (ed.), *Computational Methods in Synthetic Biology*, Methods in Molecular Biology, vol. 1244,
DOI 10.1007/978-1-4939-1878-2_2, © Springer Science+Business Media New York 2015

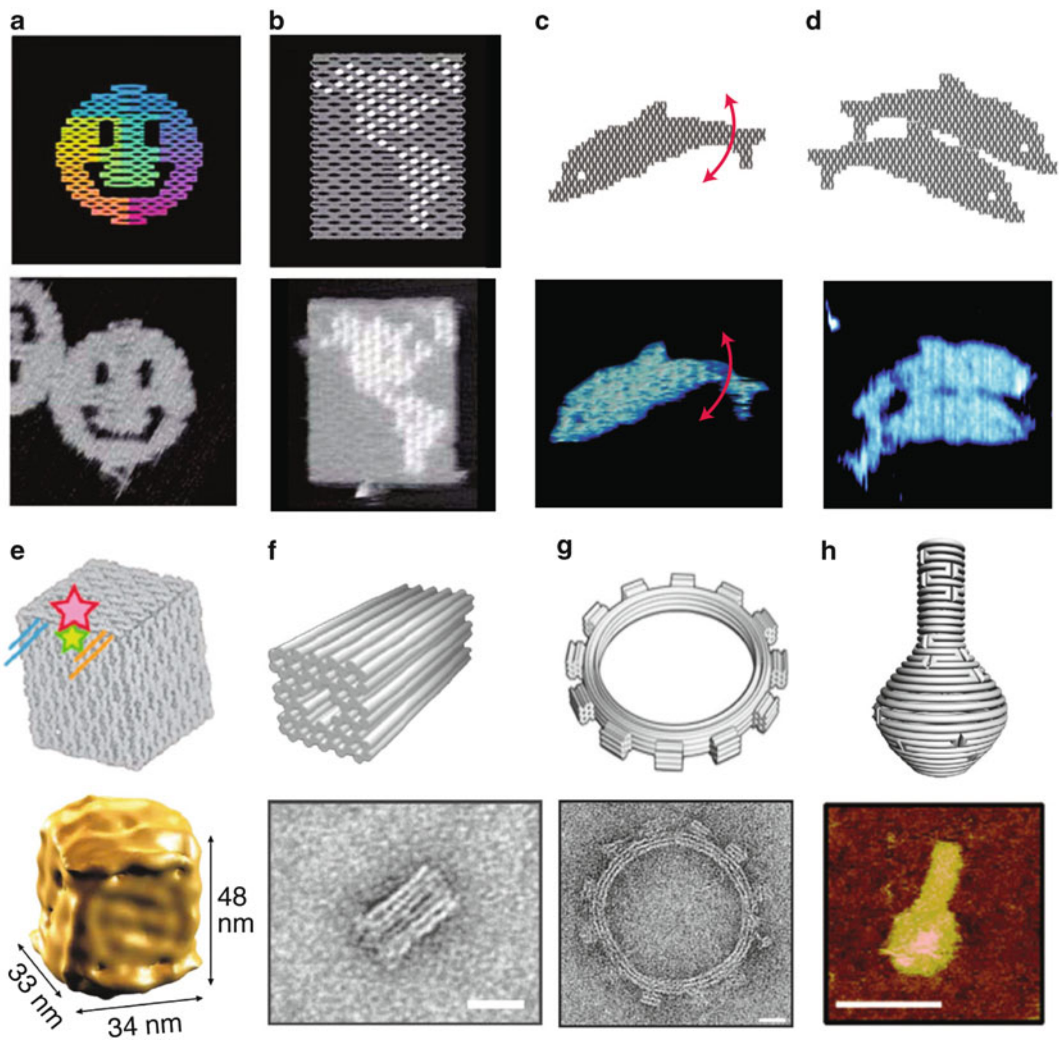


Fig. 1 DNA origami structures. (**a**–**d**) 2D DNA origami structures: (**a**) DNA smiley face strand path and corresponding AFM (atomic force microscopy) image [3]. (**b**) DNA rectangle with map and corresponding AFM image [3]. (**c**, **d**) DNA dolphin with specific shape recognition and corresponding AFM image [5]. (**e**–**h**) 3D origami structures: (**e**) DNA box with controllable lid model and cryo-EM reconstruction [8]. (**f**) Square nut model and TEM image with scale bar of 20 nm [9]. (**g**) Twelve-tooth gear model and TEM image [10]. (**h**) DNA origami flask model and AFM image with scale bar of 75 nm [11]

the desired structures form. Shapes with complicated folding paths of the scaffold strand could be obtained (Fig. 1a) and the addressability was demonstrated by the addition of "pixels" (projecting DNA structures) on the surface of the DNA origami sheets (Fig. 1b). The method was rapidly adopted by other research groups to produce new shapes [4, 5] and to label the structures with other materials [6, 7]. An early hint that DNA origamis could act as mechanical devices in solution was suggested by the design and demonstration of DNA origamis with flexible regions

(Fig. 1c) and post-assembly sequence/shape-recognition between structures in solution (Fig. 1d). A second development was the design of three-dimensional (3D) DNA origami structures. The first approach demonstrated that DNA origami sheets can be arranged in 3D by geometry-inducing crossovers to form a DNA origami box with a controllable lid (Fig. 1e) [8]. The second approach was to stack DNA origami sheets to form solid 3D shapes (Fig. 1f) [9], which could be further curved and twisted into more complicated shapes (Fig. 1g) [10]. The curvature and bending of single-layer DNA origamis could also be controlled by choosing crossover positions to enforce an overall shape (Fig. 1h) [11]. Several other design developments have been demonstrated later, e.g., the DNA origami gridiron [12], DNA origami with parallel crossovers [13], and single-stranded tiles with single crossovers that assembles efficiently in the absence of a long scaffold strand [14, 15].

Several computational tools have been developed to facilitate the design of DNA nanostructures [16]. The most relevant tools for designing DNA origami structures are listed in Table 1 and are often used in conjunction during the design process as illustrated in Fig. 2. The design process usually starts with an idea for a shape or device sketched out on paper, where the strand path and size of the object is roughly calculated (Fig. 2a). The second step is the creation of a digital blueprint of the strand path of the scaffold strand and the positions of the staple strands (Fig. 2b). Two software packages are available for this task: SARSE [5] and Cadnano [17].

Table 1
Software for DNA origami design and analysis

Name	Description	Link	References
caDNAno	Blueprint editor and 3D editor for DNA origami design	http://cadnano.org	[17]
CanDo	Coarse-grained modeling of strain and flexibility in DNA origami structures	http://cando-dna-origami.org	[21, 27]
NUPACK	Thermodynamic analysis of the annealing of multiple DNA strands	http://nupack.org	[33]
OxDNA	Coarse-grained modeling thermodynamic DNA hybridization	http://dna.physics.ox.ac.uk/	[24]
SARSE	2D editor for DNA origami design with output of atomic models	http://cdna.au.dk/software/	[5]
TIAMAT	3D editor with sequence symmetry minimization	http://yanlab.asu.edu/Resources.html	[18]
Uniquimer	3D editor with symmetry and energy minimization	http://ihome.ust.hk/~keymix/uniquimer3D/	[19]

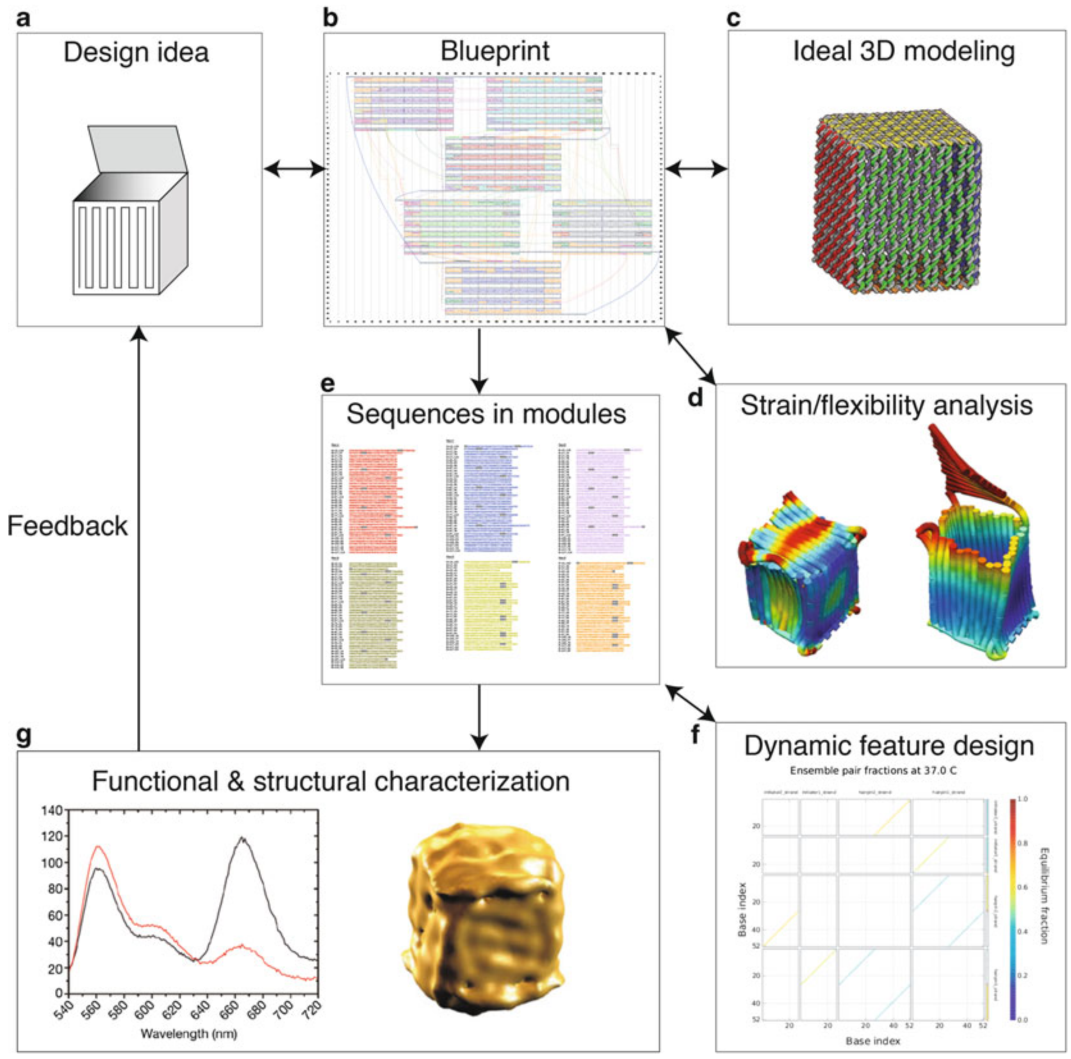


Fig. 2 DNA origami design procedure. Flow chart of the different steps in the design of a DNA origami box structure. (**a**) An initial idea is materialized by calculating the possible dimensions of the object given the length of the scaffold strand to be used. (**b**) A blueprint is constructed using the SARSE or Cadnano software packages [5, 17]. (**c**) 3D modeling is used to evaluate how modules are placed in relation to each other and used for the design of 3D staple strand connections. (**d**) Coarse-grained modeling using the CanDo program [21, 27] can be used to evaluate the strain and flexibility of the structure. (**e**) Sequences are exported from the blueprint program and sorted into functional modules that can be used to make different versions of the designed structure. (**f**) DNA strands used for strand displacement can be designed in dedicated software. (**g**) DNA is ordered, self-assembled, and tested in structural and functional experiments

The SARSE software allows automated creation of blueprints based on shapes specified in bitmap files, detailed editing of the blueprint, and export of sequences and atomic models. The software only works for flat DNA origami structures and has not been actively developed to accommodate new design principles. Cadnano allows the manual construction of DNA origami blueprints and supports both square and honeycomb lattices. The software is actively developed and the json files have become the standard blueprint file format. 3D modeling software is another important tool for designing complicated 3D DNA origami structures (Fig. 2c). Cadnano has been integrated as a plug-in in the 3D modeling environment Autodesk Maya, which provides a live 3D CAD experience by linking 2D blueprint to 3D model. Other examples of dedicated 3D modeling software for construction of DNA nanostructures are TIAMAT [18] and Uniquimer3D [19], but general molecular visualization software like UCSF Chimera [20] can also be used for model building and rendering of models. When a blueprint is finished the 3D shape can be analyzed using the CanDo web server (Fig. 2d) that uses the crossover positions in the DNA origami blueprint to determine the overall shape using a finite element analysis method [21]. CanDo can both be used to detect unwanted distortions and to design bend and twisted shapes on purpose. When a satisfactory design is achieved the DNA staple strands are exported based on the blueprint and a choice of scaffold sequence (Fig. 2e). At this stage the DNA strands are normally sorted into modules that allow functional or structural elements to be substituted at the pipetting stage. Functional modules like strand displacement systems [22] can be designed in other software like NUPACK [23] and mechanistic details can be modeled in coarse-grained modeling software OxDNA [24]. After ordering of the final DNA sequences, the DNA origami structure can be self-assembled in the laboratory, characterized by biophysical techniques, and applied for the desired purpose (Fig. 2f). As always in design processes the testing stage can be used as feedback to improve the design if it does not meet the requirements.

Tutorials for designing DNA origami structures are found both on the websites of the software and published in the literature. The original DNA origami papers describe the design principles in detail [3, 9–15, 17, 25]. We have earlier provided a tutorial for the design of 3D DNA origami objects using the SARSE software [26]. Dietz and colleagues have provided a detailed step-by-step tutorial to DNA origami design using caDNAno and CanDo as well as experimental protocols [27]. In this chapter we start by describing how crossover positions in DNA origami structures are calculated and discuss the standard flat, square and honeycomb lattices. Next, we provide a simple design example and the step-by-step tutorial for designing the structure using caDNAno and for 3D modeling and animation using Maya. We have chosen this example to show how easy it is to make a functional mechanical device and hope it will provide a stepping stone for creating more complex and advanced devices.

2 Materials

1. Download and install cadnano 2.2.0 from http://cadnano.org.
2. Download and install Autodesk Maya 2012 from http://www.autodesk.com/education/free-software/maya.
3. Three button mouse.

3 Methods

3.1 Calculating Crossover Positions Between DNA Helices

To construct a crossover between two DNA helices we can work with a simplified 3D DNA model showing only the position of phosphate (P) atoms along the backbone (Fig. 3). The simplified model has a minor groove angle of 133.0° and a twist angle of 34.48°/bp, which corresponds to 10.44 bp/turn along the helical axis for a standard B-form helix [28]. A crossover is made by aligning two P atoms of the two DNA helices, breaking the bond on the same side of the P atoms, and rejoining the strands between the helices (Fig. 3a). The crossover operation results in a four-way junction that has two possible stacking conformations and some flexibility at the junction [29]. To fix the two helices rigidly in relation to each other a second crossover has to be made. The double crossover can be made in five different ways named DAE, DAO, DPE, DPON, and DPOW, where D stands for double crossover, A/P for antiparallel or parallel strand direction at the crossover, E/O for an even or odd number of half-turns between the crossovers, and N/W for major/wide or minor/narrow groove bridging in the parallel odd case [29]. Here we only describe crossovers between antiparallel strands because these are the ones used in the standard DNA origami structures discussed below. To calculate the position of the second crossover on the same strand (DAE) we use the twist angle of 34.48°/bp (*see* **Note 1**) and find that it has an optimal spacing of 21 bp (Fig. 3b). Where phosphates meet, a second crossover can be made using the same operation as described in Fig. 3a. To calculate the position of the second crossover on the opposite strand (DAO) we use the minor groove angle of 133.0° and the twist angle of 34.48°/bp (*see* **Note 1**) and find that it has an optimal spacing of 16 bp (Fig. 3c). As seen on the side views to the right of the models a DAE has the minor grooves on the same side, while a DAO molecule has the major and minor grooves on opposite sides at the crossovers, which leads to different pseudo symmetry axes [29]. Crossover positions do not always fit perfectly when calculated along the strand, but can work anyway since the DNA helix has a significant degree of flexibility and can accommodate suboptimal conformations.

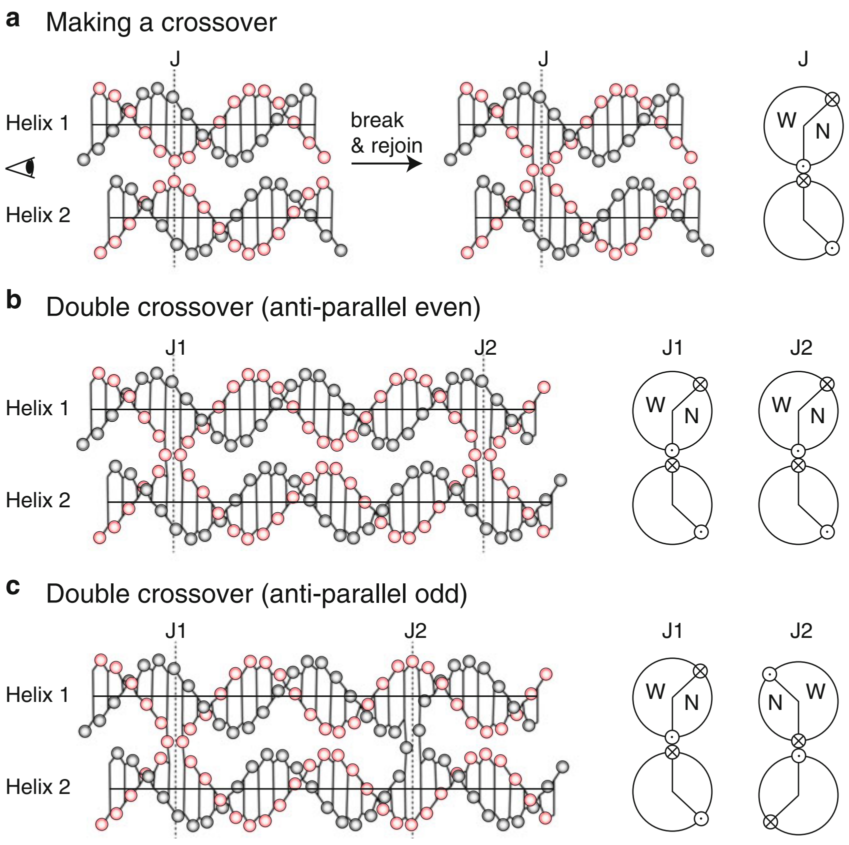


Fig. 3 Formation of crossovers between two DNA helices. (**a**) *Left*: Two phosphates are aligned in the plane between two helices. *Right*: Strands are broken and rejoined to form the crossover. (**b**) A double-crossover molecule with crossovers on the same strand with a spacing of 21 base pairs. (**c**) A double-crossover molecule with crossovers on the two complementary strands with a spacing of 16 base pairs. DNA helices are based on atomic models, but are shown here in a simplified representation, where the phosphates are shown as spheres, lines are drawn between P and C3 atoms along the backbone, and lines between two C3 atoms represent base pairs. The two complementary strands are colored with black and red. The 5′ end of the strands is seen as ending with a P sphere. Helical axes are shown and positions of crossovers are indicated by *dashed lines* (J = junction). Junctions are also shown in side view on the *right* as seen along the helical axis from the *left* of the models (*see* eye icon). The major/wide groove is marked with a W. The minor/narrow groove is marked with a N. The strand direction at the junction is shown as a *circle with a dot* (3′ end) and a *circle with a cross* (5′ end)

3.2 Scaffolded DNA Origami Crossover Patterns

In DNA origami structures the anti-parallel type of crossovers is used to connect multiple helices in several configurations (Fig. 4). The majority of crossovers in a DNA origami is made by the staple strands, which are all on the same strand in relation to each other and thus calculated as in Fig. 3b. The scaffold strand also makes crossovers as it threads through the structure and these crossovers are on the opposite strand and calculated as in Fig. 3c. The standard flat DNA origami structures [3] seeks to make all crossovers in the plane (Fig. 4a). It has a spacing of 32 bp (3 turns) between

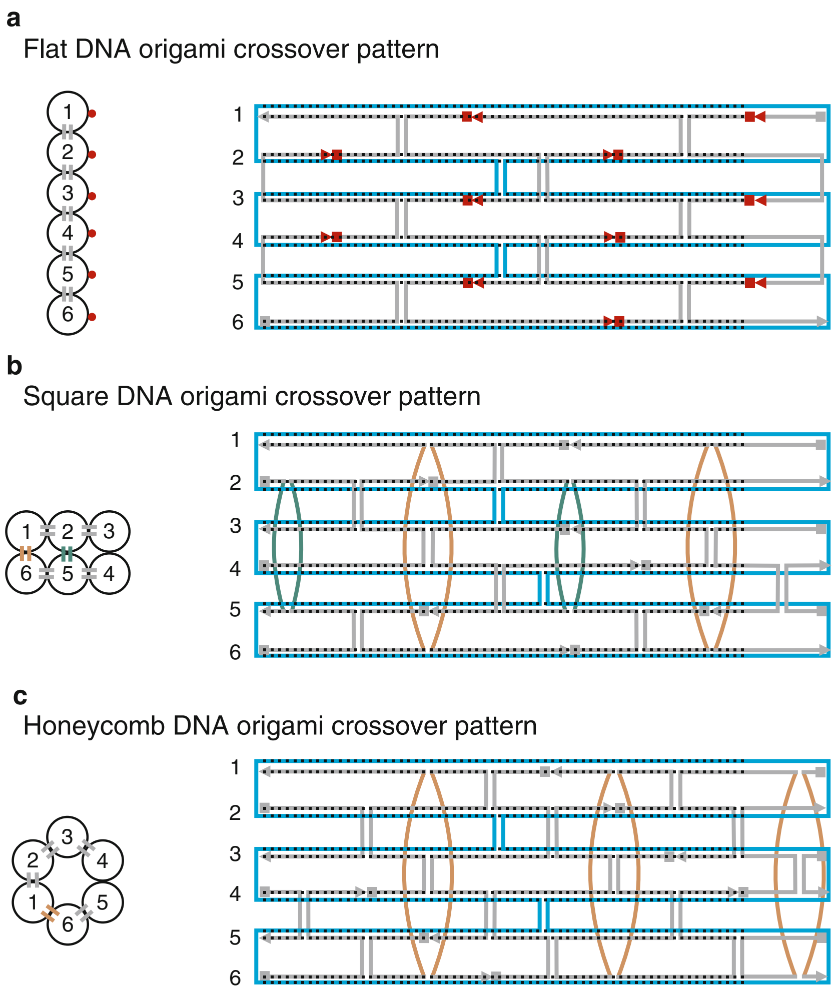

a
Flat DNA origami crossover pattern
1
2
3
4
5
6
b
Square DNA origami crossover pattern
1
2
3
4
5
6
c
Honeycomb DNA origami crossover pattern
1
2
3
4
5
6

Fig. 4 DNA origami crossover patterns. *Left*: Numbered helixes shown in side view. *Right*: Blueprints with scaffold strand in cyan and staple strands in *grey*. Crossovers between distant helices are colored with *orange* and *green*. DNA bases are marked as *black dots on the strands*. The scaffold strand is circular and runs clockwise through the structure (Color figure online)

crossovers of the same two helices, and a 16 bp (1.5 turn) spacing when crossing over to a third helix in the plane. Thus, this architecture assumes 32 bp/3 turns = 10.67 bp/turn and a twist of 360°/10.67 bp = 33.74°/bp. Since the real value is 10.44 bp/turn [28] the resulting DNA structure will be undertwisted resulting in an overall distortion of the structure which can be investigated using the CanDo software (Table 1). The staple strands are often broken in the middle of the 16 bp region on every other helix which results in a 8-16-8 bp annealing region pattern. The pattern is repeated through the whole structure and only interrupted on the edges. The way the strands are broken in Fig. 4a makes all staple strands form an "S" shape and the 5′ and 3′ ends are all positioned on the upper surface of the DNA origami sheet (marked in red in Fig. 4a). The staple strands can also be broken on alternate helices resulting in a "Z"-shaped staple strand. In this case the 5′ and 3′ end points to the other side of the DNA origami sheet. At last one can make combinations of S and Z staples to position the ends on both sides of the sheets. 5′ and 3′ ends are good for placing chemical modifications. Square DNA origami is a 3D version of flat DNA origami where the same helicity and spacing between crossovers are used. Scaffold strand segments that make up helix 1 and 6 are placed far apart on the scaffold strand and have to be connected together via staple strands for the structure to be stable. The first staple strand (from the left) of helix 1 has a crossover that connects it with helix 6 (orange crossover in Fig. 4b) and with helix 5 (Fig. 4b). This staple strand has to make 3/4 of a turn (8 base pairs) on helix 6 to be able to crossover to helix 5 after crossing over from helix 1 (side view in Fig. 4b). The same is valid for the crossovers connecting helices 2 and 5 (green crossovers in Fig. 4b). Honeycomb DNA origami uses 10.5 bp/turn, which is very close to the reported value of 10.44 bp/turn [28] (*see* **Note 2**), and allows it to have a spacing of 21 bp between the crossovers of the same strand (Fig. 3b). As seen in the side view in Fig. 4c the honeycomb arrangement of helices are linked by local crossovers (grey) and long-range crossovers between helix 1 and 6 (orange). Annealing regions in honeycomb lattices are 7 bp.

3.3 Design of a DNA Origami Device

Here we demonstrate the design process for making a DNA origami biosensor (Fig. 5). The structure consist of two six-helix bundles that are connected through a hinge at one end and held together via complementary oligonucleotides protruding the other end. Complementary oligonucleotides hold the structure together in a closed conformation. Donor and acceptor fluorophores are

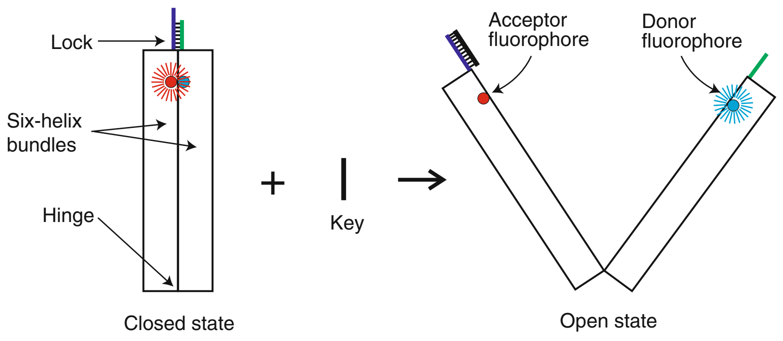


Fig. 5 The sketch of DNA origami six-helix bundle sensor. The sensor consists of two six-helix bundles that are connected in one end by a scaffold strand and held together by an oligo lock in other end. In the closed conformation there is FRET between donor and acceptor fluorophores that are positioned at the end of the each bundle. When the structure is in its open conformation the distance between the fluorophores is too large for FRET to occur

placed at the end of each bundle to detect the opening of the device by Förster resonance energy transfer (FRET). In the closed conformation FRET occurs from donor to acceptor fluorophore resulting in acceptor fluorescent signal when the donor fluorophore is excited. One of the locking strands is designed with a toehold for a "key" strand that can displace the shorter locking strand from lock helix by the strand displacement mechanism [22]. Upon the addition of a key the lock helix is opened and the biosensor opens due to the electrostatic repulsion between the DNA helices. Increased distance between fluorophores decrease FRET resulting in only donor fluorescence when it is excited. The key strand acts as a signal that can be measured by acceptor and donor fluorescence and the structure can be used as a biosensing platform for the key strand.

3.4 caDNAno Design

The DNA origami device described above can be implemented in different sizes. Here we choose a small size using 1,344 nucleotides to make an easy design process. The standard DNA origami structures consist of approximately 14,000 nucleotides.

1. Open the caDNAno program and get familiar with its interface (Fig. 6).
2. To begin, choose a honeycomb lattice from the main menu and draw a six helix bundle in the lattice window (Fig. 6a).
3. Click on the arrows just above the helices in the top right corner (Fig. 6g) to add 84 bp to the helices' length to 42 bp that are set by default.
4. Draw the scaffold strand as shown (Fig. 7d) using the "Pencil" tool from the tool box. Each bundle should be 61 bp long. A 5′ end can be connected with 3′ end using the "Pencil" tool. The scaffold strand can be connected only with another part of the scaffold strand and staple strands only with other staple strands. There is a more convenient way of drawing the scaffold path through the helices. Click with the left button of the mouse on helix "0" in the lattice window, hold the mouse button down and select all the other helices by moving the mouse clockwise. Release the button when the last helix is selected. The scaffold path centered on the "ruler" should appear connected via crossovers. Note that the cell array may not contain any drawn scaffold strands for this function to work properly. Correct automatic scaffold drawing according to the blueprint in Fig. 7d. This can be achieved by dragging the crossover elements using the "Select" tool. Remove any undesired crossovers by first selecting the crossover and then pressing the "Delete" key on the keyboard. Place 3′ and 5′ ends at desirable positions and use the "Pencil" tool to draw a crossover. The select tool only selects the elements that are defined in the "Selectable" menu above the blueprint window (*see* **Notes 3** and **4**).

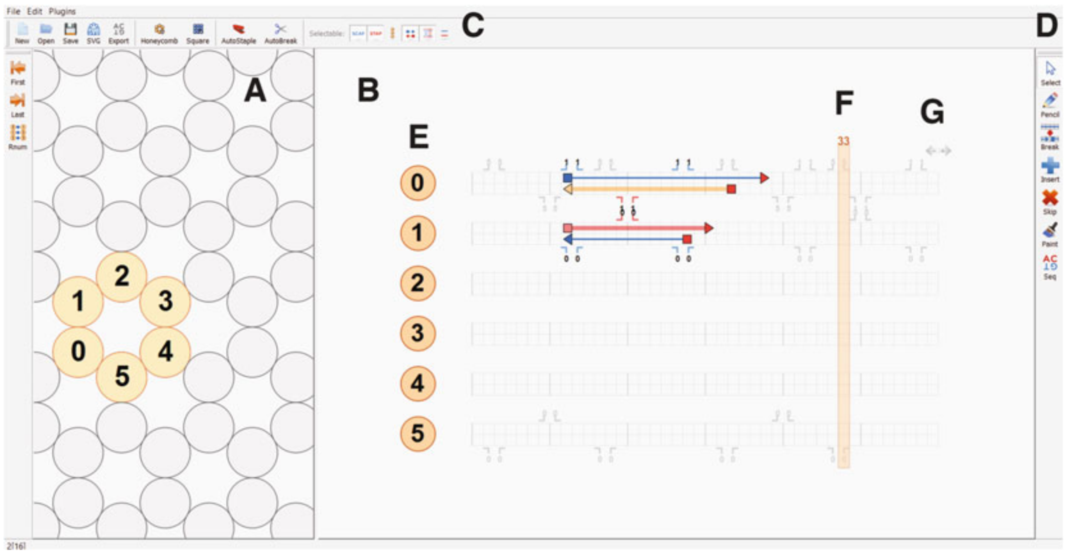


Fig. 6 caDNAno interface. (**a**) Lattice window shows how the helices are arranged in the designed structure. Honeycomb lattice is shown as an example. Each helix is numbered: the *odd* and *even numbers* represent the directionality of the scaffold/staple strand helices. Scaffold strand's directionality of the odd helices is opposite to directionality of the even ones. Alternatively a square lattice can be chosen. (**b**) Blueprint window shows how the crossovers between the helices hold the structure together. (**c**) Selectable menu allows to make the selection specific for defined elements. (**d**) Tool box: Select—use it to select elements in blueprint window, Pencil—allows to draw scaffold and staple strands in the cell arrays, Break—can be used to introduce breaks in the strands, Insert/Skip—allows to insert or skip a base on a desired strand, Paint—allows to change the color of the strands, Seq—sets the sequence of the scaffold strand and the staple strand sequence is generated automatically (**e**) Each helix is represented by 2xN cell array. The *top row* represents a scaffold strand and *bottom row* a staple strand (vice versa for odd numbered helices). Each cell represents a base of a DNA strand. (**f**) Movable ruler, (**g**) Addition/deletion of bases to the cell array

5. Click the "Autostaple" button—this will add staple strands automatically to the designed structure. Added staples are long strands that bind the scaffold strand in a regular pattern (*see* **Note 5**).
6. Use the "Break" tool from the tool box to cut the strands into shorter segments of 28–42 bp per strand. In the honeycomb lattice the staple strand pattern is usually 7-7-14-7-7 bp (or 7-7-7-14-7 bp etc.) where the number refers to the binding regions on the helix (*see* **Notes 6** and **7**).
7. Shorten the staples at the ends of each helix by two base pairs to remove the possible strain of the scaffold strand (Fig. 7e). More than one endpoint can be selected simultaneously by selecting "Staples" and "Endpoints" from the "Selectable" menu and hovering the arrow icon over the desired endpoints.
8. Now we will design oligonucleotides that will serve as the lock mechanism. Choose the two staple strands that will be functionalized with donor and acceptor fluorophores. Helices 0 and 5

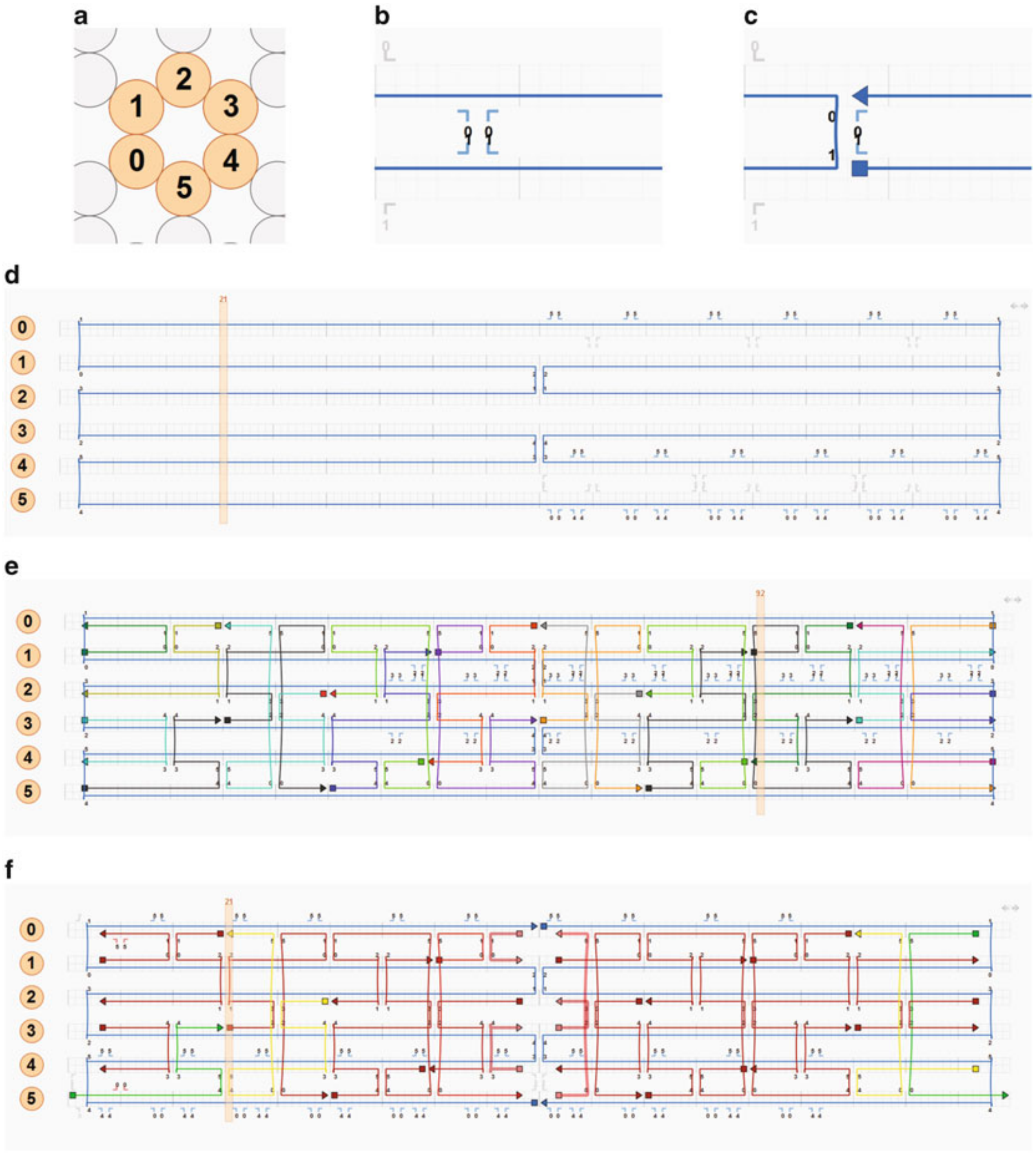


Fig. 7 Blueprint design in caDNAno. (**a**) Six-helix bundle drawn in lattice window. (**b**) Crossover markers. (**c**) One part of the (**d**) blueprint of scaffold strand. (**e**) Addition and cleavage of the staple. (**f**) Preparation of the design for Maya animation

connect the two bundles, determine the hinge axis, and are chosen for positioning the lock and FRET pair. Color all the staples with red by selecting the red "STAP" button in "Selectable" menu shown in Fig. 6c. Use both "Select" and "Paint" tool from the toolbox to color them all with red. Now color the lock staples with green and the fluorophore staples with yellow. Now make small changes on the lock and fluorophore staples as shown (Fig. 7f). Last thing to do is to break the scaffold strand at the bending points of helices 0 and 5 (Fig. 7f) (*see* **Note 8**).

9. Open Maya and click on the caDNAno plug-in button in the right upper corner beneath the “Minimize” and “Close” icons. The screen now consists of three windows: two of them are the familiar caDNAno windows and the third is the Maya window.
10. Open your caDNAno design file through the caDNAno window. Essentially, one has the same possibilities in the plug-in as in caDNAno itself. You can use “Alt + the left mouse” key to rotate the view around your object in the Maya window.
11. Our lock strands were initially designed too short. Now we can make them a little bit longer in two ways:
    (a) Use the select tool of the caDNAno window and edit the lock strands in caDNAno.
    (b) Select one of the green helices in Maya window by clicking with the left button of the mouse and drag the blue rhomb (Fig. 8f, g).
12. Do the same with the green staple strand from the other end of the object. Extend the possible helix length if needed by extending cell arrays in the caDNAno blueprint window.
13. The rest of the modeling is done in Maya only (*see* **Notes 9** and **10**) and caDNAno can now be closed to get more workspace. Click the caDNAno icon in the top right corner to close the plug-in.

3.5 Maya Modeling

The structure that is made in caDNAno depicts only the DNA part and fluorophores have to be added separately in Maya. The fluorophores are going to be represented as spheres of different color: donor is colored with green and acceptor with red.

1. The object has lost its color and only a wireframe is visible. Click the “Smooth shade all” button from the panel tool bar to bring the colors back (Fig. 8b).
2. Select Create > NURBS Primitive and deselect “interactive creation” at the bottom of the drop down menu (press “ctrl + M” on the keyboard if the menu bar is invisible). Now select Create > NURBS Primitive > Sphere and click on the small square next to it. In the Sphere option menu change the radius from 1 to 0.5. Click “create.” A sphere is now positioned at the center of the grid (the grid is invisible by default, but can be switched on by selecting Display > Grid). Press “Z” to undo an operation instead of the more common “Ctrl + Z” combination. “Shift + Z” will redo the operation.
3. The fluorophore has to be placed on the end of our yellow staple. This can be done with precision in 3D by viewing our object from *x*, *y*, and *z* directions in orthogonal view.

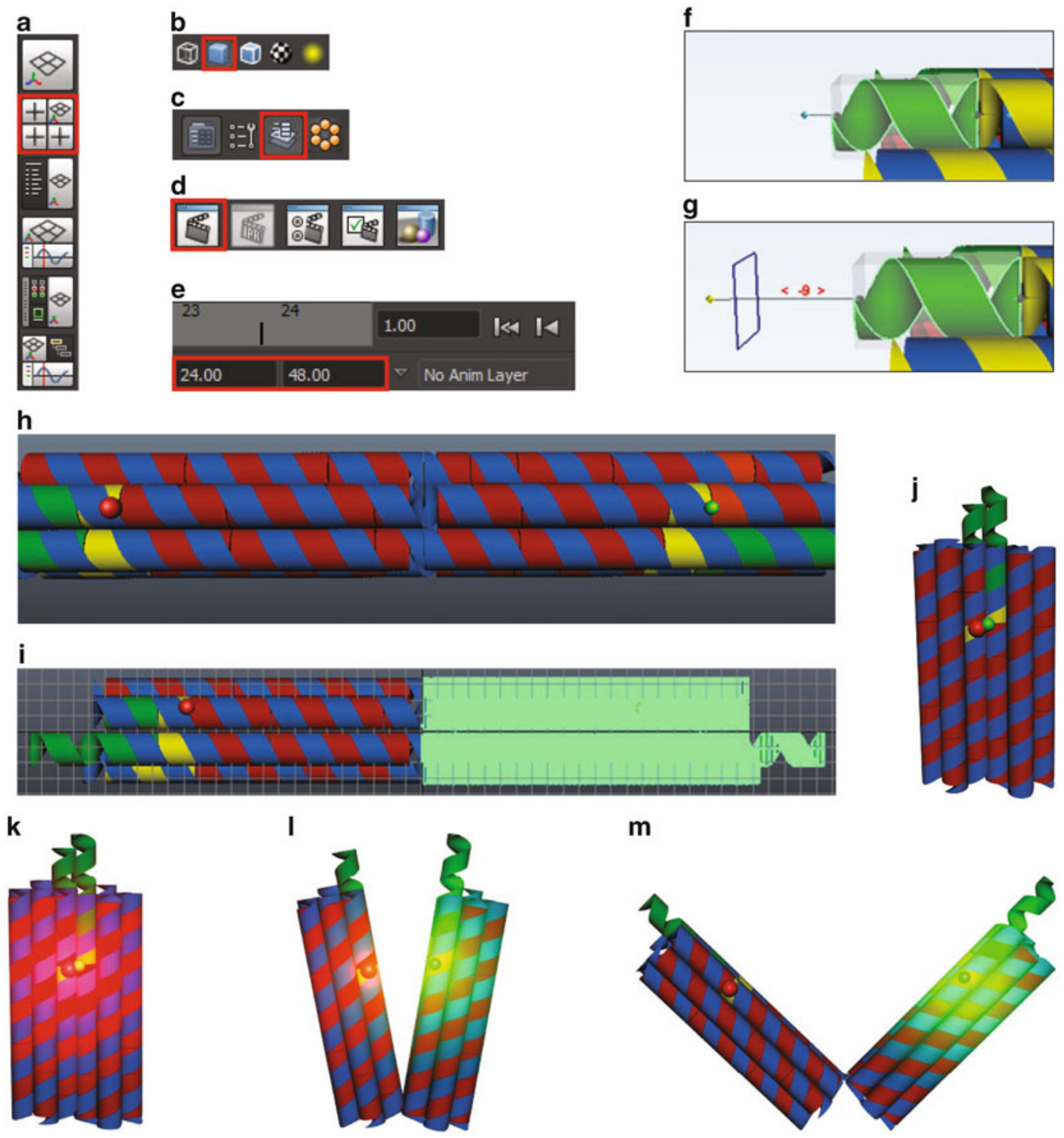


Fig. 8 3D modeling and animation in Maya. (**a**) Quick layouts buttons are placed beneath the tool bar menu. (**b**) Part of the panel tool bar. (**c**) Editor icons. (**d**) Rendering shelf. (**e**) Range slider. (**f**) Selection of a strand in Maya is shown as a *grey box*. (**g**) The length of the strand can be changed by moving the blue rhomb along the strand axis. (**h**) Adding fluorophores in Maya. (**i**) Grouping the selected elements in orthographic view. (**j**) Device in a closed state. (**k**–**m**) Animation of the opening of the device

The "quick layout" buttons allow you to change between different viewing modes. Choose the "Four view" option using the "quick layout" buttons just beneath the tool bar menu (Fig. 8a). The window is now split into four windows where the object is seen in three orthogonal views plus the perspective view.

4. Select the sphere with the select tool (it can be selected in any of the windows) and choose the move tool from the tool box on the left panel. Three arrows appear (red, green, and blue) each representing the direction that the sphere can be moved along the *x*, *y*, and *z* axes.
5. Drag the blue arrow with the left button of the mouse to the left. You can follow the movement of the sphere in all windows simultaneously. If the sphere disappeared in one of the windows use "alt + middle mouse button" to move the view in that window (scrolling the mouse wheel will zoom in/out). If a bigger view is needed in one of the windows press "space" while pointing on it with the mouse arrow. Press space again to return back to the four window view. Place the sphere on the 3′ end of the yellow staple on one six-helix bundle (use caDNAno to find out which end is 3′ and which is 5′).
6. Create the second sphere with a radius of 0.75 and place it at the 3′ end of the yellow staple on the other six-helix bundle. Both spheres are now placed at the right positions and will represent donor and acceptor fluorophores (Fig. 8h).
7. Click with the right button of the mouse on the sphere and while still holding the right mouse button choose "Assign a new Material" and release the button. Choose "Blinn" as a material and the red color for the big sphere and the green color for the small one.

   The initial state of the object is the closed one. The two bundles have to be brought together. Each bundle has a scaffold strand, multiple staple strands, and one fluorophore, which are grouped to treat them as one element in the animation.
8. Go back to the four view display and choose the right bottom window that shows the structure in side view. Select one of the halves (the selected area should turn green) and press "ctrl + G" to group all the selected elements into one group (Fig. 8i). Select and group the other half.
9. Select the Panels menu and choose Saved Layouts > Persp/Hypergraph. The view is changed to perspective and the Hypergraph window is now visible. The Hypergraph window depicts all the objects found in the workspace in a block form. When you group your objects they become associated under the "group" blocks. It is easier to navigate by selecting or deselecting objects using Hypergraph. Click on the group in the Hypergraph window which results in selection of the corresponding group in the workspace window. One can rename, hide or delete objects using Hypergraph.
10. Choose one of the groups and select the rotate tool from the tool box. You will see three circles of different color that, as with the moving tool, refer to the rotation around three axes.

Press the "Insert" key on the keyboard. The circles turn into a point—the pivot point. Use the arrows to move the pivot point from the middle of the bundle to the hinge axis that is found between helices 0 and 5. Using the four view window, place the rotating point on the outer edge of the helices 0 and 5.

11. Open the menu "Channel Box"—one of the Editor Icons found in the upper corner next to the caDNAno plug-in key (Fig. 8c). This menu provides you total control over the Move, Translate and Rotate operations. Depending on which part you have chosen first, it has to be rotated by either 90° or –90° around the *y*-axis. Repeat the operation for the other part.

    Our structure is now in the closed state (Fig. 8j). The next stage is to animate the object to the open state.

12. At the bottom of the window there are two sliders: the time slider and the range slider. The time slider shows which frame of the animation is currently shown, while the range slider tells how many frames there are in the animation and which part of it we are currently working on.

13. Change both numbers on the left of the range slider to 1 and on the right to 240 (Fig. 8e). Set the time slider to 1. Select one of the groups and press "S" on the keyboard—the red mark now appears on the first slide of the time slider. Select slide 240 on the time slider. Change the rotation around the *y* axis back to zero in the Channel Box and press "S" on the keyboard. Go back to the first frame on the time slider and press the play button in the lower right corner of Maya. You will see that one of the six-helix bundles is moving from the closed conformation to the open. Press stop and do the same for the other bundle.

14. Addition of fluorophore glow to the animation will make the function of the biosensor more visual. Go to slide 1 and click with the right button of the mouse on the red fluorophore. Choose "Material attributes . . ." from the drop-down menu. From the Material attributes menu scroll down to the Special effects function. Set glowing intensity to 1 and click with the right button of the mouse on it. Choose "Set driven key" from the drop-down menu. Choose one of the groups in Hypergraph and click on "Load Driver" in the set driven key menu. Choose rotate Y for the Driver and click "Key" in "Set driven Key." Click on the "Blinn" in the "Driven" and move the time slider to frame number 100. In the "Attribute editor" on the right set the glow to 0 and click again "Key" in "Set driven Key." The degree of glowing is now dependent on the bundle rotation around its Y axis. The glow effect is invisible until the frames are rendered. Do the same for the green fluorophore where glowing is set to 0 in the first frame and to 1 in the 100th frame.

15. The whole animation has to be rendered into a short movie sequence. Go to frame 1 and press the "Render Current Frame" button (Fig. 8d). A new window opens with a rendered frame of the device in the closed state: only the red fluorophore is glowing (Fig. 8k). Do the same for the frame 50 and 100 and you should get the frames shown in Fig. 8l, m. In the 100th frame the distance between fluorophores is too large and only the green fluorophore is glowing.
16. Choose the angle from which the animation is going to be viewed—by default the perspective view is rendered. Now we can render the first 100 frames and put them together in a small animation. Click the "Render Current Frame" button on a random frame. Choose Options > Render Settings . . .
    (a) Set "Render Using" to "Mental ray" software.
    (b) Name file sequence in the "File name prefix."
    (c) Set "Frame/Animation ext" to "name.#.ext."
    (d) Set "Image format" to "JPEG(jpg)".
    (e) "Frame padding" refers to the maximum size of the image sequence. Set it to 3 (it will run from 000 to 999, in total 1,000 images).
    (f) Set the "Start frame" to "1," "End frame" to "100" and "By frame" to "1". These options render each frame from 1 to 100.
    (g) Remember to note down the path to the directory where the rendered images will be saved.
17. Close the Render settings window and the Render window. Click "Render" on the menu bar and then "Batch render." One can follow the rendering process on the command line. It should take approximately 3–5 min to render 100 frames on a standard laptop computer. The sequence of 100 frames can be compiled into an animation. Close Maya. A PC user can use the already installed Windows Live Movie Maker while QuickTimePro for this purpose.
18. Open Windows Live Movie Maker and simply drag and drop your image sequence in the program. Go to the edit menu and change the duration of each frame to 0.05 s. Now save the movie and play it.

We have now created a 3D model and animation of a DNA origami device.

Here we have shown how to make a DNA origami blueprint in caDNAno and how to make a simple animation of a dynamical structure. The animation cannot be regarded as a simulation of the device but it makes it possible to get an idea about how the structure looks in 3D and how the dynamic parts might move in relation to each other. Furthermore, the animation of the idea can be successfully used when complicated scientific designs should be presented for the broader public but also for communication between scientists.

3.6 Suggestions for Further Work

Next you can start designing more complex shapes based on models from the literature (*see* **Note 11**), start considering how to functionalize the device by attaching RNA or protein elements (*see* **Note 12**) or order the designed structure, self-assemble, and characterize it (*see* **Note 13**).

4 Notes

1. To get comfortable with the various crossover patterns it is advised to use a spread sheet to calculate the positions of phosphates for the DNA helices involved in a crossover complex. Make two columns for each strand of the helix and specify the strand direction. Set the angle of the first crossover to zero and use the twist angle to calculate the phosphate position along the helix. Find the position where the DNA strands meet again on the same strand or the opposite strand and mark these positions with a color. Now try extending this to the multihelix DNA origami crossover patterns and verify that it fits the crossover spacings shown in Fig. 3.
2. A sketch is always a good start for designing a structure or drawing the initial idea but it should be also tried out in caDNAno to see if the structure design is possible. One of the first things to decide when designing a new structure using the DNA origami method is what lattice one would like to use. There are two main lattices that are widely used: square that was introduced in original Rothemund paper [3] and the honeycomb lattices [9]. The square and honeycomb lattices refer to the arrangement of helices in the structure (Fig. 4). The advantage of the square lattice is its compactness compared to the honeycomb lattice and can be used in construction of very dense structures. The angles between the helices in the honeycomb lattice result in close to integer numbers of bases between the crossovers that leads to less strained self-assembled structures. The square lattice spacing does not come so close to the integer numbers and is therefore more strained (Fig. 4). The square lattice can be manipulated to be more relaxed by insertion and deletion of bases in the helices to come closer to the natural DNA helicity of 10.44 bp/turn [10]. Beside these two lattices one can also use the less usual gridiron lattices [12] or curved scaffold path [11].
3. caDNAno takes that into account and shows the favorable crossovers for a selected helix with other helices in the lattice

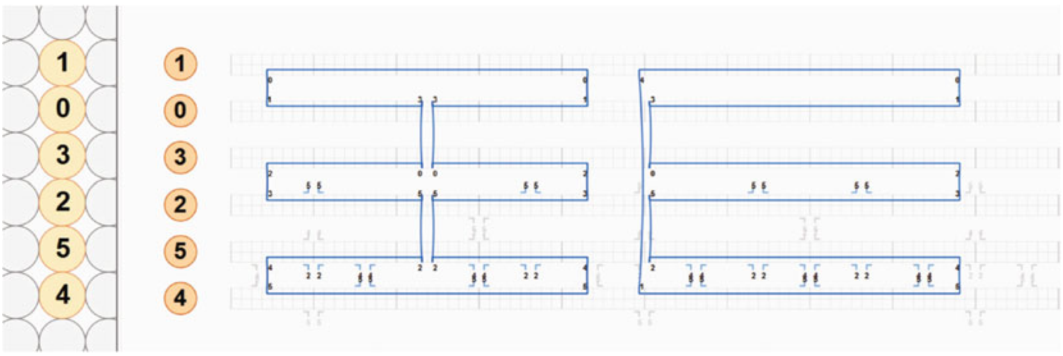


Fig. 9 Mid-seam and raster design strategies. Two blueprints proposed for rectangular DNA origami design. Blueprint on the *right* has fewer crossovers that only connect the ends of each helix with another. Blueprint on the *left* has extra crossovers that connect helices along the helix length—these crossovers are called "the seam"

as corners with numbers that show which helices are placed optimally for the crossover at that position (Fig. 7b). Crossovers are easily placed by clicking on these corners (Fig. 7c).

4. The scaffold that is used for DNA origami is usually a circular scaffold. That means that scaffold path should end where it starts. Imagine that you are designing a simple rectangular origami sheet. There are two ways that a scaffold can be wound through the structure using either raster or mid-seam design strategies (Fig. 9). The raster design proposed on the right has fewer crossovers that are only connecting the ends of each helix and includes a long crossover from helix 1 to 4 that should contain single stranded part long enough to span the length between these helices. The mid-seam design has crossovers not only placed on the ends of helices but also in between the helices. The crossovers connecting the scaffold helices inside the structure are called "the seam." The seam is another feature to consider before starting designing a DNA origami structure. The seam offers a possibility to introduce openings in the structure like the eyes and mouth of the original DNA origami smiley face [3]. The disadvantage of the seam is that it introduces breaks in the scaffold path that has to be linked with staple strands. A staple strand connecting such a break loses its long scaffold annealing region due to the break in the scaffold. The seam can be avoided by cutting the scaffold to make it linear [3]. The structure can also be designed so that it does not use or minimizes the use of the seam [30].
5. The average sequence distance between scaffold segments that are brought together by staple strands interaction is called contact order and was shown to have a big influence on self-assembly yield [31]. The maximum contact order that can be achieved depends on the design: 3D structures require smaller

contact order compared to 2D structures [3, 31]. This is especially important to have in mind when designing a structure, as introduction of the seam can reduce the contact order up to a factor of two. Consider our designs of rectangular origami sheet (Fig. 9). Both designs have the same dimensions of the structure but different contact order. Most helices are split into two parts in the mid-seam design that reduces contact order of these helices by a factor of two.

6. Another important step is the staple strand design. Staple strand design has been shown to have a great effect on self-assembly yield and stability of the structure [31, 32]. Designs where staple strands have only short 7 bp uninterrupted annealing regions on the scaffold strand do not self-assemble. Designs where staples have 11 or 12 bp stretches that anneal directly to the scaffold either do not assemble or assemble in a low yield. Designs that have the highest yields include the staple strand design where each staple has an at least 14 bp long uninterrupted region of complementarity to the scaffold and where most of the crossovers are preserved.
7. Addition of each base to a synthetic DNA oligonucleotide during chemical synthesis has a yield of 99 %. Longer oligos will result in lower quality. Therefore, staple strand lengths should be kept between 28 and 42 bp.
8. It is easier to model dynamical parts in Maya when hinge parts are not continuous—this can be done by introducing a break in the caDNAno design.
9. caDNAno2 also offers to show the position of the phosphates on the structure in the Maya window. Choose the menu: Edit > Modify mode. The black dots on the structure represent phosphate positions. These can be removed by pressing the "Modify mode" button once more.
10. Maya is commercial 3D animation software that offers a comprehensive creative feature set for 3D computer modeling, animation and simulation. As a student or educator one can get a free 3-year license.
11. Examples of DNA origami designs are found in the literature or on websites. There is currently no repository for such blueprint files, so one has either to reconstruct them in the program or ask the authors for their design files.
12. You can add molecular models of any structure from the Protein Data Bank (PDB) by using the Molecular Maya plug-in: http://www.molecularmovies.com/toolkit/. You can use the show phosphates option to find a spot where the protein can be attached on your structure.

13. In order to get the staple strand sequences for the designed structure one will need a scaffold strand sequence. Scaffold strand sequence is then imported into caDNAno using "Seq" function from the toolbox and clicking onto the scaffold strand. The scaffold strand may not have more than one break. The 5′ end of the scaffold will be a starting point of the sequence. One could use blueprint from Fig. 7e to introduce a break. Staple strand sequences will be generated automatically when the scaffold sequence is assigned. The staple strand sequences can now be exported to Excel readable csv format using "Export" function in the top menu. Every strand has a start and end coordinates for 5′ and 3′ ends respectively. The coordinates are given in the format xx[nn], where xx is a helix number and nn is a base number on the helix. One can easily trace the desired staple strand back to the blueprint using the coordinates. One can use movable ruler (Fig. 6f) for marking the right base number along the helix while helix number can be directly read (Fig. 6e). The sequence output provides the staple strand length and a specific color code. Color code can be used for sorting the sequences into separate modules for easier handling, e.g., lock strands and fluorophore strands. Staple strand sequence list can than be used for ordering oligonucleotides from a supplier of custom nucleic acids.

Acknowledgement

This work was supported by a Sapere Aude Starting Grant from the Danish Council for Independent Research (DFF-0602-01772) and the Centre for DNA Nanotechnology (http://cdna.au.dk/) funded by the Danish National Research Foundation (DNRF81).

References

1. Seeman NC (1982) Nucleic acid junctions and lattices. J Theor Biol 99(2):237–247
2. Seeman NC (2010) Nanomaterials based on DNA. Annu Rev Biochem 79:65–87
3. Rothemund PWK (2006) Folding DNA to create nanoscale shapes and patterns. Nature 440(7082):297–302
4. Qian L et al (2006) Analogic China map constructed by DNA. Chinese Science Bulletin
5. Andersen ES et al (2008) DNA origami design of dolphin-shaped structures with flexible tails. ACS Nano 2(6):1213–1218
6. Sharma J et al (2008) Toward reliable gold nanoparticle patterning on self-assembled DNA nanoscaffold. J Am Chem Soc 130 (25):7820–7821
7. Ke Y et al (2008) Self-assembled water-soluble nucleic acid probe tiles for label-free RNA hybridization assays. Science 319(5860): 180–183
8. Andersen ES et al (2009) Self-assembly of a nanoscale DNA box with a controllable lid. Nature 459(7243):73–76
9. Douglas SM et al (2009) Self-assembly of DNA into nanoscale three-dimensional shapes. Nature 459(7245):414–418
10. Dietz H, Douglas SM, Shih WM (2009) Folding DNA into twisted and curved nanoscale shapes. Science 325(5941):725–730
11. Han D et al (2011) DNA origami with complex curvatures in three-dimensional space. Science 332(6027):342–346

12. Han D et al (2013) DNA gridiron nanostructures based on four-arm junctions. Science 339 (6126):1412–1415
13. Han D et al (2013) Unidirectional scaffold-strand arrangement in DNA origami. Angew Chem Int Ed Engl 52(34):9031–9034
14. Wei B, Dai M, Yin P (2012) Complex shapes self-assembled from single-stranded DNA tiles. Nature 485(7400):623–626
15. Ke Y et al (2012) Three-dimensional structures self-assembled from DNA bricks. Science 338 (6111):1177–1183
16. Andersen ES (2010) Prediction and design of DNA and RNA structures. N Biotechnol 27 (3):184–193
17. Douglas SM et al (2009) Rapid prototyping of 3D DNA-origami shapes with caDNAno. Nucleic Acids Res 37(15):5001–5006
18. Williams S et al (2009) Tiamat: a three-dimensional editing tool for complex DNA structures. In: Goel A, Simmel F, Sosík P (eds) DNA computing. Springer, Berlin, pp 90–101
19. Zhu J et al (2009) UNIQUIMER 3D, a software system for structural DNA nanotechnology design, analysis and evaluation. Nucleic Acids Res 37(7):2164–2175
20. Pettersen EF et al (2004) UCSF Chimera—a visualization system for exploratory research and analysis. J Comput Chem 25(13):1605–1612
21. Kim D-NN et al (2012) Quantitative prediction of 3D solution shape and flexibility of nucleic acid nanostructures. Nucleic Acids Res 40(7):2862–2868
22. Zhang DY, Seelig G (2011) Dynamic DNA nanotechnology using strand-displacement reactions. Nat Chem 3(2):103–113
23. Zadeh JN et al (2011) NUPACK: analysis and design of nucleic acid systems. J Comput Chem 32(1):170–173
24. Ouldridge TE, Louis AA, Doye JP (2011) Structural, mechanical, and thermodynamic properties of a coarse-grained DNA model. J Chem Phys 134(8):085101
25. Ke Y et al (2009) Multilayer DNA origami packed on a square lattice. J Am Chem Soc 131(43):15903–15908
26. Andersen E, Nielsen M (2009) DNA origami design of 3D nanostructures
27. Castro CE et al (2011) A primer to scaffolded DNA origami. Nat Methods 8(3):221–229
28. Wang JC (1979) Helical repeat of DNA in solution. Proc Natl Acad Sci U S A 76(1): 200–203
29. Fu TJ, Seeman NC (1993) DNA double-crossover molecules. Biochemistry 32(13): 3211–3220
30. Douglas S, Bachelet I, Church G (2012) A logic-gated nanorobot for targeted transport of molecular payloads. Science 335(6070): 831–834
31. Ke Y, Voigt NV, Fradkov E, Shih WM (2012) Two design strategies for enhancement of multilayer–DNA-origami folding: underwinding for specific intercalator rescue and staple-break positioning. Chem Sci 3
32. Martin T, Dietz H (2012) Magnesium-free self-assembly of multi-layer DNA objects. Nat Commun 3:1103
33. Dirks RM et al (2004) Paradigms for computational nucleic acid design. Nucleic Acids Res 32 (4):1392–1403

Chapter 3

Computational Design of RNA Parts, Devices, and Transcripts with Kinetic Folding Algorithms Implemented on Multiprocessor Clusters

Tim Thimmaiah, William E. Voje Jr., and James M. Carothers

Abstract

With progress toward inexpensive, large-scale DNA assembly, the demand for simulation tools that allow the rapid construction of synthetic biological devices with predictable behaviors continues to increase. By combining engineered transcript components, such as ribosome binding sites, transcriptional terminators, ligand-binding aptamers, catalytic ribozymes, and aptamer-controlled ribozymes (aptazymes), gene expression in bacteria can be fine-tuned, with many corollaries and applications in yeast and mammalian cells. The successful design of genetic constructs that implement these kinds of RNA-based control mechanisms requires modeling and analyzing kinetically determined co-transcriptional folding pathways. Transcript design methods using stochastic kinetic folding simulations to search spacer sequence libraries for motifs enabling the assembly of RNA component parts into static ribozyme- and dynamic aptazyme-regulated expression devices with quantitatively predictable functions (rREDs and aREDs, respectively) have been described (Carothers et al., Science 334:1716–1719, 2011). Here, we provide a detailed practical procedure for computational transcript design by illustrating a high throughput, multiprocessor approach for evaluating spacer sequences and generating functional rREDs. This chapter is written as a tutorial, complete with pseudo-code and step-by-step instructions for setting up a computational cluster with an Amazon, Inc. web server and performing the large numbers of kinefold-based stochastic kinetic co-transcriptional folding simulations needed to design functional rREDs and aREDs. The method described here should be broadly applicable for designing and analyzing a variety of synthetic RNA parts, devices and transcripts.

Key words RNA devices, Co-transcriptional RNA folding, RNA secondary structure design, Ribozyme, Aptazyme, Ribosome-binding site (RBS), Kinefold

1 Introduction

Engineered transcript components, such as ribosome-binding sites (RBSs), transcriptional terminators, ligand-binding RNA aptamers, catalytic ribozymes and aptamer-regulated ribozymes (aptazymes), can be employed to control gene expression in bacteria, yeast, and mammalian cells [1–4]. Previously, we established a conceptual and experimental framework for engineering RNA-based genetic control devices and systems from component parts generated and

Mario Andrea Marchisio (ed.), *Computational Methods in Synthetic Biology*, Methods in Molecular Biology, vol. 1244,
DOI 10.1007/978-1-4939-1878-2_3, © Springer Science+Business Media New York 2015

characterized in vitro, in vivo, and in silico [5]. We formulated a model-driven process to engineer static, ribozyme-regulated expression devices (rREDs) and dynamic, metabolite-responsive, aptazyme-regulated expression devices (aREDs) with quantitatively predictable functions in *E. coli*. rREDs and aREDs have immediate usefulness as biosensors and controllers for engineered metabolic pathways and could serve as the bases for constructing very large and complex synthetic biological systems. Crucially, the physical implementation of functional devices in that work required the development of a novel method for designing transcripts with kinetic RNA folding simulations. The purpose of this chapter is to provide a practical description of that computational transcript design method.

Briefly, the underlying genetic control mechanism for engineered rREDs and aREDs relies upon ribozyme or aptazyme phosphodiester bond cleavage that generates 5′-OH-terminated mRNA (Fig. 1b). 5′-OH-terminated RNA is degraded in an RppH-independent (RppH−) mechanism that is characteristically slower

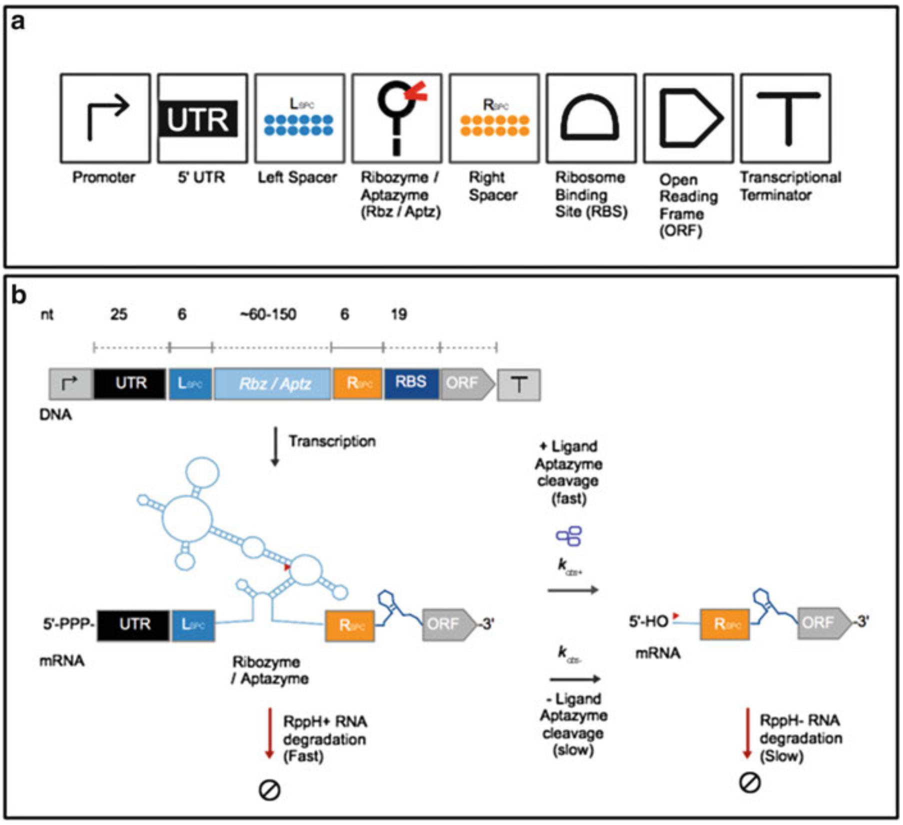

a
UTR
Promoter
5' UTR
Left Spacer
Ribozyme / Aptazyme (Rbz / Aptz)
Right Spacer
Ribosome Binding Site (RBS)
Open Reading Frame (ORF)
Transcriptional Terminator
b
nt 25 6 ~60-150 6 19
UTR
Rbz / Aptz
RBS
ORF
DNA
Transcription
+ Ligand Aptazyme cleavage (fast)
5'-PPP-
UTR
ORF
3'
mRNA
Ribozyme / Aptazyme
5'-HO
ORF
3'
mRNA
- Ligand Aptazyme cleavage (slow)
RppH+ RNA degradation (Fast)
RppH- RNA degradation (Slow)

Fig. 1 Ribozyme- and aptazyme-regulated expression devices (rREDs and aREDs). In this chapter, rREDs are designed to program static levels of quantitatively predictable genetic expression in *E. coli*, as in [5]. (**a**) Static rRED and dynamic ligand-controlled aRED schematic in SBOL format. (**b**) Schematic of the underlying rRED and aRED genetic control mechanism

than the rate with which the 5′ PPP-terminated mRNA is degraded by RppH-dependent (RppH+) pathways. As a consequence, phosphodiester bond cleavage increases mRNA stability, with a corresponding increase in protein expression that can be modeled and accurately predicted.

To assemble functional RNA devices from component parts, the design objective is to identify Spacer sequences (Fig. 1a) that enable both the ribozyme or aptazyme and the ribosome-binding site (RBS) to fold into the target secondary structures required for phosphodiester bond cleavage and efficient translation initiation, respectively [5]. Minimum free energy (MFE)-derived RNA secondary structure folding predictions are sufficient for many applications. However, the relatively short half-life of bacterial transcripts suggests that many mRNAs will not fold into MFE secondary structures in the context of the cell [6]. Consistent with that expectation, our prior work has shown that kinetic co-transcriptional RNA folding pathways must be evaluated in order to drive the design of rREDs and aREDs with predictable functions [5].

Sequence libraries flanking the insertion site of the ribozyme within the 5′ UTR of the device are computationally screened to identify spacers (e.g., L_{spc}) that enable the ribozyme and the ribosome-binding site (RBS) to fold into structures needed for efficient phosphodiester bond cleavage and translation initiation, respectively (Fig. 2). First, in Step A, the target secondary structure of the ribozyme (Rbz) or aptazyme (Aptz) part is determined using a simulated annealing folding simulation (note that the target secondary structure can also be specified—Fig. 2a). Next, in Step B the target structure(s) for the RBS in a reference device with a given set of spacer sequences is determined using stochastic, kinetic, co-transcriptional folding simulations (Fig. 2b). Finally, in Step C, the ribozyme or aptazyme sequence is inserted into the reference device transcript evaluated in the previous step (with a given set of spacer sequences) and the folding probabilities for the ribozyme and RBS parts are approximated using co-transcriptional folding simulations (Fig. 2c). Steps B and C can be iterated, as necessary, by varying the spacer sequences, until device transcripts are obtained that meet specific design criteria, as judged by the RBS and ribozyme or aptazyme folding probabilities determined in Step C.

We perform kinetic co-transcriptional folding simulations using the kinefold package created by the Isambert lab [7]. Kinefold is able to simulate the folds of secondary and pseudo-tertiary structures on minute timescales. Small numbers of stochastic, kinetic co-transcriptional folding simulations can be conducted with kinefold running on an individual computer or web server [7]. However, searching even small volumes of spacer sequence space requires a developed strategy for submitting, running, and analyzing very large numbers (i.e., tens of thousands) of computationally intensive stochastic RNA folding simulations (Fig. 3).

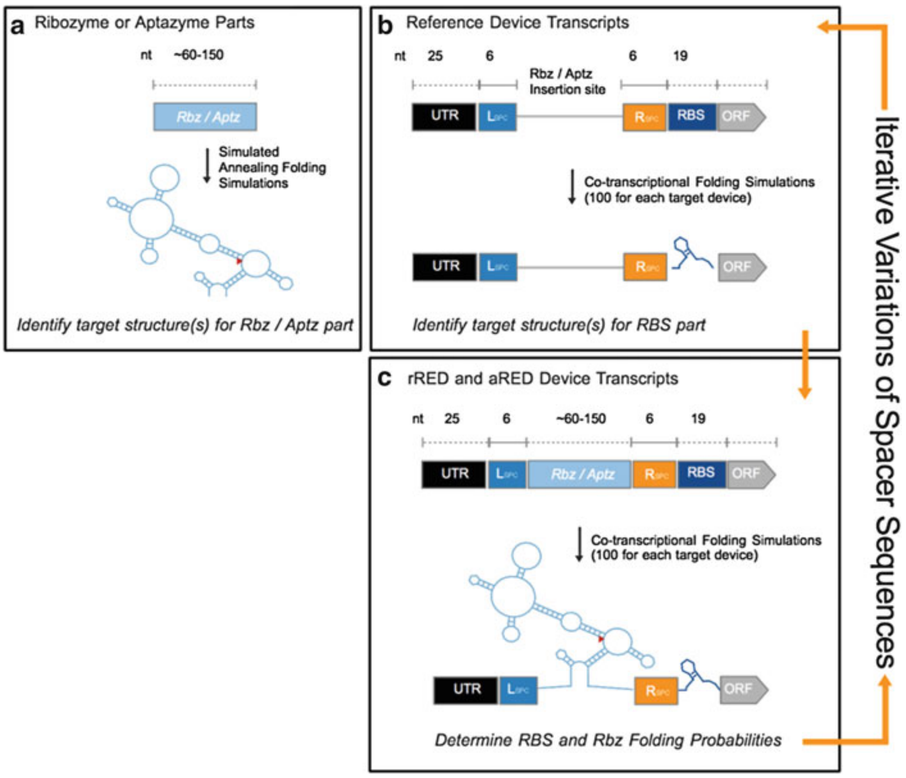

a Ribozyme or Aptazyme Parts
nt ~60-150
Rbz / Aptz
Simulated Annealing Folding Simulations
Identify target structure(s) for Rbz / Aptz part
b Reference Device Transcripts
nt 25 6 6 19
Rbz / Aptz Insertion site
UTR
RBS
ORF
Co-transcriptional Folding Simulations (100 for each target device)
Identify target structure(s) for RBS part
c rRED and aRED Device Transcripts
nt 25 6 ~60-150 6 19
Determine RBS and Rbz Folding Probabilities
Iterative Variations of Spacer Sequences

Fig. 2 Transcript design method. A schematic of the functional rRED and aRED transcript design algorithm is shown (*see* text for details)

The following tutorial details the methods required to deploy and utilize kinefold for high-throughput computational transcript design on a multiprocessor cluster system. To demonstrate the flexibility of this approach, we emphasize the incorporation of scalable, open-source services. Although we focus on the design of rREDs here, the computational framework that we describe for multiprocessor co-transcriptional folding simulation and analysis should be useful for the design of a wide variety of RNA parts, devices, and transcripts.

2 Materials

2.1 Dependencies

The following are the basic dependencies necessary to follow the tutorial in this chapter (*see* **Note 1**).

1. *Linux Based Amazon Elastic Compute Cloud* (http://aws.amazon.com/ec2) (*see* **Note 2**).
2. *MIT StarCluster* (http://star.mit.edu/cluster) (*see* **Note 3**).
3. *Kinefold Binary* (http://kinefold.curie.fr/) (*see* **Note 4**).

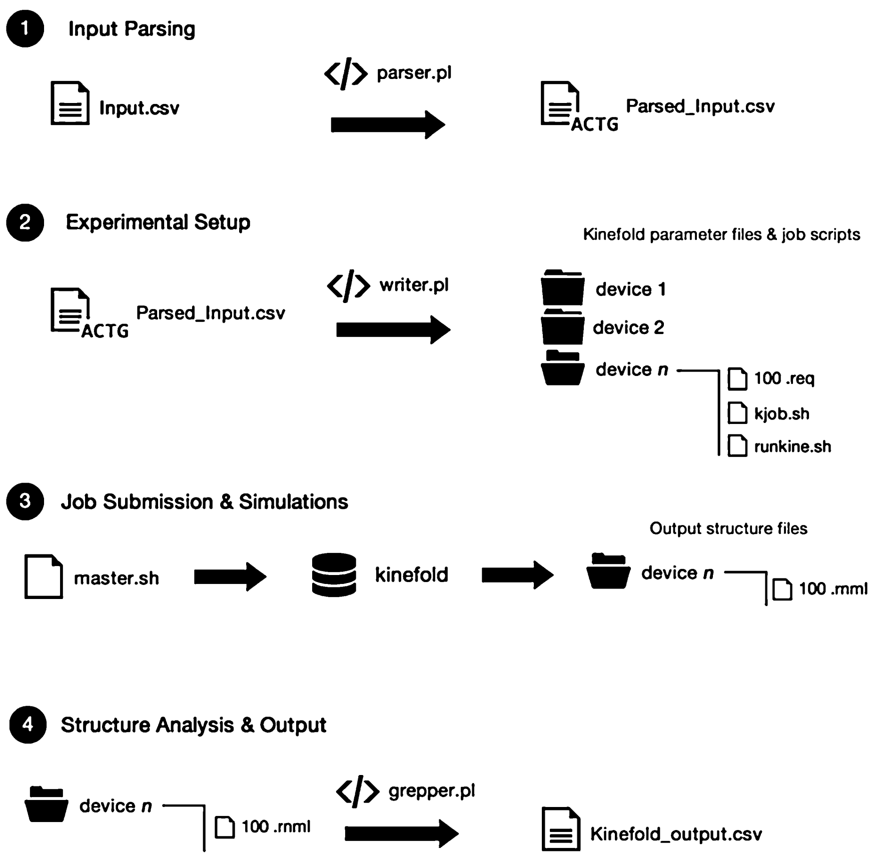


Fig. 3 Application structure and flow. A schematic outlining the computational setup and flow for the transcript design method described in this chapter

4. *BioPerl* (http://www.bioperl.org/) (*see* **Note 5**).
5. *Text::CSV_XS Perl Module* (recommended).

2.2 Installation and Setup

EC2 Cluster integration with MIT Starcluster (*see* **Note 6**).

(a) Sign up for an Amazon EC2 Account http://aws.amazon.com/console/#ec2.

(b) Install StarCluster on local machine (*see* **Note 7**).

(c) Adjust StarCluster Configuration.

- Specify Amazon EC2 Account credentials.

```
vi ~/.starcluster/config
```

  - Replace `aws_acces_key_id` and `aws_secret_acess_key` in `[aws info]` (*see* **Note 8**).

- Create access keys on the local machine to access the cluster system on EC2 directly:

```
starcluster createkey tdkey -o ~/.ssh/tdkey.rsa
starcluster help
2
vi ~/.starcluster/config
key_location=~/.ssh/tdkey.rsa
[key mykey] ->[key tdkey]
Keyname = tdkey
```

(d) Launch StarCluster on EC2:

```
starcluster start tdesign
starcluster sshmaster tdesign
```

(e) Install BioPerl on cluster (*see* **Note 9**).

(f) Install Text::CSV_XS (*see* **Note 10**).

2.3 Building a Framework for Submission

In the following subsections we will provide the framework for high throughput submissions of samples as well as processing of data. To accomplish this it will be necessary to write the following scripts:

parser.pl: will be used to process the input file

writer.pl: will be used to create the experimental directories and create job execution scripts

grepper.pl: will be used to process the output data

2.3.1 Application Hierarchy

Maintain the following hierarchy on the StarCluster based EC2 cluster:

```
~ /home
    /experiment_name
        /kinefold
            kinefold_binary_static
     input.csv
     parser.pl
     writer.pl
     grepper.pl
```

2.3.2 Parsing Genbank Files (parser.pl)

The following pseudocode outlines a function that parses the input file into a new csv that can be used to set up the experiment on the cluster (*see* **Note 11**):

```
csv_parse_genbank()
  open input file (input.csv) using Text::CSV_XS
  for each device create Bio::SeqIO Object
    get start & stop bp for part and window contexts
    convert DNA sequence objects to RNA sequence
    objects
    define simulation time (ms) (see Note 12)
    convert polymerase rate to ms/nt
  splice original input file to add RNA sequences, DNA
  sequences, simulation time
  create parsed-input.csv
```

2.3.3 Setting Up Directories (writer.pl)

The following pseudocode outlines how to set up simulation directories and construct the required job scripts for simulation submission (*see* **Note 13**):

```
kinefold_setup_dir()
  open parsed-input.csv
  splice out variables
    var deviceName
    var windowSequence
    var simulationType
    var kpolRate
    var simulationTime
    var ntAddTime
  for each device
    create directory (deviceNumber.deviceName)
      create .dat file
        write deviceName
        write windowSequence (RNA)
      create 100 .req files
        for each .req file
          write random seed (4 digit integer)
          add directory reference for kinefold output
          files
            .p, .e, .rnm, .rnms, .rnml, .rnm2
          add constants for .dat file
            0 (0 = RNA, 1 = DNA)
            6.3460741 (free energy kcal/mol)
            10000000
            simulationTime (ms)
            0 (Pseudoknots 1 = yes, 0 = no)
            0 (Entanglements 1 = yes,0 = no)
            simType ntAddTime (simType 1 = renaturation 2 =
            cotrans, ntAddTime ms/nt)
            deviceName
            deviceName.zip
    create jobscript (kjob.sh)
        write reference to working directory $WORKINGDIR
        write reference to .req file $REQFILE
        $WORKINGDIR/kinefold/kinefold_long_static
        $REQFILE -noprint
    create job execution script (runkine.sh)
        write reference to working directory
        $WORKINGDIR
        write reference to device folder $DEVDIR
        for each .req file
        qsub $WORKINGDIR/$DEVDIR/kjob.sh
    create master job submission script (master.sh)
        for each device execute job execution script
          (runkine.sh)
```

2.3.4 Submitting Jobs to Cluster

The master job submission script ("master.sh") will be used to submit a batch job for each device incorporating 100 folding simulations to EC2 StarCluster. The master job script executes a single job file ("runkine.sh") that loops through each .req file while passing the job ("kjob.sh") to the kinefold binary script in the top directory of your experiment (*see* **Note 14**).

2.3.5 Parsing Output (grepper.pl)

The following pseudocode outlines an algorithm to parse kinefold output data:

```
makeWindowStructures()

    for each device directory
        open each .rnml file
        grep out structure
        create a single window structure file with 100
        structures from 100
        simulations
makePartStructures()
    var uniquePartStructures = Array of N where N is the
    number of devices
    open parsed-input.csv
        splice out part start and part stop

    for each device directory
        open window structure file
        for each line in file
            grep out part structure from each window
            structure
        create concatenated part structure file for
        all 100 simulations
        filter part structure file > isolate unique
        part structures
        push count of unique part structures to unique-
        PartStructures
        Array for device N
```

The structures from this will be processed by a comparison function, as outlined below:

```
foldingComparison()

    var foldFrequency = Array of N of K Arrays where N is
    the number of devices and K is the number of unique
    part structures in reference device N

    open parsed-input.csv
        splice out device name
        splice out respective reference device for
        each device
    for each reference device for device N
```

```
        open the reference devices unique part struc-
        ture file
        for each reference devices unique part structure
            open the original devices window structure
            file
            count times unique part structure shows up
            in window structure
            file of device N
            push count of fold frequency to foldFre-
            quency Array for unique part structure K in
            device Ns reference device
    for two reference devices of device N
        open the first reference devices unique part
        structure file
        for each of the first reference devices unique
        part structure
            open the original devices window structure
            file
        push window structure of every match with the
        unique part structure in a temporary array
        open second reference devices unique part
        structure file
        for each of the second reference devices unique
        part structure
            count times each unique part structure
            shows up in the temporary array
            push count of fold frequency to foldFre-
            quncy Array for each combination of unique
            part structures K in the reference devices
            of device N
```

From the arrays of values created in the subfunctions above, create an output sheet concatenating all the data created from the starting input.csv:

```
makeOutputSheet()
    open parsed_input.csv
    for each device N
        insert total folding frequency
            sum of foldFrequency[N]
        insert number of unique part structures in
        device Ns reference devices
            uniquePartStructures[N]
        insert folding frequency of each unique part
        structure K of device Ns reference device
            foldingFrequency[N][K]
    concatenate output to parsed_input.csv as a new
file, Kinefold_output.csv
```

3 Methods

3.1 Experimental Design

The objective of this experiment is to identify spacer sequences that enable the correct folding of RNA regulatory devices within synthetic constructs. For the experiment a ribozyme will be introduced into the 5′ UTR of an mRNA. The proximity of the ribozyme to the ribosome-binding site (RBS) may cause structural changes to the RBS. Consequently, the secondary structure of both the ribozyme and the ribosome binding site will be evaluated to determine proper folding in the context of a simulated, elongating, transcript.

For this tutorial, we will first demonstrate the utility of the method by considering an already designed synthetic construct that integrates an *S. mansoni* hammerhead ribozyme (SMan) and two RBSs. The main point of this construct is the combination of the *S. mansoni* ribozyme and the first RBS, annotated as SD-RBS (1), required for assembly of a ribozyme-regulated expression device (rRED) engineered to produce a static level of gene expression. A second RBS, annotated as SD-RBS, is included here to investigate RNA part-folding independent of additional transcript control elements. Reference devices will serve as the positive controls for which expression levels have been characterized (*See* Introduction). The first reference construct, where SD-RBS (1) and SD-RBS are known to function, will be a model for identifying the RBS target structures. The second reference construct will be a model for identifying the target ribozyme structure.

Two implementations of kinefold are required to evaluate a given spacer sequence. The first is used to identify the target secondary structures of the RBS and ribozyme (Fig. 2b—*see* **Note 15**). The second is used to determine the probability that the RBS and ribozyme will fold in the context of the designed, synthetic construct (Fig. 2c—*see* Subheading 3.2.1). For this demonstration, a small library of spacer sequences will be considered, and only the 3′ spacer sequence will be varied (*see* **Note 16**). Here, the parts are treated as having folded correctly if the predicted part structure matches any member of the ensemble of predicted target structures.

To probe the space of possible folds for a given transcript sequence, 100 unique, stochastic, co-transcriptional folding simulations are run with random seeds for each of the expression devices and spacer sequence permutations. For each sequence x the predicted folding frequency (approximating a probability) f_x is taken as a fraction of elongating transcripts with RBS and Rbz subsequences $a = x_j x_{j+1} \ldots x_k$ folding into the target structure S_a at time t, where j is the starting base position of the window sequence and k is the ending position of the window sequence. The RNA polymerase elongation rate k_{pol} will be set to 25 nt s^{-1} with minimum helix energy $= -6.346$ kcal mol^{-1}. Simulation time for each stochastic

Table 1
Input for determining RBS and Rbz target structures

Genbank file name	Device name	Part start	Part stop	Window start	Window stop	Pol. elongation rate (nt/s)	Co-trans?	Time for addition of base (ms)	Ref. device
110916_ des1.gb	dev1.1	1,623	1,636	1,492	1,641	25	1 (yes)	40	dev2.1
110916_ des1.gb	dev1.2	1,535	1,616	1,492	1,641	25	1 (yes)	40	dev3.1
110916_ des1.gb	dev1.3	2,975	2,988	2,925	3,013	25	1 (yes)	40	dev2.2
110916_ ref1.gb	dev2.1	1,544	1,557	1,492	1,582	25	1 (yes)	40	dev2.1
110916_ ref1.gb	dev2.2	2,896	2,909		2,934	25	1 (yes)	40	dev2.2
110916_ ref2.gb	dev3.1	1	82	1	82	25	0 (no)	40	dev3.1

folding simulation is determined by $t_1 = [\text{len}(a))] / k_{\text{pol}}\ (s)$, where a is the length from the 5′ end to the RBS or Rbz center + (k_{pol} × 1 s) as 1 s is the shortest time for a ribosome to interact productively with an RBS [5].

3.2 Experimental Method

3.2.1 Determining Target Structures for RBS and Rbz

a. Input
Table 1 exemplifies the structure of an input file for the first part of the design experiment to determine RBS and Rbz target structures for the rRED device (*see* **Note 17**). All of the devices will undergo co-transcriptional folding simulations excluding the reference construct for the *S. mansoni* hammerhead ribozyme. Since this entire sequence is of the ribozyme itself, a renaturation fold type (snap cooling from 99 to 37 °C, 1 M NaCl) will be sufficient for identifying the respective reference structure (*see* **Note 18**).

b. Structure Files
After setting up the experiment on the cluster system and submitting simulation jobs, kinefold generates a collection of structure files. The .rnml structure files contain the final output structure in the minimum simulation time, determined by the length of the input sequence. The grepper script (Subheading 2.3.5) processes the data into two files for each device. The first contains the full output structure in the entire window context for each of the 100 simulations. The second contains only a subsection of the window structures respective to the part context and is filtered to include only unique part structures.

Table 2
Analysis of stochastic folding simulations for determining target RBS and Rbz structures

Device name	Feature	Total folding frequency	Number of unique structures	Structure 1 frequency (%)	Structure 2 frequency	Structure 3 frequency	Reference device
dev1.1	SD-RBS(1)	0.73	3	73	0 %	0 %	dev2.1
dev1.2	SMan Rbz	0.73	1	73	–	–	dev3.1
dev1.3	SD-RBS	1	1	100	–	–	dev2.2
dev2.1	SD-RBS	1	3	95	1 %	4 %	dev2.1
dev2.2	SD-RBS(1)	1	1	100	–	–	dev2.2
dev3.1	SMan Rbz	1	1	100	–	–	dev3.1

c. Folding Comparisons
The processed structure files are compared using the subfunction grepper.pl. Its output provides statistics for how well the parts in the reference devices fold in the context of a larger window sequence (Table 2). For example, we can observe that the RBS from the first reference device (exp2.1) folds into three unique structures. Out of these three structures, the first is the dominant structure, which folds with relatively high frequency (73 %) in the designed construct (dev1.1). The other RBS has a single structure which folds with 100 % frequency in the designed construct. The Rbz device also has a single structure that folds with 73 % frequency in the designed construct (*see* **Note 19**). By definition, the parts within the reference devices fold at 100 % frequency. The structures identified from the reference device transcripts are used as target structures for determining the frequency of part folding in the context of putative synthetic device transcripts (*see* Subheading 1).

3.2.2 Spacer Sequence Library

(a) Input
Any given combination of spacer sequences may not allow proper folding for a single device, however sparse samplings of random sequences may identify motifs that allow both RBS and ribozyme folding. Table 3 shows an example of left spacer library design (*see* **Note 20**). Co-transcriptional simulations are performed with the same parameters for polymerase elongation rate as in the first stage of the experiment and the previously determined RBS and ribozyme target structures (Table 2).

(b) Folding Comparisons
After setting up the experiment on the cluster and submitting the jobs for stochastic simulations, the methods for parsing

Table 3
Input for spacer sequence library for optimizing folding of target RBS and Rbz structures

Device Name	Feature	Lspacer	RBS part Start	RBS part Stop	Rbz part Start	Rbz part Stop	Window start	Window stop	Polymerase elongation rate (nt/s)	Co-transcriptional	Time for addition of base (ms)	Ref. device 1	Ref. device 2
exp1.1.1	RBS, SMan Rbz	ACTAGT	131	144	43	125	1	150	25	1	40	dev2.1	dev3.1
exp1.2.1	RBS, SMan Rbz	TGATCA	131	144	43	125	1	150	25	1	40	dev2.1	dev3.1
exp1.3.1	RBS, SMan Rbz	AGGCCT	131	144	43	125	1	150	25	1	40	dev2.1	dev3.1
exp1.4.1	RBS, SMan Rbz	GCTAGC	131	144	43	125	1	150	25	1	40	dev2.1	dev3.1
exp1.5.1	RBS, SMan Rbz	GTCGAC	131	144	43	125	1	150	25	1	40	dev2.1	dev3.1

Table 4
Observed folding frequencies of RBS and Rbz folding together with varying L_{spc} arrangements

		Part folding frequency		
Device name	LSpacer	SD-RBS (%)	SMan Rbz (%)	Simultaneous folding frequency (%)
exp1.1	ACTAGT	42	41	41
exp1.2	TGATCA	43	43	43
exp1.3	AGGCCT	94	5	5
exp1.4	GCTAGC	11	31	10
exp1.5	GTCGAC	45	44	43

the folding frequencies (i.e., probabilities) can be re-implemented from Subheading 3.2.1. Since the reference devices are from the previous experiment, the same unique part structure files can be used for the folding comparison function (*see* Subheading 2.3.4). The output, shown in Table 4 demonstrates that different part folding probabilities are obtained for each of the five L_{spacer} variations.

4 Notes

1. This tutorial will cover how to set up a personal computational cluster system with resources that are publicly and freely accessible. However, the framework of this tutorial is easily adaptable to most linux-based computational clusters.
2. The Amazon EC2 is a highly scalable web server that provides resizable computing capacity on the cloud offering multiple operating systems. You may use as large of a Linux cluster instance as your work requires, but this tutorial will employ the smallest instance available, the t1.micro.instance. If the cluster does not have Perl pre-installed, it must be added.
3. StarCluster is an open-source cluster computing toolkit developed specifically for the Amazon Elastic Compute Cloud. StarCluster allows creation, configuration, and management of the cluster system on a virtual machine through Amazon EC2. A cluster environment which allows batch job submission is required for high throughput processing of stochastic folding simulations.
4. The kinefold Linux executable binary, developed by the Isambert Lab, is implemented to perform individual co-transcriptional folding simulations.

5. The BioPerl distribution, developed in Perl, is a toolkit for performing several computational tasks in biology. It is required to read sequence files, translate sequences, and parse sequences for simulations.
6. The StarCluster system will automatically set up Amazon Elastic Block Storage (EBS) volumes for the EC2 cluster for persistent storage of data. It is possible to disable this service to avoid accruing cost. In this case, be sure to enable data sharing across the network file system (NFS), so that the master and slave nodes may exchange data. Regardless, it is recommended that you choose to run your experiment from a directory that is shared via NFS.
   (a) Start cluster installation instructions can be found here http://star.mit.edu/cluster/docs/latest/installation.html. It is recommended to install using the distributed version control and repository client git from a downloaded snapshot or with the git command utility. (http://git-scm.com/).
   (b) Any files such as input files or output files can be moved back and forth from EC2 cluster using the following commands:
      - starcluster put tdesign /path/to/local/file/or/dir /remote/path/
      - starcluster get tdesign /path/to/remote/file/or/dir /local/path/
7. This information is found under Account > Security Credentials within the Amazon Web Services Console.
8. In this configuration script, square brackets are references to the variable described in context.
9. BioPerl is easily installed on the cluster using CPAN:
   (a) perl -MCPAN -e shell
   (b) cpan > d /bioperl/
   (c) cpan > install CJFIELDS/BioPerl-1.6.1.tar.gz
10. This module may be easily installed on the cluster using CPAN:
    (a) perl -MCPAN -e shell
    (b) cpan > install Text::CSV_XS)
11. BioPerl is only required if Genbank files are used as an input for sequences.
12. Simulation time for each stochastic folding simulation is determined by $t_1 = [\mathrm{len}(a))]/k_{\mathrm{pol}}\ (s)$, where a is the size of the window sequence, plus any additional time that is desired by the experimenter. k_{pol} represents the average *E. coli* polymerase elongation rate (set to 25 nt/s).

13. To simplify the job submission and the output parsing processes, directories are created to contain the simulations and cluster output for each device independently. Each device directory (e.g., device 1, device 2) will contain 100 .req files representing 100 folding independent, stochastic, simulations.
14. Make sure that the working directory is located under /home where all resources are NSF shared across all master and slave nodes on cluster:
    (a) cd /home
    (b) bash master.sh
15. The Genbank plasmid files for use in this tutorial can be located in the JBEI Public Registry (https://public-registry.jbei.org). Part IDs for the synthetic and two reference constructs are JPUB_001234, JPUB_001233, and JPUB_001232, respectively.
16. A rigorous implementation of this method would consider a set of randomly varied left and right spacer sequences, with further iterations holding either the left or right spacer constant to iterate toward the optimal design. There are 4,096 possible combinations for a six-nucleotide sequence; therefore one should consider ways to limit the possible design space. For example, designing around restriction sites that will help for sub-cloning.
17. We recommend providing inputs as a human readable .csv file. This enables quick and easy edits to experimental design and easy organization of input Genbank sequence files, device nomenclature, and corresponding parameters that kinefold needs to run the stochastic folding simulations.
18. The kinefold binary can be used to perform two types of folding simulations, both with fixed parameters of 37 °C with 1 M NaCl and no divalent ions. A "Renaturation Fold" refers to an RNA folding simulation starting from a completely denatured state, whereas a "Co-transcriptional fold" refers to a process where RNA folding proceeds simultaneously with transcript elongation, with a rate that can be specified by the user. The current length limitations of folding are 400 bases for both Renaturation and Co-transcriptional folding [7].
19. The kinefold binary returns several output files in each device design directory. The ".rnml" files contain the final secondary structures used to determine the structure of the components of interest.
20. Because the co-transcriptional folding simulations are stochastic, there will be statistical variation in the computed folding frequencies and probabilities. Additional runs can be performed to estimate statistical properties of the outputs.

References

1. Win MN, Smolke CD (2009) A modular and extensible RNA-based gene-regulatory platform for engineering cellular function. Proc Natl Acad Sci 106:14283–14288. doi:10.1073/pnas.0908785106
2. Lou C, Stanton B, Chen Y-J, Munsky B, Voigt CA (2012) Ribozyme-based insulator parts buffer synthetic circuits from genetic context. Nat Biotechnol 30:1137–1142. doi:10.1038/nbt.2401
3. Khalil AS, Collins JJ (2010) Synthetic biology: applications come of age. Nat Rev Genet 11:367–379. doi:10.1038/nrg2775
4. Medema MH, van Raaphorst R, Takano E, Breitling R (2012) Computational tools for the synthetic design of biochemical pathways. Nat Rev Microbiol 10:191–202. doi:10.1038/nrmicro2717
5. Carothers J, Goler J, Juminaga D, Keasling J (2011) Model-driven engineering of RNA devices to quantitatively program gene expression. Science 334:1716–1719
6. Selinger DW, Saxena RM, Cheung KJ, Church GM, Rosenow C (2003) Global RNA half-life analysis in Escherichia coli reveals positional patterns of transcript degradation. Genome Res 13:216–223. doi:10.1101/gr.912603
7. Xayaphoummine A, Bucher T, Isambert H (2005) Kinefold web server for RNA/DNA folding path and structure prediction including pseudoknots and knots. Nucleic Acids Res 33: W605–W610. doi:10.1093/nar/gki447

Chapter 4

Regulatory RNA Design Through Evolutionary Computation and Strand Displacement

William Rostain, Thomas E. Landrain, Guillermo Rodrigo, and Alfonso Jaramillo

Abstract

The discovery and study of a vast number of regulatory RNAs in all kingdoms of life over the past decades has allowed the design of new synthetic RNAs that can regulate gene expression in vivo. Riboregulators, in particular, have been used to activate or repress gene expression. However, to accelerate and scale up the design process, synthetic biologists require computer-assisted design tools, without which riboregulator engineering will remain a case-by-case design process requiring expert attention. Recently, the design of RNA circuits by evolutionary computation and adapting strand displacement techniques from nanotechnology has proven to be suited to the automated generation of DNA sequences implementing regulatory RNA systems in bacteria. Herein, we present our method to carry out such evolutionary design and how to use it to create various types of riboregulators, allowing the systematic de novo design of genetic control systems in synthetic biology.

Key words Bacterial riboregulator, Computational design, Energy model, Posttranscriptional regulation, Synthetic biology, Strand displacement

1 Introduction

RNA is a versatile molecule, and its many roles in the cell include enzyme-like activity and regulation, in addition to its various roles in translation [1]. Some of RNA's many mechanisms for modulating biological processes have been harnessed by synthetic biologists for creating artificial regulators of gene expression [2, 3]. Examples in bacteria or yeast involve positive and negative riboregulators [4–9], ribozyme-based systems [10–13], or CRISPR systems [14–16]. These regulatory RNAs can control gene expression through base pairing with a messenger RNA (mRNA) or DNA, and typically have a defined secondary structure that ensures stability and functionality (mainly for interaction).

In this work, we focus on bacterial small RNAs (sRNAs) that can interact with the 5′ untranslated region (5′ UTR) of a given

Table 1
List of engineered riboregulators of gene expression

System	Sequence[a]	Fold change[b]	References
RR12	>sRNA ACCCAAATCCAGGAGGTGATTGGTAGTGGTGGTTAA TGAAAATTAACTTACTACTACCATATATCTCTAGA >5′ UTR GAATTCTACCATTCACCTCTTGGATTTGGGTATTAAA GAGGAGAAAGGTACCATG	10–20 (activator)	[4]
RAJ11	>sRNA GGGAGGGTTGATTGTGTGAGTCTGTCACAGTTCAGC GGAAACGTTGATGCTGTGACAGATTTATGCGAGGC >5′ UTR CCTCGCATAATTTCACTTCTTCAATCCTCCCGTTAAA GAGGAGAAATTATGAATG	10–20 (activator)	[7]
RAJ12	>sRNA GGGCAGGAAGAAGGGTTCCTTTGAGCGAATCTAGC GGCACCTCGCTAGGATTTGCTCGAAGGGATTCTGGG >5′ UTR ACCCAGTATCATTCTCTTCTTCCTGCCCACGCGGAAA GAGGAGAAAGGTGTAATG	~10 (activator)	[7]
IS10	>sRNA TCGCACATCTTGTTGTCTGATTATTGATTTTTCGCGA AACCATTTGATCATATGACAAGATGTGTATCCACCT TAACTTAATGATTTTTACCAAAATCATTAGGGGATTC ATCAG >5′ UTR GCGAAAAATCAATAAGGAGACAACAAGATG	~10 (repressor)	[8]
MicC-like B1	>sRNA ATGACGTTCTCACTGCTCGCCATATATTTGTCTTTCT GTTGGGCCATTGCATTGCCACTGATTTTCCAACATA TAAAAAGACAAGCCCGAACAGTCGTCCGGGCTTTTT TTCTCGAG >5′ UTR AAGGAGGACAAATATATGGCGAGCAGTGAGAACGT CAT	10–50 (repressor)	[9]

[a]Shine-Dalgarno box and start codon *underlined*
[b]Apparent fold change by fluorometry at the population level

mRNA (*see* Table 1). Using computational tools, from which RNA secondary structures and base pairing energies and probabilities can be predicted [17, 18], in combination with in silico evolutionary design principles [19, 20], various RNA systems can be engineered. Although synthetic RNAs could be designed *by hand*, fast and large-scale engineering of complex RNA circuits for synthetic biology cannot be based on this lengthy, case-by-case design strategy. To attain this goal, computer-assisted design will be required to accelerate the design process [7, 12].

Automated design has been successful for some types of RNAs, as well as for nucleic acids in general, proteins, and circuits (applications reviewed in ref. 19), allowing the diffusion of these methods as general tools for molecular biology. Evolutionary computation of riboregulation starts with two RNA sequences and iterates alternative rounds of mutation and selection of species that show the desired structural characteristics. This method adapts strand displacement techniques from nanotechnology and it has successfully been applied to the de novo design of bacterial riboregulators [7]. It has the advantage of being based on physiochemical principles, assessing all specifications and constraints of the design process in silico and without manual inspection, and it can be tailored to the design of simple or complex riboregulation [21].

In the following pages, we detail the strategy used for creating a piece of software capable of performing the evolutionary computation of two interacting sRNAs. This involves creating an objective function used to score RNA sequences and to determine whether they will show the desired behavior, together with a mutation operator used to efficiently search the sequence space. We then explain how to use this method to create positive and negative riboregulation [2], providing along the way some tips and resources for the design, implementation, and characterization of such systems. We expect this approach will be of value for engineering genetic circuits in vivo and for increasing our ability to design more sophisticated regulatory systems.

2 Materials

2.1 Software

To computationally design regulatory RNA systems, which perform a particular logical operation, a combinatorial optimization algorithm is constructed, in which thermodynamic and structural parameters are used to evaluate a system at any time during the optimization (*see* Fig. 1). To estimate those parameters, the ViennaRNA package [22] is used. The different intra- and intermolecular secondary structures and the corresponding free energies are thus easily calculated throughout the evolutionary process (*see* Subheading 3.1.2). Once the solutions have converged, the Nupack package [23] is used to carry out a post-analysis of the designed sequences. This independent software serves as a subsequent filtering and validation tool (*see* Subheading 3.1.4).

2.1.1 ViennaRNA (Version 2.0)

1. Download the ViennaRNA package from http://www.tbi.univie.ac.at/~ronny/RNA/index.html.
2. Follow the supplied instructions for installation. There is no need for additional libraries.

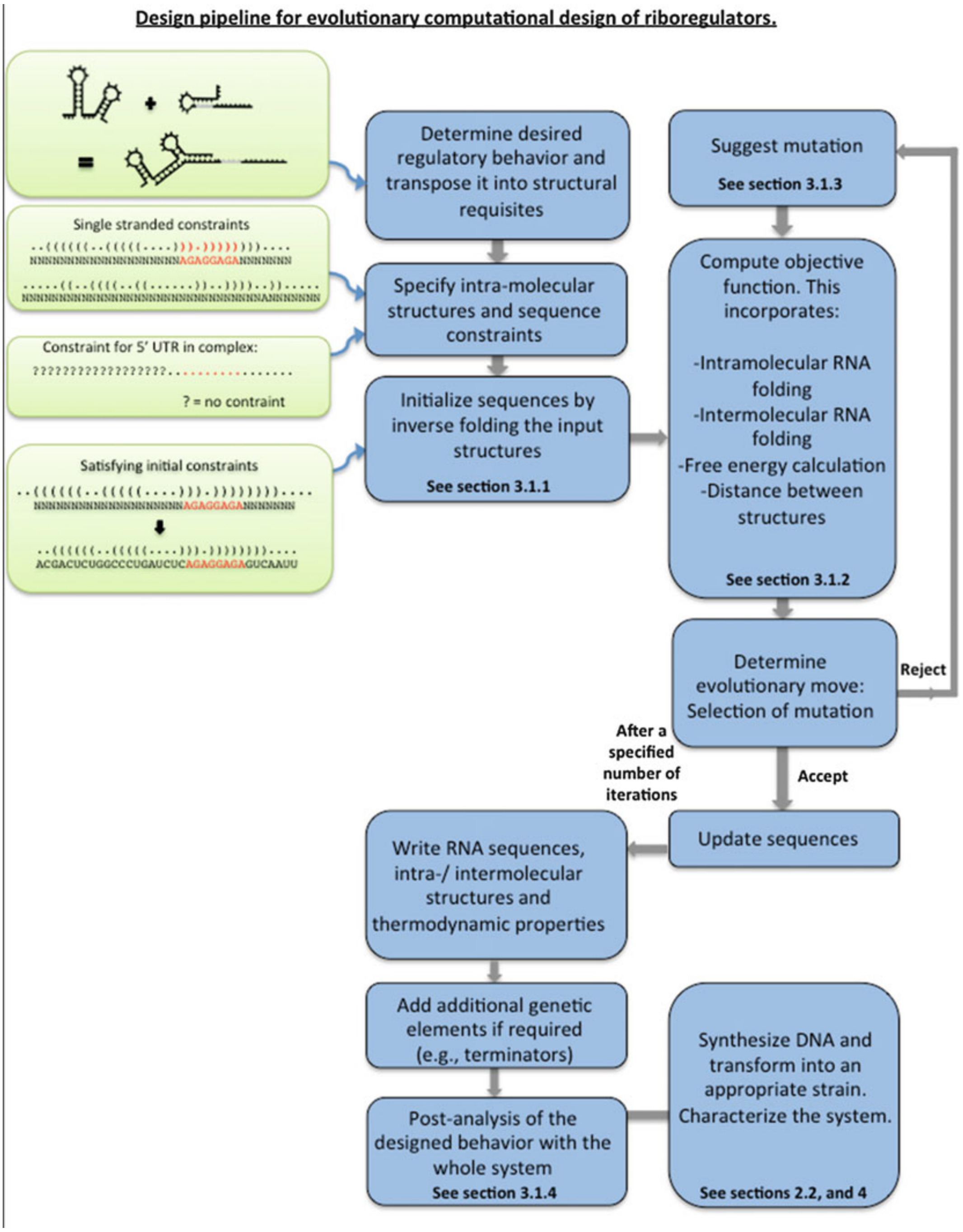

Design pipeline for evolutionary computational design of riboregulators.
Single stranded constraints
..((((((..(((((....))).))))))))....
NNNNNNNNNNNNNNNNNNNNAGAGGAGANNNNNNN
.....((..((((..((.......))..))))..)).....
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNANNNNNNN
Constraint for 5′ UTR in complex:
???????????????????.................
? = no contraint
Satisfying initial constraints
..((((((..(((((....))).))))))))....
NNNNNNNNNNNNNNNNNNNNAGAGGAGANNNNNNN
..((((((..(((((....))).))))))))....
ACGACUCUGGCCCUGAUCUCAGAGGAGAGUCAAUU
Determine desired regulatory behavior and transpose it into structural requisites
Specify intra-molecular structures and sequence constraints
Initialize sequences by inverse folding the input structures
See section 3.1.1
Suggest mutation
See section 3.1.3
Compute objective function. This incorporates:
-Intramolecular RNA folding
-Intermolecular RNA folding
-Free energy calculation
-Distance between structures
See section 3.1.2
Determine evolutionary move: Selection of mutation
Reject
Accept
After a specified number of iterations
Update sequences
Write RNA sequences, intra-/ intermolecular structures and thermodynamic properties
Add additional genetic elements if required (e.g., terminators)
Post-analysis of the designed behavior with the whole system
See section 3.1.4
Synthesize DNA and transform into an appropriate strain. Characterize the system.
See sections 2.2, and 4

Fig. 1 RNA circuit design pipeline. Automatic evolutionary design uses secondary structure scaffolds as a starting point to generate random starting RNAs. These are then sequentially mutated and selected to evolve them towards a solution which satisfies a user-defined behavior

Table 2
List of useful promoters for riboregulation

Name	Sequence[a]	Regulation	References
PLlacO1	AATTGTGAGCGGATAACAATTGACATTGTGAGC GGATAACAAGATACTGAGCACA	Repressed by LacI	[24]
PLtetO1	TCCCTATCAGTGATAGAGATTGACATCCCTATC AGTGATAGAGATACTGAGCACA	Repressed by TetR	[24]
J23119	TTGACAGCTAGCTCAGTCCTAGGTATAATGCT AGCA	Constitutive	[25]
T7-Pro	GCGAAATTAATACGACTCACTATAGGG	Transcribed by T7 RNAP	[26]

[a]Position +1 *underlined*

3. Use the following functions within the ViennaRNA package: (1) *fold*, to predict minimum free energy secondary structures; (2) *cofold*, as fold but for two different species; and (3) *inverse_fold*, to get sequences with predefined structures.

2.1.2 Nupack (Version 3.0)

1. Download the Nupack package from http://www.nupack.org/downloads/source.
2. Follow the supplied instructions for installation. There is no need for additional libraries.
3. Use the following functions within the Nupack package: (1) *concentrations*, to predict the equilibrium concentration of each species (single or complex) in a dilute solution given initial concentrations of the single ones, and (2) *subopt*, to determine all possible structures of the thermodynamic ensemble within an energy gap, and to check for structural robustness.

2.2 Necessary Elements for in Vivo Implementation and Expression

2.2.1 Promoters

1. Use well-characterized constitutive or inducible promoters and choose them accordingly to the genetic background (*see* Table 2 for a list of useful promoters; *see* also **Note 1**).
2. It is essential that the chosen promoters have a known and characterized transcription start site (+1 position) to avoid truncation or alteration of the RNA sequence. Pay special care when using a promoter truncated to the transcription start site because the RNA sequence may affect the transcription initiation.

2.2.2 Transcription Terminators

1. Transcription terminators are placed just downstream of the sRNA and of the coding sequence. Favor short rho-independent terminators with strong activity (i.e., with high free energy and long poly(U) tail; *see* Table 3 for a list of useful terminators in riboregulation).

Table 3
List of useful transcription terminators for riboregulation

Name	Sequence	Strength[a]	References
T500	CAAAGCCCGCCGAAAGGCGGGCTTTTCTGT	98 %	[27]
his-Ter	GCCCCCGGAAGATCACCTTCCGGGGGCTTTTTTATT	97 %	[28]
B1006	AAAAAAAAACCCCGCCCCTGACAGGGCGGGGTTTTTTTT	98 %	[28]
T7-Ter	TAGCATAACCCCTTGGGGCCTCTAAACGGGTCTTGAGGG GTTTTTTG	90–95 %	[26]
aspA-Ter[b]	AAATAAAAAAGGCACGTCAGATGACGTGCCTTTTTTCTT	98 % (fwd) 99 % (rev)	[29]
fur-Ter[b]	AACGAGAAAAGCCAACCTGCGGGTTGGCTTTTTTATGCA	95 % (fwd) 93 % (rev)	[29]

[a]Calculated experimentally
[b]Reversible

Table 4
List of useful *E. coli* strains for studying riboregulation

Strain	Genotype	References
MGZ1	MG1655, *lacIq*, PN25:*tetR*	[31]
HL770	MG1655, *lacIq*, *Δhfq*	[32]
JW2798	BW25113, *ΔrppH*	[33]
BL21(DE3)	B, *lacIq*, lacUV5:*T7 RNAP*	[26]

2. Ideally, the computational design process should take into account the terminator sequence and structure, especially for the riboregulator. The terminator sequence can be specified as a sequence constraint. Alternatively, known toy models of transcription termination [29, 30] could be quantitative and predictive enough to be incorporated in the objective function.

In addition, the expression of designed circuits in different genetic backgrounds depends on the characteristics of the engineered system and the functional properties one wants to study. To this end, several *E. coli* strains which are particularly suitable for RNA synthetic biology can be used (*see* Table 4).

2.2.3 Strains (Cellular Chassis)

1. The strain MGZ1 [30] is very useful for characterizing circuits. It constitutively expresses the transcription factors TetR and LacI (*see* also **Note 2**), enabling a control over promoters containing the relevant operators, like PLlacO1 and PLtetO1 [24].

Using IPTG and aTc, the level of expression of both RNA molecules can be finely tuned (*see* also **Note 3**).

2. In order to alter the function of an RNA system, various strains that are depleted in factors that are known to affect RNA in vivo can be used. These include (1) RNase-deficient strains such as HT115 (Δ*rnc*, deleting RNase III) [34] or BL21 Star (mutated *rne*, making RNase E less efficient; Invitrogen), (2) co-factor-deficient strains such as HL770 (Δ*hfq*) [32], or (3) strains deficient in RNA-processing proteins such as JW2798 (Δ*rppH*) [33].

2.2.4 Plasmids

1. The RNA components can be together on one single plasmid, or separated on two different ones for co-transformation (*see* also **Note 4**).
2. Favor high copy number plasmids, with which the response is easier to detect due to concentration effects (presumably, the effective dissociation constant between two synthetic RNAs is high).

3 Methods (Computational Design)

In this section, we first describe the general strategy for implementing an automated design method. We formulate a combinatorial optimization problem to which we apply an evolutionary algorithm. In our case, this is based on Monte Carlo simulated annealing [35]. Then, we describe in detail how to use such an approach to design riboregulators acting, either as repressors or activators, on target 5′ UTRs (*see* Table 1 for experimentally verified examples). Finally, we describe ways to create more sophisticated systems.

3.1 General Strategy of Evolutionary Design

Evolutionary design of synthetic RNA systems uses alternative rounds of computational assessment of RNA-RNA interaction followed by mutation [7]. Figure 1 outlines the general design pipeline. After initialization of the process, an objective function is used to evaluate the structures and free energies of RNAs, and a mutation operator modifies nucleotides or base pairs whilst keeping structural constraints for each species over the course of the evolutionary process. These rounds of mutation, scoring and selection are continued for a specified number of iterations or until a satisfactory solution is found. Afterwards, the sequences are output, and reviewed with an independent piece of software, and assembled for characterization (*see* Fig. 2).

3.1.1 Specifications and Initiation

The desired regulatory behavior is converted into (intra- and intermolecular) structure specifications that can be read by a computer, coded in dot-bracket notation. For examples of specifications that

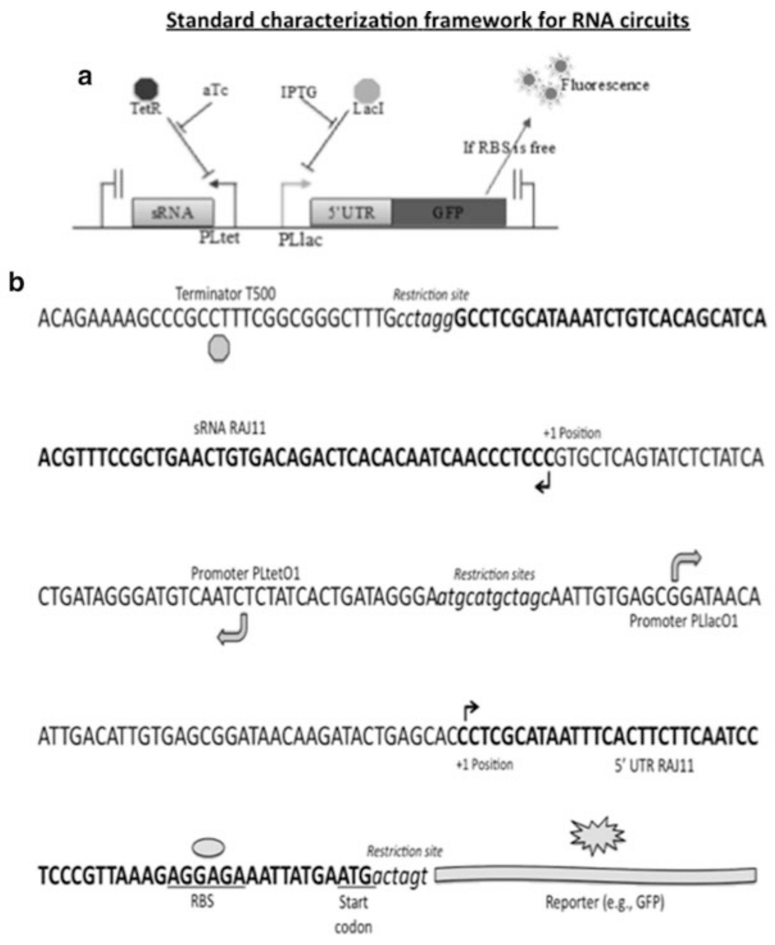

Standard characterization framework for RNA circuits
a
TetR
aTc
IPTG
LacI
Fluorescence
If RBS is free
sRNA
PLtet
PLlac
5'UTR
GFP
b
Terminator T500
Restriction site
ACAGAAAAGCCCGCCTTTCGGCGGGCTTTGcctaggGCCTCGCATAAATCTGTCACAGCATCA
sRNA RAJ11
+1 Position
ACGTTTCCGCTGAACTGTGACAGACTCACACAATCAACCCTCCCGTGCTCAGTATCTCTATCA
Promoter PLtetO1
Restriction sites
CTGATAGGGATGTCAATCTCTATCACTGATAGGGAatgcatgctagcAATTGTGAGCGGATAACA
Promoter PLlacO1
ATTGACATTGTGAGCGGATAACAAGATACTGAGCACCCTCGCATAATTTCACTTCTTCAATCC
+1 Position
5′ UTR RAJ11
Restriction site
TCCCGTTAAAGAGGAGAAATTATGAATGactagt
RBS
Start codon
Reporter (e.g., GFP)

Fig. 2 (**a**) Characterization framework for an RNA circuit and (**b**) Example of an assembled synthetic RNA device (here, the RAJ11 riboregulator and its 5′ UTR target controlling GFP), within an operon containing controllable promoters (PLlac and PLtet), terminators and with restriction sites for assembling controls or changing the reporter

can be used, *see* Subheadings 3.2 and 3.3 describing the design of negative and positive riboregulation, respectively. The definition of such structural specifications, as well as the sequence constraints, is a sensitive step, as the whole design process depends on it. The intramolecular structure specifications could be considered as constraints if desired, although this is not necessary, then the optimization of sequences would be performed over neutral networks in structure [7]. The free energies of hybridization and activation of the system and both the intra- and intermolecular structures of the 5′ UTR are used to evaluate our objective function.

1. Specify intramolecular structural specifications for each RNA species. The 5′ UTR and the sRNA can be fully constrained.

Table 5
Examples of structure and sequence specifications for designing positive (system RAJ11) and negative (system RAJ-N[a]) riboregulation

Sequence	Specified structure[b] and sequence contraints[c]	References
5′ UTR RAJ11 intramolecular	......((((((((.((((((.(((......))).)))))).))))))))..... NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNAGGAGANN NNNNNNATG	[7]
5′ UTR RAJ11 intermolecular	??????????????????????????????????............	[7]
sRNA RAJ11 intramolecular	...........((((((((((((((((((((((((((....)))))).))))))))))))))))))))... NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN NNNNNNNNN NNNNNNNNNNNNNNNN	[7]
5′ UTR RAJ-N intramolecular	.... NNNNNNNNNNAGGAGANNNNNNNNATG	[21]
5′ UTR RAJ-N intermolecular	??????||||||||||???????????	[21]
sRNA RAJ-N intramolecular	...(((((((.....)))))))...(((((((((((((.......))))))).)))))).((((((((((.....)))))))))).. NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN NNNNNNNNNN NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN	[21]

[a]We call here RAJ-N the system between RNAs cR01 and tR31 from ref. [21]
[b]? indicates unspecified
[c]Sequence constraints are only associated here to intramolecular structures

Table 5 provides some usable structures, and natural or synthetic riboregulatory devices can inspire others.

2. Specify a conformational pattern for the complex species, where the ribosome-binding site (RBS) should be the only region subject to structural specifications, whilst the rest of the sequence of the 5′ UTR and the whole sRNA should have a large degree of freedom.
3. Specify sequence constraints, by defining nucleotides that are not allowed to mutate during the evolutionary process. This is essential in order to ensure that important RNA motifs such as RBSs or aptamers are not mutated.
4. Optionally, the sequence of one species can be fully constrained, and the system can be left to evolve the other RNA towards the intended regulation (constrained design).
5. Once all sequence and structure constraints are defined, use the inverse folding routine to find the initial, random sequences satisfying them.

3.1.2 Selection

1. Build an objective function that can score a given RNA system taking into account the free energy of each species in the system as well as their secondary structures. Typically, this function is constructed with three weighted terms. The first term is the hybridization energy between the two RNAs. The second term is the activation energy, based on the exposed nucleotides of the seed region, which initiate the reaction [7]. The third is a structural term that accounts for the distance between the targeted structures and the actual ones (*see* Subheadings 3.2 and 3.3 for examples).
2. The hybridization energy is the free energy release due to the RNA-RNA interaction. The energy gap should be as high as possible to ensure interaction in vivo (it is estimated at −17 kcal/mol with Nupack for the system RAJ11 [7]).
3. The activation energy is given by the free energy release due to the interaction between the seed regions (six nucleotides for the system RAJ11 [7]). These regions must be unpaired in the single stranded forms so that the interaction can take place (*see* also **Note 5**).
4. The degree of exposition or blockage of the RBS within the secondary structure of the 5′ UTR (i.e., whether the Shine-Dalgarno box and surrounding nucleotides are paired or not) serves as the variable that accounts for function (such as protein translation).

3.1.3 Mutation Operator

1. The mutation operator takes both sequences as input, the riboregulator and 5′ UTR, and randomly mutates one of them. Nucleotides that are specified to be fixed (e.g., RBS) are not mutated. The mutation operator can make two types of mutation: single-point mutations, or directed mutations.
2. For a single-point mutation, it chooses a nucleotide randomly. If this nucleotide is structurally unconstrained or unpaired at the intramolecular level, this operator simply mutates it to another one. If this nucleotide is constrained and paired intramolecularly, it mutates the base pair in order to keep secondary structure (*see* also **Note 6**).
3. Directed mutations are used to accelerate the search. The operator picks a set of consecutive nucleotides in one sequence (usually between 2 and 4 bases long), and introduces its reverse complement randomly in other sequence. As before, the operator assesses the structural constraints. From 50 to 90 % of mutations can be directed.
4. The mutation operator refuses any mutation that creates a sequence of four or more identical nucleotides in a row.

3.1.4 Post-analysis

In our settings, not all computational jobs converged to good solutions, and screening with a cutoff value of the objective function was required. In addition, false positives may be generated because the process is completely unsupervised. A post-analysis of good solutions (according to ViennaRNA and for the objective function defined) should therefore be performed, using the Nupack software and criteria that were not specified originally.

1. Analyze the base pair probabilities of key regions (typically the RBS) for the intra- and intermolecular folding states using the partition function. During the evolutionary design, the algorithm only considers the minimal free energy structure for simplicity and for reducing computation time, although in fact there is an ensemble of structures [17]. If the percentage of structures in the ensemble satisfying the structural specifications is lower than 90 %, the system should be rejected.
2. Check the behavior of the system at equilibrium with Nupack for defined concentrations of 1 μM for each single species. If the complex species does not represent at least 95 % of total at the equilibrium, the system should be rejected.
3. If it is not included in the design process, the terminator should be added to the final sequence of the sRNA before post-analysis (*see* also **Note** 7).
4. If the evolutionary process selects for structures that are only partially constrained, favor designs that have similar interaction patterns to those of known natural regulatory RNAs.

3.2 Negative Riboregulation

Negative riboregulators have the ability to reduce protein expression of a target gene by either acting on its transcription rate (CRISPRi) [15], the mRNA stability (sRNA-induced degradation), or its translation rate. In this last case, the sRNA interacts with the 5′ UTR to block the RBS (and sometimes the start codon), by direct intermolecular interaction or by inducing a conformational change that results in intramolecular trapping, hiding it from the ribosome (*see* Fig. 3a) [21]. Table 5 shows structural specifications to design negative riboregulators.

1. The single stranded structure of the 5′ UTR must have an unpaired RBS.
2. Within the complex, the RBS must be paired (inter- or intramolecularly).
3. The predicted hybridization energy should be lower than −20 kcal/mol.
4. Favor intramolecular trapping of the RBS to avoid unwanted cross-interaction.

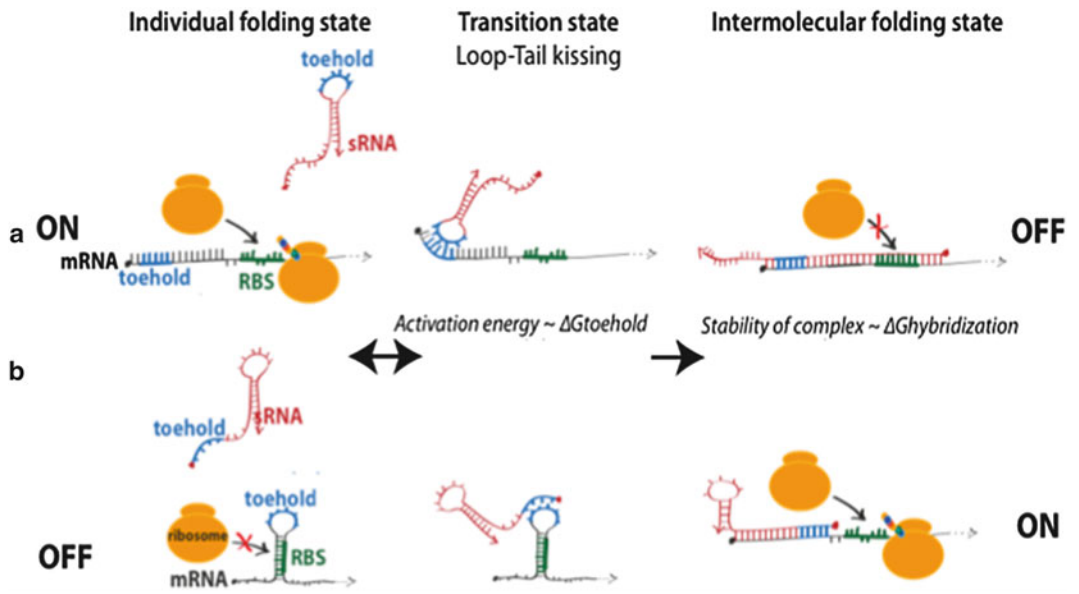


Fig. 3 Mechanisms of: (**a**) Negative riboregulators, where the default state is ON. The binding of sRNA blocks the RBS and represses translation; and of (**b**) Positive riboregulators, where the default state is OFF, due to the RBS being hidden by the 5′UTR secondary of the mRNA. The binding of sRNA unfolds this structure, exposing the RBS and allowing translation. In both cases, the binding of the sRNA goes through a transition phase, when the toeholds—exposed regions of the RNAs—interact, followed by a full hybridization between the complementary regions. The length of the toeholds fixes the activation energy—ΔGtoehold—which determines the speed of the reaction, whereas the stability of the complex—ΔGhybridization—determines the equilibrium of the reaction

3.3 Positive Riboregulation

Positive riboregulators have the ability to increase protein expression of a target gene by acting on its transcription rate (with a fusion between the omega subunit of the RNA polymerase and a Cas9 nuclease) [16], or its translation rate. In this last case, the RBS in the target 5′ UTR (single stranded form) is *cis*-repressed, whilst the interaction with the sRNA causes a conformational change that releases the RBS and allows translation (*see* Fig. 3b). Table 5 shows structural specifications to design positive riboregulators.

1. The single-stranded structure of the 5′ UTR must have a paired RBS.
2. Within the complex, the RBS must be unpaired.
3. The predicted hybridization energy should be lower than -15 kcal/mol.
4. Favor intermolecular interactions between different regions of the sRNA molecule.

3.4 A Generic Methodology for the Design of Logic RNA Devices

In addition to its use for designing positive and negative riboregulators, the described methodology can be applied to optimize further structure-based regulatory specifications. This enables the design of RNA systems based on a larger diversity of mechanisms. Some strategies are described here.

3.4.1 Three-Input Logic Systems

The de novo design of RNA systems based on more than two RNA species can be challenging due to the exponential increase of the size of the search space. However, by either running the program on high-performance computers or by constraining the sequence space, convergence can be achieved. Indeed, in our recent work [21], we presented different examples of interaction mechanisms between two sRNAs and one 5′ UTR.

3.4.2 Pseudo-3D Modeling

One current limitation of automated RNA design can be the number of false positives produced, that is, sequences that behave as desired computationally but that do not work at the experimental level. Certainly, this is due to the lack of a comprehensive intermolecular interaction model (*see* also **Note 8**). It is expected that the incorporation of more structural and functional parameters into the objective function would increase the proportion of good designs. Although full 3D modeling would require too much computing task to be efficiently implemented in an evolutionary algorithm, a library of RNA 3D motifs and noncanonical base pairing, not restricted to the Watson-Crick model, could be used [36, 37].

3.4.3 Integration of Aptamer and Ribozyme Sequences

The possibility of creating modular and signal-processing RNA circuits with various kinds of inputs is of great interest. Aptamers are powerful examples of sensing modules that can be combined with actuator modules. They are hard to design de novo using energy models, as their function is mainly derived from their 3D structure. However, they can be exploited to design aptamer-based riboregulators, as shown in recent experiments with the theophylline aptamer [5, 38]. In addition, it is also possible to incorporate ribozymes or aptazymes as functional elements that can be rearranged in different RNA contexts [12]. This opens up the way for computer-assisted engineering of pathways that have RNA molecules as an input and as an output.

3.4.4 Integration with CRISPR Systems

In bacteria, engineered sRNAs based on the CRISPR-Cas system [14–16] have recently received a lot of attention, and have been harnessed for chromosome engineering [39] as well as regulation at the transcriptional [15, 16] and translational level [14]. Since an sRNA guides the Cas9 nuclease, this opens up the possibility for interaction with designed riboregulators. Indeed, we could design riboregulators to target the Cas9 recognition hairpin or the seed region of the guide RNA, leading to combinatorial RNA-mediated effects.

4 Notes

1. Recombination between homologous regions of promoters is a particularly frequent problem, and care must be taken when selecting promoters to avoid repetitions, symmetrical operators and other sequences prone to recombination [40].

2. Other strains that have the Z1 cassette integrated onto the genome are also available, such as DH5αZ1. However, in our experience, the MGZ1 cells are larger than DH5αZ1 cells, making them more suitable for microfluidics characterization. DH5αZ1 cells are also known to have a relatively low growth rate, which is why we suggest favoring the MGZ1 strain.
3. Bacterial growth rate must be taken into account for the analysis of the characterization data of the riboregulatory systems [41]. Translation rate, and not only protein expression, should be reported. This will avoid eventual artifacts produced by toxic inducers (such as aTc or theophylline), as protein expression is inversely proportional to growth rate, unless it has a degradation tag.
4. Riboregulators tend to work slightly better when both components are present on the same plasmid rather then when they are spread out on two different ones. This is presumably due to intracellular RNA diffusion and unexpected degradation.
5. The seed region is critical for a proper RNA-RNA interaction, as it is responsible for the initiation of the interaction. Mutations in this region can completely disrupt the regulatory behavior. On the other hand, this fact can be exploited to design orthogonal systems easily [6, 8].
6. The enforcement of a given structure for the two single species constrains the sequence space of possible solutions [7, 21]. By leaving those structures unconstrained, we could perform additions and/or deletions (not only replacements) of nucleotides during the optimization, and could potentially speed up the search of a solution.
7. In the case of the mRNA, the terminator and 5′ UTR are separated and isolated by the coding sequence and are unlikely to interact. For the sRNA, however, the terminator often represents a very significant proportion of the molecule (e.g., with the sequences provided in Table 3, the terminator would represent 20–40 % of the final RNA). It is therefore essential to account for this, using preferentially short and stable ones.
8. Computational models do not account for RNA chaperones (e.g., Hfq) [42], nor for cofactors such as Mg^{2+} or Zn^{2+}, which might have an impact on the designs. Moreover, the kinetics of RNA folding, binding, and turnover will have significant impact on the performance of designed RNA circuits [6, 12], and not only the thermodynamic properties.

Acknowledgements

Work supported by the FP7-ICT-043338 (BACTOCOM) grant (to A.J.). We thank Anna Młynarczyk for critical reading of the manuscript. W.R. is supported by a DGA graduate fellowship, T.E. L by an AXA research fund Ph.D. fellowship, and G.R. by an EMBO long-term fellowship co-funded by Marie Curie actions (ALTF-1177-2011).

References

1. Waters LS, Storz G (2009) Regulatory RNAs in bacteria. Cell 136:615–628
2. Isaacs FJ, Dwyer DJ, Collins JJ (2006) RNA synthetic biology. Nat Biotechnol 24:545–554
3. Liang JC, Bloom RJ, Smolke CD (2011) Engineering biological systems with synthetic RNA molecules. Mol Cell 43:915–926
4. Isaacs FJ, Dwyer DJ, Ding C, Pervouchine DD, Cantor CR, Collins JJ (2004) Engineered riboregulators enable post-transcriptional control of gene expression. Nat Biotechnol 22:841–847
5. Bayer TS, Smolke CD (2005) Programmable ligand-controlled riboregulators of eukaryotic gene expression. Nat Biotechnol 23:337–343
6. Lucks JB, Qi L, Mutalik VK, Wang D, Arkin AP (2011) Versatile RNA-sensing transcriptional regulators for engineering genetic networks. Proc Natl Acad Sci U S A 108:8617–8622
7. Rodrigo G, Landrain TE, Jaramillo A (2012) De novo automated design of small RNA circuits for engineering synthetic riboregulation in living cells. Proc Natl Acad Sci U S A 109:15271–15276
8. Mutalik VK, Qi L, Guimaraes JC, Lucks JB, Arkin AP (2012) Rationally designed families of orthogonal RNA regulators of translation. Nat Chem Biol 8:447–454
9. Na D, Yoo SM, Chung H, Park H, Park JH, Lee SY (2013) Metabolic engineering of Escherichia coli using synthetic small regulatory RNAs. Nat Biotechnol 31:170–174
10. Win MN, Smolke CD (2007) A modular and extensible RNA-based gene-regulatory platform for engineering cellular function. Proc Natl Acad Sci U S A 104:14283–14288
11. Wieland M, Hartig JS (2008) An improved aptazyme design and in vivo screening enable riboswitching in bacteria. Angew Chem Int Ed 47:2604–2607
12. Carothers JM, Goler JA, Juminaga D, Keasling JD (2011) Model-driven engineering of RNA devices to quantitatively program gene expression. Science 334:1716–1719
13. Klauser B, Hartig JS (2013) An engineered small RNA-mediated genetic switch based on a ribozyme expression platform. Nucleic Acids Res 41:5542–5552
14. Qi L, Haurwitz RE, Shao W, Doudna JA, Arkin AP (2012) RNA processing enables predictable programming of gene expression. Nat Biotechnol 30:1002–1006
15. Qi LS, Larson MH, Gilbert LA, Doudna JA, Weissman JS, Arkin AP, Lim WA (2013) Repurposing CRISPR as an RNA-guided platform for sequence-specific control of gene expression. Cell 152:1173–1183
16. Bikard D, Jiang W, Samai P, Hochschild A, Zhang F, Marraffini LA (2013) Programmable repression and activation of bacterial gene expression using an engineered CRISPR-Cas system. Nucleic Acids Res. doi:10.1093/nar/gkt520
17. McCaskill JM (1990) The equilibrium partition function and base pair binding probabilities for RNA secondary structure. Biopolymers 29:1109–1119
18. 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
19. Rodrigo G, Carrera J, Landrain TE, Jaramillo A (2012) Perspectives on the automatic design of regulatory systems for synthetic biology. FEBS Lett 586:2037–2042
20. Foster JA (2001) Evolutionary computation. Nat Rev Genet 2:428–436
21. Rodrigo G, Landrain TE, Majer E, Daròs JA, Jaramillo A (2013) Full design automation of multi-state RNA devices to program gene expression using energy-based optimization. PLoS Comput Biol 9:e1003172
22. Hofacker IL, Fontana W, Stadler PF, Bonhoeffer LS, Tacker M, Schuster P (1994) Fast

folding and comparison of RNA secondary structures. Monatsh Chem 125:167–188

23. Dirks RM, Bois JS, Schaeffer JM, Winfree E, Pierce NA (2007) Thermodynamic analysis of interacting nucleic acid strands. SIAM Rev 49:65–88
24. Lutz R, Bujard H (1997) Independent and tight regulation of transcriptional units in Escherichia coli via the LacR/O, the TetR/O and AraC/I1-I2 regulatory elements. Nucleic Acids Res 25:1203–1210
25. Registry of Standard Biological Parts, MIT. http://parts.igem.org
26. Studier FW, Rosenberg AH, Dunn JJ, Dubendorff JW (1990) Use of T7 RNA polymerase to direct expression of cloned genes. Methods Enzymol 185:60–89
27. Larson MH, Greenleaf WJ, Landick R, Block SM (2008) Applied force reveals mechanistic and energetic details of transcription termination. Cell 132:971–982
28. Cambray G, Guimaraes JC, Mutalik VK, Lam C, Mai QA, Thimmaiah T, Carothers JM, Arkin AP, Endy D (2013) Measurement and modeling of intrinsic transcription terminators. Nucleic Acids Res 41:5139–5148
29. Chen YJ, Liu P, Nielsen AA, Brophy JA, Clancy K, Peterson T, Voigt CA (2013) Characterization of 582 natural and synthetic terminators and quantification of their design constraints. Nat Methods 10:659–664
30. D'Aubenton Carafa Y, Brody E, Thermes C (1990) Prediction of rho-independent Escherichia coli transcription terminators. A statistical analysis of their RNA stem-loop structures. J Mol Biol 216:835–858
31. Dunlop MJ, Cox RS 3rd, Levine JH, Murray RM, Elowitz MB (2008) Regulatory activity revealed by dynamic correlations in gene expression noise. Nat Genet 40:1493–1498
32. Hussein R, Lim HN (2011) Disruption of small RNA signaling caused by competition for Hfq. Proc Natl Acad Sci U S A 108:1110–1115
33. Baba T, Ara T, Hasegawa M, Takai Y, Okumura Y, Baba M, Datsenko KA, Tomita M, Wanner BL, Mori H (2006) Construction of Escherichia coli K-12 in-frame, single-gene knockout mutants: the Keio collection. Mol Syst Biol 2:2006.0008
34. Takiff HE, Chen SM, Court DL (1989) Genetic analysis of the rnc operon of Escherichia coli. J Bacteriol 171:2581–2590
35. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680
36. Leontis NB, Stombaugh J, Westhof E (2002) The non-Watson-Crick base pairs and their associated isostericity matrices. Nucleic Acids Res 30:3497–3531
37. Das R, Karanicolas J, Baker D (2010) Atomic accuracy in predicting and designing noncanonical RNA structure. Nat Methods 7:291–294
38. Qi L, Lucks JB, Liu CC, Mutalik VK, Arkin AP (2012) Engineering naturally occurring trans-acting non-coding RNAs to sense molecular signals. Nucleic Acids Res 40:5775–5786
39. Jiang W, Bikard D, Cox D, Zhang F, Marraffini LA (2013) RNA-guided editing of bacterial genomes using CRISPR-Cas systems. Nat Biotechnol 31:233–239
40. Romero D, Martínez-Salazar J, Ortiz E, Rodríguez C, Valencia-Morales E (1999) Repeated sequences in bacterial chromosomes and plasmids: a glimpse from sequenced genomes. Res Microbiol 150:735–743
41. Klumpp S, Zhang Z, Hwa T (2009) Growth rate-dependent global effects on gene expression in bacteria. Cell 139:1366–1375
42. Vogel J, Luisi BF (2011) Hfq and its constellation of RNA. Nat Rev Microbiol 9:578–589

Part II

Circuit Design

Chapter 5

Programming Languages for Circuit Design

Michael Pedersen and Boyan Yordanov

Abstract

This chapter provides an overview of a programming language for Genetic Engineering of Cells (GEC). A GEC program specifies a genetic circuit at a high level of abstraction through constraints on otherwise unspecified DNA parts. The GEC compiler then selects parts which satisfy the constraints from a given parts database. GEC further provides more conventional programming language constructs for abstraction, e.g., through modularity. The GEC language and compiler is available through a Web tool which also provides functionality, e.g., for simulation of designed circuits.

Key words Programming languages, Genetic circuits, Design, Simulation, Constraint solving

1 Introduction

Computer science has a long history of designing formal languages. Some of these are *programming languages*: a program describes computation at a high level of abstraction and can be compiled to binary machine code for execution on a computer. Others are *modeling languages*: a model captures information about an object of interest, such as a communication protocol, and can typically be executed on a computer in order to study emerging behavior.

Over the past decade a number of general purpose modeling languages, such as the Pi calculus [1] and PEPA [2], have been applied in the biological setting, and others such as Kappa [3] and BioPEPA [4] have been developed specifically for applications in biology. Some have also been applied in synthetic biology. For example the Kappa language, as described in Chapter 6, can naturally capture the notions of DNA parts (such as promoters and protein coding regions) and their composition into devices.

But the notion of a programming language also has a natural place in synthetic biology. Here, the *machine* of interest is the cell, and the *machine code* is DNA. Rather than describing DNA explicitly, a programming language for synthetic biology should seek to describe DNA at a suitably high level of abstraction in programs which can then be compiled down to DNA.

This chapter is concerned with one such language called GEC: Genetic Engineering of Cells [5]. A GEC program describes a desired system in terms of constraints, e.g., on relationships between otherwise unspecified parts, and a GEC compiler then selects parts which satisfy the constraints. The parts are selected from a given database which captures relevant logical and quantitative properties of parts. GEC also provides other programming language features such as modularity which facilitate the description of large systems in terms of their components.

Compilation of a GEC program typically results in multiple possible DNA targets, or *solutions*. The GEC compiler can generate a set of chemical reactions for each solution which can be simulated or otherwise analyzed. The solutions that exhibit the desired behavior can then be synthesized and put to work in living cells. Although there is no guarantee that a solution which produces the desired simulation results will function correctly inside a living cell, simulation is an effective way to rapidly detect design errors prior to the costly process of building a physical device in the lab.

In this chapter we first describe the Visual GEC tool [6] for editing, compilation and simulation of GEC programs. We then proceed with an informal overview of the GEC language itself based on small examples, and we provide a syntax definition for the GEC language. We end with a discussion of related and on-going work.

2 The Visual GEC Tool

The Visual GEC tool [6] is implemented as a Silverlight application which runs directly within a browser and is available online.[1] The following description of the tool is based on the autumn 2013 release. The main screen of the tool is divided into two sections: a *design* section towards the left, and an *output* section towards the right.

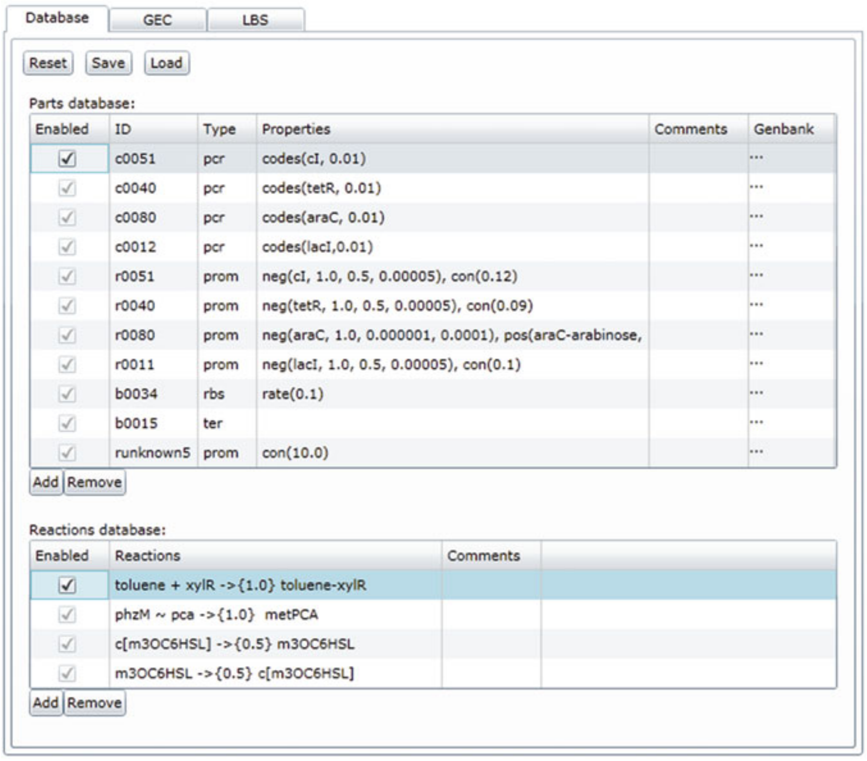
2.1 The Design Section

The design section of Visual GEC contains three tabs: one for editing a parts database, one for editing a GEC program, and one for editing the reactions associated with a given solution to a GEC program.

2.1.1 The Database Tab

The GEC compiler relies on a given database of parts with associated logical and quantitative information. The Database tab contains a sample parts database which can be used as a basis for experimenting with sample models; a screenshot is shown in Fig. 1. The database can be modified or expanded, and any changes

[1] `http://lepton.research.microsoft.com/webgec`.



Parts database:

Enabled	ID	Type	Properties	Comments	Genbank
✓	c0051	pcr	codes(cI, 0.01)		...
✓	c0040	pcr	codes(tetR, 0.01)		...
✓	c0080	pcr	codes(araC, 0.01)		...
✓	c0012	pcr	codes(lacI,0.01)		...
✓	r0051	prom	neg(cI, 1.0, 0.5, 0.00005), con(0.12)		...
✓	r0040	prom	neg(tetR, 1.0, 0.5, 0.00005), con(0.09)		...
✓	r0080	prom	neg(araC, 1.0, 0.000001, 0.0001), pos(araC-arabinose,		...
✓	r0011	prom	neg(lacI, 1.0, 0.5, 0.00005), con(0.1)		...
✓	b0034	rbs	rate(0.1)		...
✓	b0015	ter			...
✓	runknown5	prom	con(10.0)		...

Reactions database:

Enabled	Reactions	Comments	
✓	toluene + xylR ->{1.0} toluene-xylR		
✓	phzM ~ pca ->{1.0} metPCA		
✓	c[m3OC6HSL] ->{0.5} m3OC6HSL		
✓	m3OC6HSL ->{0.5} c[m3OC6HSL]		

Fig. 1 A screenshot of the Design section Database tab

can be saved to and loaded from a file through the "Save" and "Load" buttons, respectively.

Note that the screenshot in fact shows two databases, one for parts (at the top) and one for reactions (at the bottom). In both cases, a new row can be added by pressing the "Add" button, and an existing, selected row can be deleted by pressing the "Delete" button.

The *parts database* has the following columns:

- *Enabled*. This indicates whether a part is enabled. If it is not, it will be ignored by the compiler. This can be useful for experimenting with and debugging a model.
- *ID*. This identifies the part in the database and must be unique. In the sample database, most of the IDs refer to those of the MIT Registry of Standard Biological Parts.[2]
- *Type*. This indicates whether a part is a promoter ("prom"), a ribosome binding site ("rbs"), a protein coding region ("pcr") or a terminator ("ter"). These are the four types currently supported by GEC.

[2] Available at www.partsregistry.org.

- *Properties*. Properties constitute a high-level characterisation of a part. They are used both for resolving constrains in a GEC model, and for constructing the reactions to be used for simulation.
  - *Terminators* currently have no properties.
  - *Ribosome binding sites* have a property of the form "rate (0.1)" which represents a rate of translation of the mRNA, arising from upstream parts, to proteins, arising from downstream parts.
  - *Protein coding regions* have a property of the form "codes (tetR, 0.1)" where the first component is the protein coded by the part (here tetR), and the second component is the degradation rate of this protein.
  - *Promoters* can have several properties. The *constitutive* property of the form "con(0.0001)" represents the constitutive rate of transcription from the promoter. The *negative* property of the form "neg(tetR, 1.0, 0.5, 0.00005)" states that the promoter is negatively regulated by the given transcription factor, here tetR. The remaining real-numbered components represent the rate of promoter-transcription factor binding, the rate of unbinding, and the rate of transcription in the bound state, respectively. There is a corresponding *positive* property of the form "pos(toluene-xylR, 0.001, 0.001, 1.0)"; in this case, the transcription factor is a complex.
- *Comments*. This can be used to specify additional relevant information for a part.
- *Genbank*. This optionally contains Genbank-formatted sequence information and annotations. The Genbank data can be entered by double-clicking the database cell: a small editor then appears. In cases where a part is available from the MIT Registry of Standard Biological Parts and is given the same ID as in the Registry, the Genbank data can be imported automatically by pressing an "Import" button at the bottom of the editing field. Genbank data is not currently used by the GEC tool, but future releases may provide functionality for sending solutions and associated sequence information to automated assembly tools.

The reactions generated by the GEC compiler for a given solution explicitly model transcription at the level of transcription factor binding, based on the quantitative information specified in part properties. If, on the other hand, a promoter is characterized at a different level of abstraction, e.g., based on a Hill function, a custom functional rate can be specified as a promoter property on the form "frate(E)" where E is an algebraic expression over protein names. The "frate" property can be combined in a GEC program with non-quantitative versions of the “pos” and “neg” properties, i.e. of the form “neg(tetR)” and “pos(toluene-xylR)”. These are then used only for constraint satisfaction, and not for generating reactions for simulation.

The *reactions database* is included as a proof of concept, and is intended to represent a comprehensive knowledge base of possible reactions. These can be used to further constrain the selection of parts by the compiler, e.g., by requiring that the proteins expressed by two protein coding regions can dimerise, or be transported in and out of a cell. The reactions database is, however, not necessary and can be ignored for practical purposes: reactions which are needed for simulation can be included directly in a model in the GEC editor as we demonstrate later.

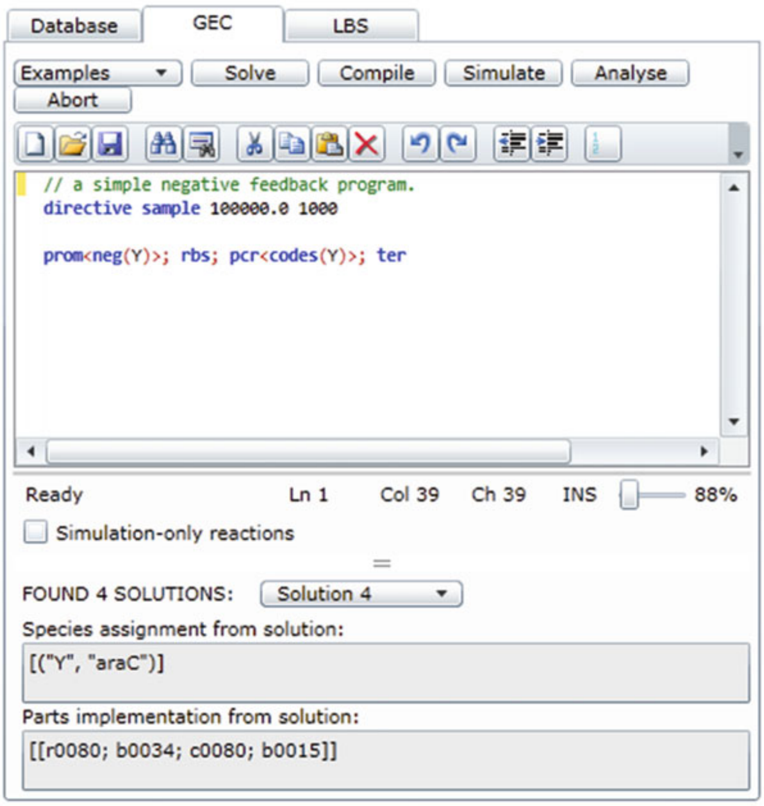
The reactions database has “Enabled” and “Comments” columns similar to the parts database. It also has a “Reactions” column which describes the actual reactions, of the form: “enzyme ~ s1 + . . . + si -> p1 + . . . + pj”. The enzymatic part can optionally be omitted. Transport reactions for representing transport in and out of a cell take the form “s -> c[s]” and “c[s] -> s”, respectively, where “s” is a species and “c” denotes some compartment; the choice of compartment name is insignificant in the context of the database. Note that with the current release of the GEC tool, all numbers in the databases must be written with a decimal point (e.g., “1.0” instead of “1”).

2.1.2 The GEC Tab

The GEC tab provides functionality for editing and compiling GEC programs; a screenshot is shown in Fig. 2. The top section of the tab contains a code editor. The editor has a row of buttons with standard editing functionality, e.g., for saving and opening GEC files. Hovering the mouse over a button shows a description of its functionality. Directly below the editor are numbers showing the line and column numbers of the curser, and a slider for changing the font size as needed. The “Simulation-only reactions” box indicates that any reactions in a model should be used for simulation only, and should not be considered as constraints of the model to be matched against the reactions database.

Visual GEC comes bundled with a number of sample GEC programs which can be selected from the drop-down box labeled “Examples”. The Basic example shown in the screenshot, for instance, is a simple model of a negative feedback loop. The first line of this example is a “//”-prefixed single-line comment. Multi-line comments are also possible: these are opened with “(*” and closed with “*)”.

The second line of the Basic example, starting with “directive”, has no bearing on the model itself, but rather specifies simulation parameters. Out of several available directives, the following two are most commonly used:



```
// a simple negative feedback program.
directive sample 100000.0 1000

prom<neg(Y)>; rbs; pcr<codes(Y)>; ter
```

```
Ready    Ln 1    Col 39    Ch 39    INS    88%
Simulation-only reactions
FOUND 4 SOLUTIONS: Solution 4
Species assignment from solution:
[("Y", "araC")]
Parts implementation from solution:
[[r0080; b0034; c0080; b0015]]
```

Fig. 2 A screenshot of the Design section GEC tab

- *Sample* directives are of the form "directive sample 10000.0 1000" as shown in the screenshot. This specifies that a simulation should run for 10000.0 time units (the first number) and that 1000 data points should be collected for plotting (the second number). Increasing the number of data points in the same period of simulation time produces more fine-grained results but the display may be less responsive. Similarly, if the number of data points stays constant but the simulation time is extended (or shortened), the resulting plot will be less (or more) detailed.
- *Plot* directives are of the form "directive plot A; B; C". This specifies which species to plot during simulation; see for example the bundled Repressilator model for an example. Take care to spell the species names correctly. If for example compartments are included in the model, the correct compartment should also be used in the plot directive, and must be applied per-species in the cases of complexes; for example, write "directive plot c[A]-c[B]" rather than "directive plot c[A-B]".

Line 3 of the Basic example shown in the screenshot constitutes the central part of the model, representing expression of a protein, Y, in a negative feedback loop. We explain the GEC language itself in more detail in the next section. For now, the point to note is that the model contains unspecified parts with certain properties of the kind described previously in the database subsection. A property may contain variables (here Y), and there may be implicit constraints on these variables (here that the same Y is expressed by the protein coding region *and* represses the promoter).

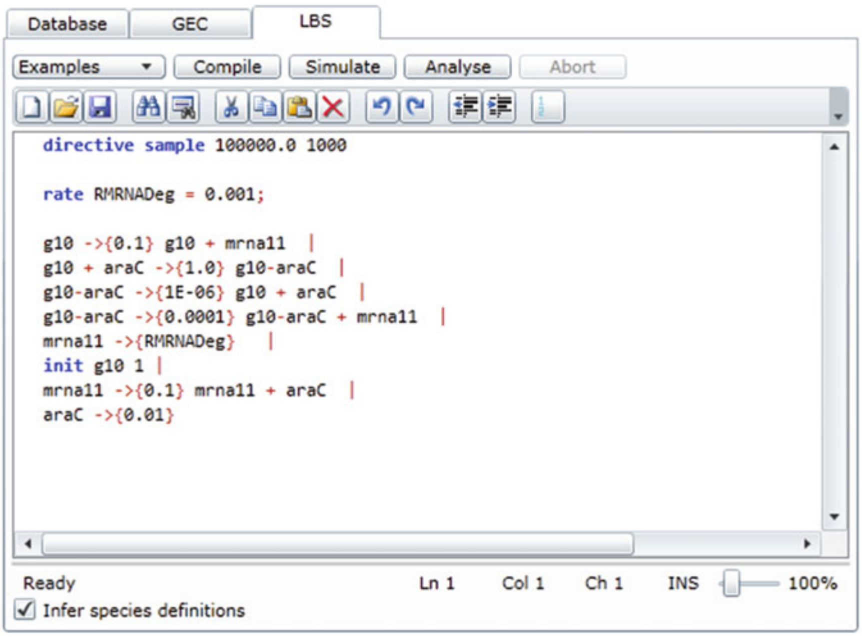
A model can be solved by pressing the "Solve" button. If there are any errors, a message box appears with an indication of the cause. Otherwise, the number of possible solutions is displayed below the editor, with a drop-down box allowing individual solutions to be selected. Note that there may be no solutions if there are no appropriate parts in the database satisfying the constraints of the model.

Selecting a solution populates the "Species assignment" and "Parts implementation" boxes. The species assignment box shows which species have been assigned to variables. For instance, [("Y", "araC")] indicates that araC has been assigned to the variable Y for a given solution in the Basic example. The parts implementation box shows which concrete parts have been chosen for a given solution. For instance, [[r0080; b0034; c0080; b0015]] indicates that r0080 is the chosen promoter; that b0034 is the chosen ribosome binding site; that c0080 is the chosen protein coding region; and that b0015 is the chosen terminator in the case of the Basic example. The parts are selected from the database, and the part names are their database IDs.

2.1.3 The LBS Tab

When a solution is chosen in the GEC tab, a corresponding reaction model is generated for simulation. This model is given in a language called LBS: a *Language for Biochemical Systems* [7], and the model appears automatically under the LBS tab; a screenshot is shown in Fig. 3. In many cases the LBS model can be used as-is, without any modification, but it is made available for editing for cases where further customization or model reduction is needed.

The LBS editor itself is similar to that for GEC, and so are the directives (e.g., "plot" and "sample"). These directives are copied directly from the GEC model, but with species variables substituted for their corresponding values in any given solution. The LBS model generally begins with a rate definition which is used for all generated mRNA degradation reactions ("rate RMRNADeg = 0.001"). The reason is that mRNA in the general case can be polycistronic, so mRNA degradation rates cannot easily be included directly in the parts database. If different mRNA degradation rates are needed for individual reactions, these can be entered manually in the LBS model. The global default rate can also be specified directly in the GEC model, where it takes the slightly different form "rateDef RMRNADeg 0.001".



```
directive sample 100000.0 1000

rate RMRNADeg = 0.001;

g10 ->{0.1} g10 + mrna11  |
g10 + araC ->{1.0} g10-araC  |
g10-araC ->{1E-06} g10 + araC  |
g10-araC ->{0.0001} g10-araC + mrna11  |
mrna11 ->{RMRNADeg}  |
init g10 1 |
mrna11 ->{0.1} mrna11 + araC  |
araC ->{0.01}
```

Ready Ln 1 Col 1 Ch 1 INS 100%
✓ Infer species definitions

Fig. 3 A screenshot of the Design section LBS tab

The LBS model generally contains a number of reactions separated by the "|" symbol. Rates are generally assumed to be mass-action and are enclosed by curly brackets ("{", "}") after a reaction arrow. If however a promoter with a custom functional rate is included, the functional rate is enclosed by square brackets ("[", "]") in the LBS reactions. If the GEC model contains compartments, so will the corresponding LBS model.

Below the LBS editor is a checkbox labeled "Infer species definitions". This instructs the LBS compiler to allow species to be used without first being declared. As the LBS output from a GEC solution does not declare species, the "Infer species definitions" box is checked by default when LBS is used in conjunction with GEC.

Pressing the "Compile" button will result in any changes in the LBS model taking effect in the output section. The "Simulate" button is simply a shortcut to starting simulation in the Output section of the tool discussed below. Note that any unsaved changes made to the LBS model will disappear when selecting another solution under the GEC tab.

2.2 The Output Section

The right hand side of Visual GEC contains a number of tabs related to the reaction output of a GEC solution; a screenshot is shown in Fig. 4. They fall into one of two categories, namely *compilation* and *simulation*, each of which is described in turn below. Note that there are additional tabs containing experimental functionality which we do not discuss here.

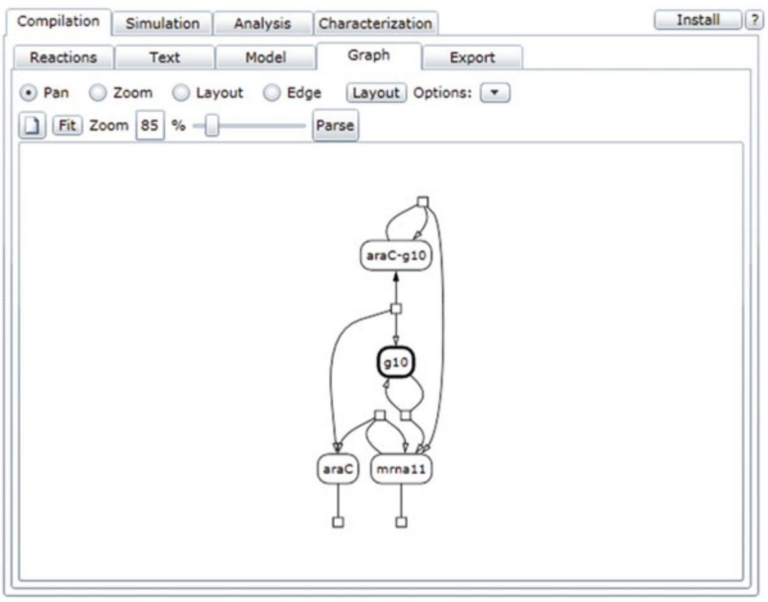


Fig. 4 A screenshot of the Design section Compilation tab

2.2.1 Compilation

The compilation tab contains the following sub-tabs which all represent the result of compilation in a particular format:

- *Reactions* and *text*. These sub-tabs present a flat view of model reactions; the first in a graphical format and the second in a textual format.
- *Model*. This sub-tab shows an editable view of the model, including both reactions and species together with initial conditions.
- *Graph*. This sub-tab shows the reaction network as a graph with two kinds of nodes: rounded rectangles represent species, and square boxes represent reactions.
- *Export*. This sub-tab contains model exports in a number of formats. These include established formats such as SBML, Matlab, and PRISM, as well as formats for specialized spatial modeling tools such as Chaste [8] and CellModeller [9].

2.2.2 Simulation

The simulation tab provides controls for starting, stopping, and stepping through simulations. It also provides controls for parameters such as simulation duration and the number of data points, which by default are populated from directives specified in GEC programs. Several simulation methods, including deterministic and stochastic, are available through a drop down menu. Several views of simulation results are also available, including the following:

- *Chart*. This sub-tab shows simulation data as a plot of species concentrations or populations over time.

- *Table*. This sub-tab shows simulation data in a tabular form, with each row starting with a sample time and subsequent cells containing species concentrations or populations.

3 The GEC Language

Having introduced the Visual GEC tool we now give a brief and informal introduction to the GEC language itself through examples.

3.1 Part Types

On the most basic level, a model can simply be a sequence of part IDs together with their types. The following model is an example of a transcription unit which expresses the protein tetR and which happens to be a negative feedback loop. The corresponding graphical representation is shown in Fig. 5.

The symbol ":" is used to write the type of a part, and the symbol ";" is the sequential composition operator used to put parts together in sequence. Writing this simple model requires the modeler to know that the protein coding region part c0040 codes for the protein tetR and that the promoter part r0040 is negatively regulated by this same protein, two facts which we can confirm by inspecting the parts database from Fig. 1. In this example the GEC compiler has an easy job: it just produces a single list with the given part IDs, while ignoring the part types (*see* Fig. 6).

3.2 Part Variables and Properties

We can increase the abstraction level of the model by using *variables* and *properties* for expressing that any parts will do, as long as the protein coding region codes for the protein tetR and the promoter is negatively regulated by tetR (*see* Fig. 7).

As discussed for entries in the parts database, the angle brackets <> delimit one or more properties. Upper-case names such as X1

a

```
r0040:prom ; b0034:rbs ; c0040:pcr ; b0015:ter
```

b

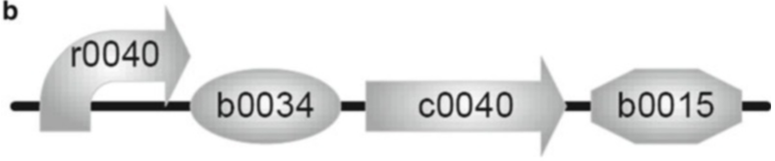
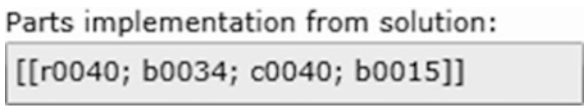


Fig. 5 Model as a sequence of part IDs and types. (**a**) Part list. (**b**) Graphical scheme


Parts implementation from solution:

[[r0040; b0034; c0040; b0015]]

Fig. 6 Part IDs for the tetR mediated negative feedback loop

a

```
X1:prom<neg(tetR)> ; X2:rbs ; X3:pcr<codes(tetR)> ; X4:ter
```

b

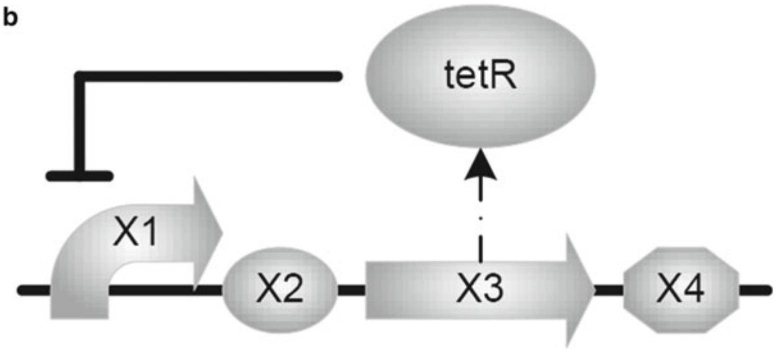


Fig. 7 Use of variables and properties to describe part functions. (**a**) Syntax. (**b**) Schematic

a

```
X1:prom<neg(Y)> ; X2:rbs ; X3:pcr<codes(Y)> ; X4:ter
```

b

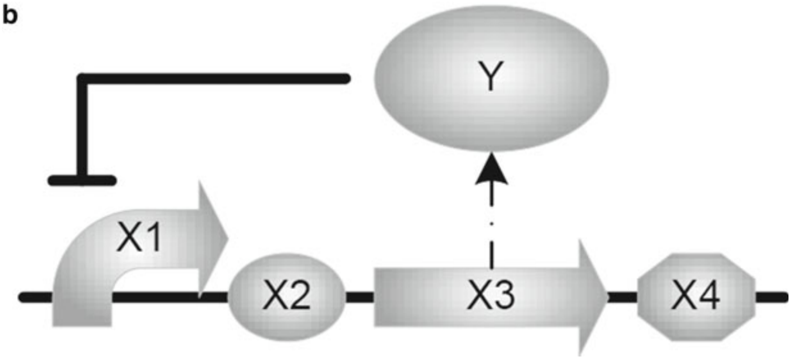


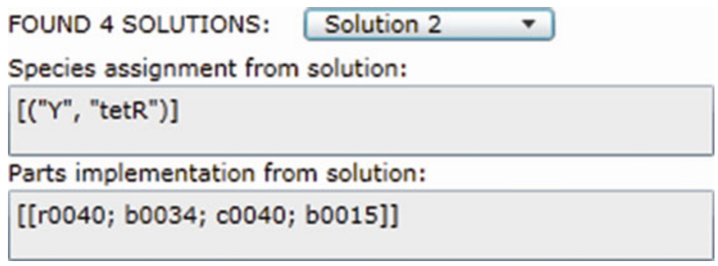
Fig. 8 Abstract representation of a negative feedback loop. (**a**) Syntax. (**b**) Circuit scheme

represent variables (undetermined part names or species). Compiling this model produces exactly the same result as before, only this time the compiler does the work of finding the specific parts required based on the information stored in the parts database.

The compiler may in general produce several results. For example, we can replace the fixed species name tetR with an upper-case variable, thus resulting in a program expressing *any* transcription unit behaving as a negative feedback device as shown in Fig. 8.

Based on the parts database shown in Fig. 1, the compiler now produces four solutions, one of them being the tetR device from above. Selecting one of these solutions populates the "Species assignment" section (*see* Fig. 9).

When variables are used only once, as is the case for X1, X2, X3, and X4 above, their names are of no significance and we will use the



Species assignment from solution:

```
[("Y", "tetR")]
```

Parts implementation from solution:

```
[[r0040; b0034; c0040; b0015]]
```

Fig. 9 Solutions for a negative feedback loop

a

```
prom<neg(Y)> ; rbs ; pcr<codes(Y)> ; ter
```

b

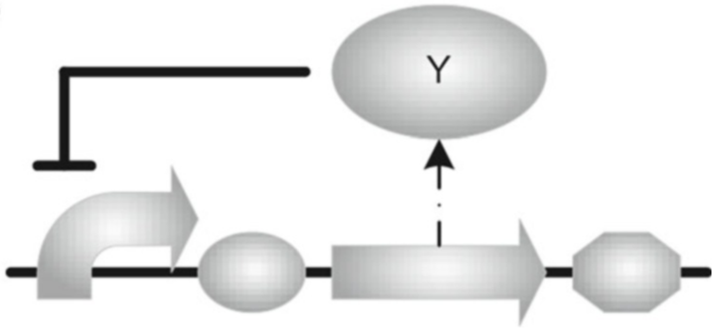


Fig. 10 Negative feedback loop with wild cards. (**a**) Formal and (**b**) graphical representation

```
module gateNeg(i,o) {
  prom<neg(i)>; rbs; pcr<codes(o)>; ter
};
```

Fig. 11 Parameterized representation of a negative feedback loop

wild card, _, instead. When there is no risk of ambiguity, we may omit the wild card altogether and write the above program more concisely as in Fig. 10.

3.3 Parameterized Modules

Parameterised modules can be used to add a further level of abstraction to a model. For example, a module which acts as a negative gate and thus generalizes the above example can be written as in Fig. 11, where “i” denotes *input* and “o” denotes *output*:

Using this module, the *repressilator* circuit [10], in which three genes repress each other, can be written concisely as in Fig. 12.

In general, the “module” keyword is followed by the name of the module, a list of formal parameters, and the body of the module

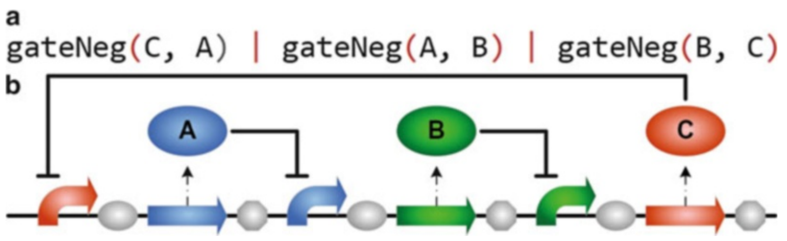
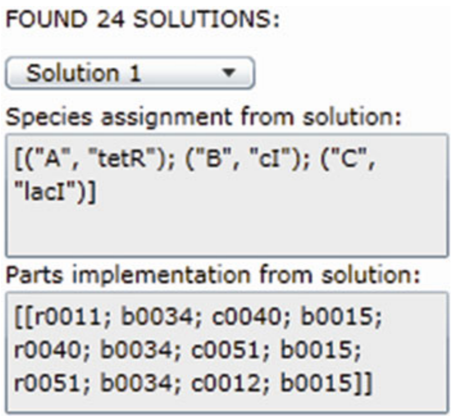


Fig. 12 Concise representation for the repressilator. (**a**) GEC syntax. (**b**) Schematic


FOUND 24 SOLUTIONS:
Solution 1
Species assignment from solution:
Parts implementation from solution:

```
[("A", "tetR"); ("B", "cI"); ("C",
"lacI")]
```

```
[[r0011; b0034; c0040; b0015;
r0040; b0034; c0051; b0015;
r0051; b0034; c0012; b0015]]
```

Fig. 13 A possible repressilator solution in terms of part components

enclosed in brackets; a module can be invoked simply by writing its name followed by a list of actual parameters in parentheses.

The repressilator model yields 24 solutions based on the sample database in Fig. 1. The first solution is shown in Fig. 13.

3.4 Compartments and Reactions

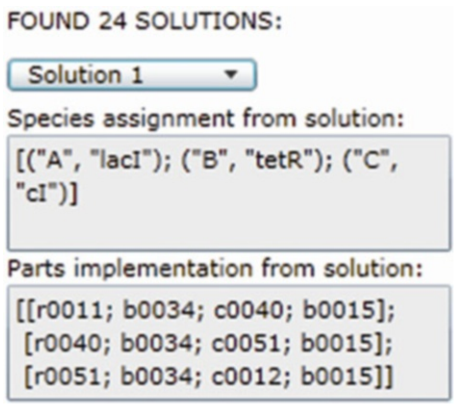
Compartments can be used to represent the location of devices in cases where a multi-cellular system is being designed. For example, the contrived model in Fig. 14 can be thought of as a multi-cellular repressilator, with each gene located in different cells c1, c2, and c3.

Because the genes are in different cells and do not follow each other on the same piece of DNA, the *parallel composition* operator "||" is used rather than the sequential composition operator ";" previously used. As a result, the parts implementations for solutions to this program no longer consists of a single list of parts, but instead of three lists, reflecting the fact that the respective DNA strands are physically disjoint (*see* Fig. 15).

Reactions, which may include compartments, can be used to impose additional constraints on parts. For example, we might impose the additional constraints that the protein A can move in

```
c3[gateNeg(C, A)] | c1[gateNeg(A, B)] | c2[gateNeg(B, C)]
```

Fig. 14 Multicellular repressilator in GEC syntax



```
FOUND 24 SOLUTIONS:
Solution 1
Species assignment from solution:
[("A", "lacI"); ("B", "tetR"); ("C",
"cI")]
Parts implementation from solution:
[[r0011; b0034; c0040; b0015];
 [r0040; b0034; c0051; b0015];
 [r0051; b0034; c0012; b0015]]
```

Fig. 15 Part list for a multicellular repressilator

```
c3[gateNeg(C, A)] | c1[gateNeg(A, B)] | c2[gateNeg(B, C)]
| A -> c1[A] | c1[A] -> A
| A ~ B + C -> B-C
```

Fig. 16 Reactions in GEC syntax

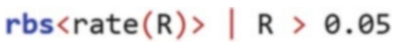
and out of the first cell and, arbitrarily, that proteins B and C can dimerise under the catalysis of A (*see* Fig. 16).

The constraints are composed using the *constraint composition* operator, "|". For this particular example, there are no solutions based on the database in Fig. 1. If, however, the reactions are not intended as constraints to be resolved against the reactions database, they can be used purely for simulation by appending a star (*) to the reaction arrows. If this is done for the above model, compilation yields the same solutions as for the simple repressilator model, but the additional reactions now appear in the resulting LBS model. If all reactions in a model are intended to be used for simulation only, the "Simulation-only" box under the GEC editor can be checked in which the starred version of reaction arrows is not needed.

3.5 Quantitative Constraints

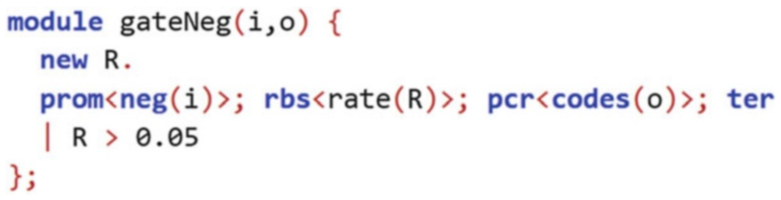
Additional quantitative constraints can be imposed on a model. For example, the instruction in Fig. 17 specifies any ribosome binding site for which the rate of translation is greater than 0.05.

Rate variables in modules should typically be used in conjunction with the *new variable* operator to ensure that different



```
rbs<rate(R)> | R > 0.05
```

Fig. 17 Quantitative constraints in GEC syntax



```
module gateNeg(i,o) {
  new R.
  prom<neg(i)>; rbs<rate(R)>; pcr<codes(o)>; ter
  | R > 0.05
};
```

Fig. 18 A possible constraint on the translation rate in the repressilator

instances of a module can have different numbers assigned to the variable, subject to the quantitative constraints. For example, a constraint on the translation rate in the repressilator module could be written as in Fig. 18.

3.6 The Syntax of GEC

This section defines the concrete syntax of the GEC language in terms of a context-free grammar. The grammar relies on the following lexical conventions, where we write "digit" for a single character in the range 0–9, and "alphanumeric" for any character in the range A–Z or a–z.

- *Name*: the first character of a name must be a lower-case alphanumeric. This is followed by a possibly empty sequence of characters which may be alphanumeric or digits.
- *Variable*: the first character of a variable must be an upper-case alphanumeric. This is followed by a possibly empty sequence of characters which may be alphanumeric or digits.
- *Comma-separated lists* (possibly empty) are written with an underline.
- *Integer*: a non-empty sequence of digits.
- *Float*: a floating point number following standard notational conventions, e.g., 3.141 or 5e-3.

Single-line comments are prefixed with "//". Multi-line comments are opened with `(*` and closed with `*)`, and may be nested.

The GEC language is then defined in terms of the grammar in Table 1, where terminal symbols are written in `teletype` font and non-terminals are in **bold**. Note that functional rates associated with promoter properties are only used in the database, so do not feature in the language grammar.

Table 1
The syntax of the GEC language defined through a context free grammar

Non-terminal	Definition	Description
DGEC ::=	**Directive** GEC	A GEC program with directives
Directive ::=	`directive sample` **Float**	End time for simulation
|	`directive sample` **Float Integer**	With data points
|	`directive plot` **Plots**	Species to plot
Plots ::=	**PlotSpec**	One species to plot
|	**PlotSpec** `;` **Plots**	Or more
PlotSpec ::=	**LName**	Atomic located name
|	**LName** - **PlotSpec**	Complex located name
LName ::=	**Name**	Simple species name
|	**Name** [**Name**]	In location/compartment
GEC ::=	**APart**	Abstract part
|	**GEC; GEC**	Sequential composition
|	**GEC || GEC**	Parallel composition
|	**Name** `[`**GEC**`]`	Compartment
|	**GEC** `|` **Constraint**	Constraint composition
|	`new` **`Variable .`** **GEC**	New variable
|	`module` **Name**`(`**FrmPar**`){`**GEC**`};` **GEC**	Module definition
|	**Name**(**ActPar**)	Module invocation
APart ::=	**AName** `:` **PartType**<**Property**>	Abstract part
|	**PartType**<**Property**>	Without name
|	**AName** `:` **PartType**	Without properties
|	**PartType**	Without name & properties
PartType ::=	`prom`	Promoter type
|	`rbs`	Ribosome binding site type
|	`pcr`	Protein coding region type
|	`ter`	Terminator type
Property ::=	`pos(`**ASpec**`)`	Positive regulation
|	`pos(`**ASpec, AFloat, AFloat, AFloat**`)`	With rates
|	`neg(`**ASpec**`)`	Negative regulation
|	`neg(`**ASpec, AFloat, AFloat, AFloat**`)`	With rates
|	`con(`**AFloat**`)`	Constitutive expression

(continued)

Table 1 (continued)

Non-terminal	Definition	Description
|	rate(**AFloat**)	Translation rate
AFloat ::=	**Float**	Concrete abstract float
|	**Variable**	Variable abstract float
|	_	Wild card
AName ::=	**Name**	Concrete abstract name
|	**Variable**	Variable abstract name
|	_	Wild card
ASpec ::=	**AName**	Atomic abstract species
|	**AName - ASpec**	Complex abstract species
Constraints ::=	**Reaction**	Reaction constraint
|	**Transport**	Transport constraint
|	**NumCons**	Numerical constraint
|	**Constraint** | **Constraints**	Constraint composition
Reaction ::=	**ASpec~Sum Arrow Sum**	Enzymatic reaction
|	**Sum Arrow Sum**	Standard reaction
Sum ::=		Empty sum
|	**ASpec+Sum**	Composite sum
Arrow ::=	->	Simple arrow
|	*->	Simulation-only
|	->{**AFloat**}	Arrow with rate
|	*->{**AFloat**}	Simulation-only
Transport ::=	**ASpec Arrow Name** [**ASpec**]	Transport into compartment
|	**Name** [**ASpec**] **Arrow ASpec**	Transport out of compartment
NumCons ::=	**ASpec>ASpec**	Greater than constraint
FrmPar ::=	**Name**	Formal parameter
ActPar ::=	**AName**	Actual name parameter
|	**Float**	Actual float parameter

4 Discussion

4.1 Related Work

A number of dedicated tools for computer assisted genetic engineering have emerged in recent years. We refer to, e.g., [11] for an overview. In the following we outline four tools which, like GEC, are based on programming languages. They all allow for simulation of designs, but otherwise embody different design choices and priorities.

GenoCad [12] is a Web-based tool for the design of genetic circuits from standard parts in a structured, top-down manner. The tool ensures that the designed circuits are *syntactically valid*, e.g., that a promoter is followed immediately by a ribosome binding site and not terminator. Syntactic validity is formally defined by means of a context-free grammar where terminal symbols are given by parts, analogous to how English words form the basic components in the English grammar. An extension to the framework [13] furthermore allows for reactions to be deduced from designs and for subsequent simulation. Although the approach inherently allows for abstraction away from specific parts, there is no tool support for exploring combinatorial instantiations of such abstractions.

Antimony [14] is a language for describing biochemical systems in a modular fashion. It includes a range of language features for capturing general properties of biochemical processes, for example reactions and events, and models can be translated to SBML. It also includes dedicated support for synthetic biology through constructs for representing and composing DNA strands, but it does not support higher-level abstractions away from specific parts.

Eugene [15] is a dedicated language for design of genetic circuits. Eugene allows for abstraction away from specific parts in a similar fashion to GEC, but uses more verbose language constructs to specify combinatorial compositions of parts and constraints on these. It additionally provides a range of features similar to those of general purpose programming languages, for example explicit control flow with *while* loops, explicit access to data structures such as arrays, and statements for printing strings to a console. In this sense Eugene can be considered more of an imperative language in contrast to GEC which is more declarative. Eugine also provides support for automated assembly through integration with the Clotho [16] framework.

Finally, *TASBE* (*Toolchain to Accelerate Synthetic Biology Engineering*) [17] in an end-to-end tool for designing circuits starting from programs in Proto, a language for capturing general spatial processes which has been adapted to the synthetic biology domain. Like GEC and Eugene, programs can abstract away from the level of specific parts and the tool automatically selects relevant parts. Like Eugene, TASBE also goes a step further in its support for automated assembly.

4.2 Limitations

The computational design framework implemented within Visual GEC is inspired by the notion of programming languages, as well as by abstraction and standardization conventions adopted recently in the field of synthetic biology. It relies on several assumptions such as the modularity of genetic parts and the availability of quantitative information describing their properties, in order to provide a convenient, high-level programming language. In the following, we discuss these assumptions before highlighting some on-going developments and extensions of GEC.

Modularity. The abstraction of DNA sequences as parts is an appealing and pervasive concept in synthetic biology. However, this notion often relies on the implicit assumption that the various genetic components used for the construction of circuits are modular and can be combined in different ways while their properties are preserved. In the context of GEC, such modularity is assumed for all parts collected in the parts database and the devices constructed from them. Therefore, the quantitative properties associated with parts remain unchanged, regardless of the context in which they are used. For example, the constitutive expression rate from a given promoter is the same, even when different proteins are expressed from it. This modularity assumption enables the construction of mathematical models for arbitrary GEC solutions directly from the quantitative properties provided within the parts database.

In practice, modularity of components and devices is not always observed in experimental implementations. Several approaches have been developed to overcome various context effects, thus facilitating the modular design and construction of systems. In [18], the concept of *retroactivity* was introduced to capture the effects of interconnections on the input-output characteristics of components. Drawing inspiration from strategies used in electrical circuit design to surmount similar modularity issues, a feedback insulation mechanism was proposed in [18] to enable the design of modular genetic circuits. More recently, design strategies on the level of DNA sequences were proposed in [19] to ensure the reliable and predictable performance of various combinations of genetic elements. More specifically, it was demonstrated that using appropriately engineered expression cassette architectures to control transcription and translation initiation results in more reliable expression from various component combinations. Besides the application of engineering strategies to ensure the modularity of parts and devices, a better understanding of the mechanisms that break modularity allows the construction of more detailed computational models that include the effects of context and can guide design efforts even when context-sensitive components are used.

Part characterisation. To construct computational models that allow various gene circuit design solutions to be studied further through simulation and analysis, the GEC compiler relies on the quantitative information associated with the parts in the supplied database. Such quantitative measurements are not always available in practice and current part and device repositories do not require such information to be contributed.

Even so, powerful techniques have been developed for the in vivo characterisation of parts and devices, where experimental measurements are used to obtain quantitative information, enabling the computational procedures integrated within GEC. Most prominently, the use of fluorescent reporter proteins has emerged as a convenient characterisation strategy in the field, allowing the high-throughput measurements of population-averaged signals (e.g., using fluorescence microplate readers), signal distributions across a population (e.g., using flow cytometry) and even single-cell measurements (e.g., using fluorescent microscopy).

The use of fluorescent reporter proteins provides a convenient characterisation tool but the obtained measurements are often affected by experimental conditions. To overcome such effects and attenuate experimental variability, the use of relative measurement was proposed in [20] for the characterisation of promoter activity. Recently, this strategy was extended and generalized in [21], leading to an alternative characterisation framework. There, spectrally distinct reporter proteins were used to enable the simultaneous measurement of multiple fluorescent signals, allowing the direct ratiometric characterisation of parts and devices.

While the characterisation data for genetic components is currently sparse, as high-throughput experimental procedures are developed and more *part datasheets* are produced, the quantitative information required within the GEC database for the construction of models is becoming more readily available.

Part selection and parameter tuning. As part of the design methodology implemented within GEC, a number of genetic circuit solutions are proposed for a given program based on the availability of parts. Additional model simulations and analysis procedures are then applied to study these designs and identify the ones that implement some required properties. With certain computational analysis approaches, alternative model parameters corresponding to physical quantities such as expression or degradation rates are proposed in order to "tune" a given (flawed) circuit for some specific behavior. Such a strategy was used, for example, in [22] to design and tune genetic circuits that give rise to certain desired population dynamics in bacterial colonies.

The tuning of circuits provides an orthogonal and complementary methodology to the selection of a design from the set proposed automatically by GEC. Intuitively, parameter tuning allows the exploration of a design space that goes beyond the set of all possible circuits realizable given the available parts. It also offers insights about design robustness and the specific parameter ranges where some behavior is observed. However, to implement a tuned circuit, novel parts with modified characteristics are often required, which has promoted the construction of libraries of related parts that share most properties but differ with respect to the one that is being tuned. Once such libraries are constructed, integrating them within the GEC database allows additional designs to be achieved through part selection rather than parameter tuning, with the guarantee that the parts required for their implementation are already available.

4.3 On-Going Developments

To address some of the limitations outlined in the previous section, extend the functionality, and improve the usability of the GEC programming language and the Visual GEC tool, a number of developments are currently on-going. In the following, we briefly describe several directions of current and future work.

Characterisation methods. As experimental measurements characterizing the properties of genetic parts and devices become more readily available, computational methods capable of integrating such quantitative information into reliable mathematical models become necessary. To provide such functionality, we are currently implementing novel data processing capabilities within GEC [23] to support characterisation studies using the ratiometric experimental procedures developed in [21]. In addition to exploiting the strategies integrated within this characterisation method to attenuate experimental variability, our computational procedure relies on Bayesian parameter inference, which allows parameter uncertainty to be captured and propagated within the models.

Due to experimental limitations (e.g., measuring only a small number of fluorescent signals which represent the concentrations of certain reporter proteins), characterisation procedures such as the one from [21] often do not provide enough information to infer all the parameters needed for detailed mechanistic models of, e.g., binding between promoters and transcription factors. This is the motivation for allowing promoter properties with functional rates in the parts database.

Database development. In the current GEC implementation, only minimal information is included within the parts database, as required in order to represent parts and devices while retaining information about their corresponding DNA sequences and to construct mathematical models for arbitrary GEC solutions. As more experimental data characterizing different genetic components (possibly in different contexts such as hosts and environmental conditions) becomes available, the database can expand to include and make use of such information. Additional developments targeting the integration of the database with other resources and repositories, and facilitating the sharing of components and characterisation data within the community, are also possible.

Abstractions. The GEC programming language allows the structure (e.g., the order of parts on a plasmid) and logical properties (e.g., positive and negative regulations) of designs to be specified at an abstract level, while additional quantitative constraints on the relevant parameters are also supported. However, for certain applications, it is often convenient to describe designs even more abstractly by focusing on required system properties and behavior, rather than on specifying and constraining the possible implementations. In other fields, similar *formal synthesis* methods have been developed with the goal of automatically proposing correct-by-construction designs from a set of functional requirements, additional constraints and available components. While these problems are inherently difficult, significant progress has been achieved recently in the context of engineering computer software and hardware systems.

In recent work, the problem of composing (previously constructed) devices with characterized dynamical properties into larger scale systems was considered [24]. There, a library of devices was used as a starting point, mirroring the library of parts within GEC. In addition, a number of assembly constraints (e.g., preventing chemical cross-talk) as well as specifications of required dynamic behavior were considered as part of the design process. In that context, an automatically identified "solution" corresponds to a particular arrangement of devices within a system, similarly to the composition of parts within a device, as defined by a GEC solution. However, an abstract dynamical model capturing only the logical interactions between components was studied in [24] and additional research on the methods level is needed to target more detailed models, such as the ones constructed by GEC. Further work is also needed on the language level to reflect such abstractions in GEC programs.

Constraint solving. As the number of components available within parts databases increases, identifying valid solutions for a given GEC program becomes challenging. Indeed, it was recently demonstrated that such "parts matching" problems are computationally hard [25]. In addition, once higher-level properties specifying required system behavior and capturing various assembly constraints are considered, powerful new methods are needed to tackle these challenging problems.

Recently, the significant progress in the development of constraint solving methods has offered potential strategies for addressing these issues. In particular, Satisfiability Modulo Theories (SMT) and the available solvers targeting SMT problems provide one promising direction for the formalization and scalable implementation of methods extending the GEC compiler. SMT can be seen as a form of constraint satisfaction that extends the classical Boolean satisfiability (SAT) problem (the search for variable assignments leading to the satisfaction of a given Boolean formula), allowing richer theories including real numbers, integers, bit-vectors and various data structures. Among a number of available SMT solvers, Z3 [26] has emerged as a robust and powerful competitor in the field, which motivated its use as part of the methods developed in [24]. An SMT-based framework provides a convenient formalization of the parts matching problem and has the capacity to capture higher level properties and constraints. The integration of these methods within GEC is currently on-going.

References

1. 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. doi: 10.1142/9789814447362_0045
2. Hillston J (1996) A compositional approach to performance modelling. Cambridge University Press, New York
3. Danos V, Feret J, Fontana W et al (2007) Rule-based modelling of cellular signalling. In: CONCUR, volume 4703 of LNCS. Springer, pp 17–41. doi: 10.1007/978-3-540-74407-8_3
4. Ciocchetta F, Hillston J (2009) Bio-PEPA: a framework for the modelling and analysis of biological systems. Theor Comput Sci 410 (33–34):3065–3084. doi:10.1016/j.tcs.2009.02.037
5. Pedersen M, Phillips A (2009) Towards programming languages for genetic engineering of living cells. J R Soc Interface. ISSN 1742-5662. doi: 10.1098/rsif.2008.0516.focus
6. Pedersen M, Lakin M, Polo M et al (2013) GEC tool. http://research.microsoft.com/gec
7. Pedersen M, Plotkin G (2010) A language for biochemical systems: design and formal specification. In: Trans Comput Syst Biol, volume 5945. Springer, pp 77–145. doi:10.1007/978-3-642-11712-1_3
8. Mirams GR, Arthurs CJ, Bernabeu MO et al (2013) Chaste: an open source C++ library for computational physiology and biology. PLoS Comput Biol 9(3):e1002970. doi:10.1371/journal. pcbi.1002970
9. Rudge TJ, Steiner PJ, Phillips A et al (2012) Computational modeling of synthetic microbial biofilms. ACS Syn Biol. doi:10.1021/sb300031n
10. Elowitz MB, Leibler S (2000) A synthetic oscillatory network of transcriptional regulators. Nature 403(6767):335–338
11. Clancy K, Voigt C (2010) Programming cells: towards an automated 'Genetic Compiler'. Curr Opin Biotechnol 21:572–581. doi:10.1016/j.copbio.2010.07.005
12. Cai Y, Hartnett B, Gustafsson C et al (2007) A syntactic model to design and verify synthetic genetic constructs derived from standard biological parts. Bioinformatics 23 (20):2760–2767. doi:10.1093/bioinformatics/btm446
13. Cai Y, Lux MW, Adam L et al (2009) Modeling structure function relationships in synthetic DNA sequences using attribute grammars. PLoS Comput Biol 5(10):e1000529. doi:10.1371/journal.pcbi.1000529
14. Smith LP, Bergmann FT, Chandran D et al (2009) Antimony: a modular model definition language. Bioinformatics 25(18):2452–2454. doi:10.1093/bioinformatics/btp401
15. Bilitchenko L, Liu A, Cheung S et al (2011) Eugene—a domain specific language for specifying and constraining synthetic biological parts, devices, and systems. PLoS One 6(4). doi: 10.1371/journal.pone.0018882
16. Densmore D, Van Devender A, Johnson M et al (2009) A platform-based design environment for synthetic biological systems. In: The 5th Richard Tapia celebration of diversity in computing conference: intellect, initiatives, insight, and innovations, TAPIA'09. ACM, pp 24–29. doi: 10.1145/1565799.1565806
17. Beal J, Weiss R, Densmore D et al (2012) An end-to-end workflow for engineering of biological networks from high-level specifications. ACS Syn Biol. doi:10.1021/sb300030d
18. Vecchio DD, Ninfa AJ, Sontag ED (2008) Modular cell biology: retroactivity and insulation. Mol Syst Biol 4. doi:10.1038/msb4100204
19. Mutalik VK, Guimaraes JC, Cambray G et al (2013) Precise and reliable gene expression

via standard transcription and translation initiation elements. Nat Methods 10(4):354–360, http://dx.doi.org/10.1038/nmeth.2404

20. Kelly JR, Rubin AJ, Davis JH et al (2009) Measuring the activity of BioBrick promoters using an in vivo reference standard. J Biol Eng 3:4. doi:10.1186/1754-1611-3-4
21. Brown JR (2011) A design framework for self-organised Turing patterns in microbial populations. Phd dissertation, University of Cambridge
22. Dalchau N, Smith MJ, Martin S et al (2012) Towards the rational design of synthetic cells with prescribed population dynamics. J R Soc Interface 9(76):2883–2898. doi:10.1098/rsif.2012.0280
23. Yordanov B, Dalchau N, Grant P et al (2013) Automated ratiometric characterization using GEC. In: 2013 international workshop on biodesign automation (IWBDA'13), July 2013
24. Yordanov B, Wintersteigern CM, Hamadi Y et al (2013) SMT-based analysis of biological computation. In: NASA formal methods symposium 2013. Springer. doi:10.1007/978-3-642-38088-4_6
25. Bhatia Y, Bhatia S, Adler A et al (2012) Automated selection of synthetic biology parts for genetic regulatory networks. ACS Syn Biol 1 (8):332–344. doi:10.1021/sb300032y
26. de Moura LM, Bjørner N (2008) Z3: an efficient SMT solver. In: TACAS, volume 4963 of LNCS. Springer, pp 337–340. doi:10.1007/978-3-540-78800-3_24

Chapter 6

Kappa Rule-Based Modeling in Synthetic Biology

John Wilson-Kanamori, Vincent Danos, Ty Thomson, and Ricardo Honorato-Zimmer

Abstract

Rule-based modeling, an alternative to traditional reaction-based modeling, allows us to intuitively specify biological interactions while abstracting from the underlying combinatorial complexity. One such rule-based modeling formalism is Kappa, which we introduce to readers in this chapter. We discuss the application of Kappa to three modeling scenarios in synthetic biology: a unidirectional switch based on nitrosylase induction in *Saccharomyces cerevisiae*, the repressilator in *Escherichia coli* formed from BioBrick parts, and a light-mediated extension to said repressilator developed by the University of Edinburgh team during iGEM 2010. The second and third scenarios in particular form a case-based introduction to the Kappa BioBrick Framework, allowing us to systematically address the modeling of devices and circuits based on BioBrick parts in Kappa. Through the use of these examples, we highlight the ease with which Kappa can model biological interactions both at the genetic and the protein–protein interaction level, resulting in detailed stochastic models accounting naturally for transcriptional and translational resource usage. We also hope to impart the intuitively modular nature of the modeling processes involved, supported by the introduction of visual representations of Kappa models. Concluding, we explore future endeavors aimed at making modeling of synthetic biology more user-friendly and accessible, taking advantage of the strengths of rule-based modeling in Kappa.

Key words Rule-based modeling, Kappa, Synthetic biology, Switch, Repressilator, BioBrick parts, Modularity

1 Introduction

Rule-based modeling languages, including Kappa [1] and the BioNetGen Language [2], have been the focus of attention in developing biological models that are concise, comprehensible, and easily extensible [3]. Such languages represent biological entities such as proteins, functional units of DNA, and mRNAs as agents. Agents are named sets of sites that can be used to hold state or bind and interact with other agents. Interactions are represented by rules in the form of precondition and effect, governed by an associated rate constant that determines how frequently the interaction occurs. Rules differ from reactions in that participating agents need not be fully specified in the precondition—for example, phosphorylation at a particular site may occur independently from whether its neighboring site is bound or not—which means that a single rule may encompass any number of individual reactions. In this way, rule-based approaches alleviate the combinatorial explosion that results from molecular entities existing under multiple different conditions (for example, states of phosphorylation). The combination of different independent rule sets implicitly generates different overall systems, thus allowing modular development of subsystems and their composition into a conjoined whole.

The goal of this chapter is to provide an introduction to modeling in synthetic biology using the aforementioned Kappa rule-based modeling language. We do so by revisiting two of the best-known devices in synthetic biology, the toggle switch and the repressilator, in a series of case studies designed to gradually introduce the reader to the techniques involved. These techniques include the visual representations of protein interactions, and the Kappa BioBrick Framework for modeling BioBrick parts. We deliberately introduce these concepts and models in a nonspecialist manner, in the hopes of reaching out to as wide an audience as possible.

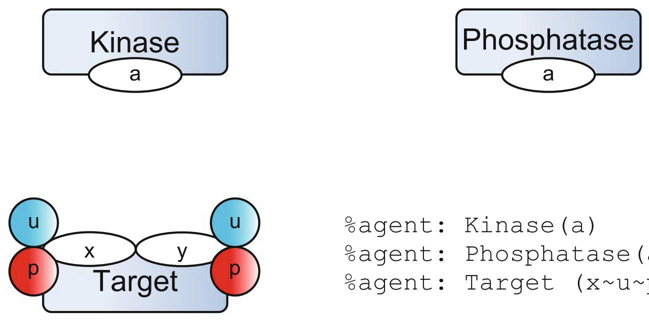
Although we adopt Kappa as our language of choice, the differences between it and the BioNetGen Language are minimal both in syntax and in implementation. However, one benefit of Kappa over BioNetGen is that Kappa tools utilize formal methods, such as causal summaries and reachability analysis, to aid information discovery in and debugging of large models.

Let us begin by introducing the key elements of Kappa in further detail.

An *agent* in Kappa is simply an entity with a name and a number of labeled *sites*. A site may hold internal *state*, typically used to denote a post-translational modification such as the agent's phosphorylation status. State may also be used to denote the agent's location in a model that takes into account different reaction compartments such as the cytosol and nucleus.

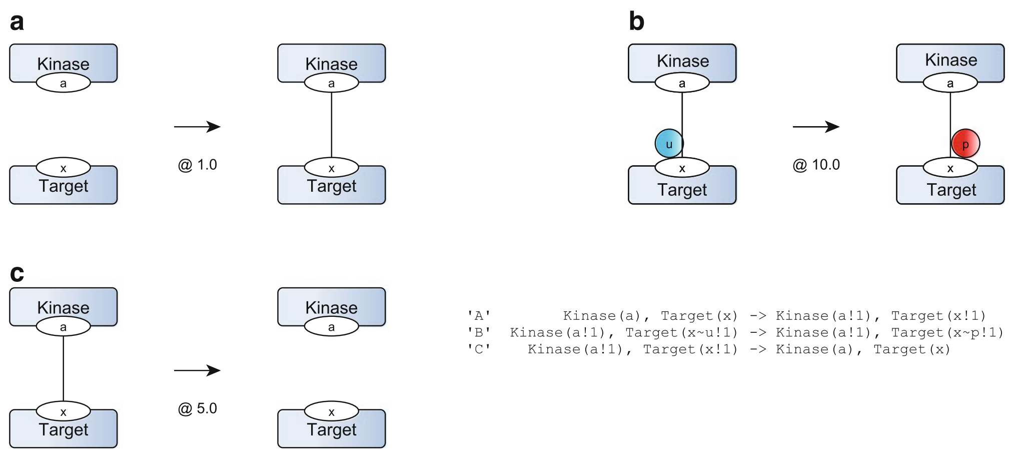
Agent interactions are described via *rules*. Rules often correspond to elementary mechanistic interactions such as the binding or unbinding of two agents, modification of the state of a site, or the creation or deletion of an agent. Complex rules can also describe combinations thereof.

As an introductory example, let us consider a basic kinase-phosphatase model. One has three agents (Fig. 1): a Kinase, a Target with two sites "x" and "y" that may be independently phosphorylated, and a Phosphatase. A phosphorylation event may be simply described by three elementary actions (binding, modification, and unbinding) and their corresponding rules (Fig. 2). In the corresponding textual notation (also shown in Figs. 1 and 2), internal states are represented as "~u" (unphosphorylated) and



```
%agent: Kinase(a)
%agent: Phosphatase(a)
%agent: Target (x~u~p, y~u~p)
```

Fig. 1 The agents involved in the basic kinase-phosphatase model: a Kinase with a single binding site "a," a Phosphatase also with a single binding site "a," and a Target with two binding sites "x" and "y" that may also switch state between phosphorylated ("~p") and unphosphorylated ("~u"). We show the textual code corresponding to each agent visualization at *bottom right*



```
'A'         Kinase(a), Target(x) -> Kinase(a!1), Target(x!1)    @ 1.0
'B'  Kinase(a!1), Target(x~u!1) -> Kinase(a!1), Target(x~p!1) @ 10.0
'C'    Kinase(a!1), Target(x!1) -> Kinase(a), Target(x)        @ 5.0
```

Fig. 2 The three rules describing a phosphorylation in the kinase-phosphatase model. (A) The Kinase binds its Target at one of the two sites (in this case "x"); (B) the Kinase phosphorylates the site at which it is bound; (C) the Kinase dissociates (unbinds) from its target. Note the possibility that rule C might fire before rule B, thus not every binding event between a Kinase and a Target will result in a phosphorylation

"~p" (phosphorylated), and bindings as "!" with shared indices across agents indicating the two endpoints of a link. The left hand side of the rule describes the precondition that must be satisfied for the rule to apply. The right hand side describes its effect.

Note that not all sites of an agent need be present in a rule's precondition: the rules shown in Fig. 2 never mention the Target's "y" site. Likewise, even if a site is mentioned its internal state may be left unspecified: binding a Kinase to its Target does not take into account the fact that the Target may already be phosphorylated at that site. These are both examples of the "don't care, don't write" philosophy, where only the minimal information describing the triggering of a rule need be represented in the left hand side.

This is what allows Kappa to alleviate the combinatorial explosion inherent in biological models.

The action of the Phosphatase, which works to counteract the Kinase, may be described using similar rules. The only difference between the two would then lie in the modification rule (Fig. 2b), where the state of the Target changes from "~p" to "~u" rather than vice versa. Of course, this description would be a choice made by the modeler, and it would be entirely possible to model the Phosphatase (and the Kinase) differently but with a similar mechanistic effect. For example, we could design a "smart" Phosphatase that only binds when a Target is already phosphorylated, or we could ensure that it never unbinds without dephosphorylating by combining the two effects into a single rule.

Every rule is associated with a *rate constant*, which controls the probability of the rule "firing" during the simulation. At any given time in the simulation, the rule may apply to the *mixture* (the pool of interacting agents, including their binding configuration and internal state) multiple times according to how often the precondition holds. Each possible application has the same rate, hence the number of applications is multiplied by the rate of the rule to determine the rule's propensity (flux) at that point in time. The likelihood that the rule will fire next is proportional to this flux. Any obtained trajectory, of course, is but one realization of a stochastic process that may differ when repeated. Manipulating the rate constants will influence these trajectories.

Perturbations form an integrated extension to the base Kappa language allowing the modeler to specify the effect of external influences on the model. For example, one may add and delete agents from the mixture conditioned on simulation time or other conditions pertaining to the state of the system.

Given a set of agents and rules as defined above, as well as appropriate *initial conditions*, one can then track user-defined *observables* in a stochastic *simulation*. The kinase-phosphatase example of a whole Kappa model that may be used for such a simulation is shown in Fig. 3. The stochastic trajectories are obtained by using a rule-based variant of Gillespie's method [4] to simulate a continuous time Markov chain.

As well as this agent-centric view of system dynamics, automated tools exist to track causality and precedence in a model in a contrasting rule-centric view, for example to answer the question of what succession of events results in a fully phosphorylated Target. The idea behind such *stories* is to retain only the events in the causal lineage that contribute a net progression towards an event of interest (i.e., by removing causal loops).

A useful overall view of the rule set is the *contact graph* (Fig. 4), akin to a protein–protein interaction map. The contact graph is a graph whose nodes are the agents with their interfacing set of sites, and whose edges represent possible bindings between sites.

```
### Agents:
%agent: Kinase(a)
%agent: Phosphatase(a)
%agent: Target(x~u~p,y~u~p)


### Rules:

# Kinase action
'Kinase x binding'                Kinase(a), Target(x) -> Kinase(a!1), Target(x!1)   @ 1.0
'Kinase x phosphorylation'  Kinase(a!1), Target(x~u!1) -> Kinase(a!1), Target(x~p!1) @ 10.0
'Kinase x unbinding'          Kinase(a!1), Target(x!1) -> Kinase(a), Target(x)       @ 5.0
'Kinase y binding'                Kinase(a), Target(y) -> Kinase(a!1), Target(y!1)   @ 1.0
'Kinase y phosphorylation'  Kinase(a!1), Target(y~u!1) -> Kinase(a!1), Target(y~p!1) @ 10.0
'Kinase y unbinding'          Kinase(a!1), Target(y!1) -> Kinase(a), Target(y)       @ 5.0

# Phosphatase action
'Phtase x binding'                Phosphatase(a), Target(x) -> Phosphatase(a!1), Target(x!1)   @ 1.0
'Phtase x phosphorylation'  Phosphatase(a!1), Target(x~p!1) -> Phosphatase(a!1), Target(x~u!1) @ 10.0
'Phtase x unbinding'          Phosphatase(a!1), Target(x!1) -> Phosphatase(a), Target(x)       @ 5.0
'Phtase y binding'                Phosphatase(a), Target(y) -> Phosphatase(a!1), Target(y!1)   @ 1.0
'Phtase y phosphorylation'  Phosphatase(a!1), Target(y~p!1) -> Phosphatase(a!1), Target(y~u!1) @ 10.0
'Phtase y unbinding'          Phosphatase(a!1), Target(y!1) -> Phosphatase(a), Target(y)       @ 5.0


### Initial Conditions:
%init: 100 (Target(x~u,y~u))
%init: 10  (Kinase(a))
%init: 10  (Phosphatase(a))
```

Fig. 3 The textual representation of the kinase-phosphatase model in Kappa. Agent definitions (Fig. 1) are followed by the rules of the model (Fig. 2) and the initial population. Simulation observables are not included, but are defined in a similar manner

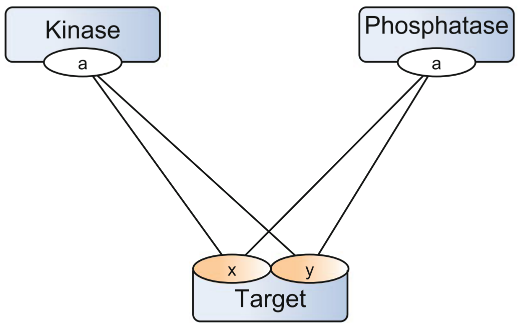


Fig. 4 A contact graph for a simple kinase-phosphatase model. The three agents in the model are represented as nodes along with their sites, and each site is connected to the other sites it can bind to. If a site may be modified by this interaction, it is so indicated by a color. Although simple enough, the associated rule set (three agents, 12 rules) already generates 38 non-isomorphic complexes (of which 36 contain the Target in various stages of binding and phosphorylation) (Color figure online)

Possible changes in state are indicated by a color code upon the site in question.

The basic contact graph does not provide causal information, only possibilities, but we may extend it (Fig. 5) by mapping to it the model rules categorized depending on their effect: binding and unbinding, creation, destruction, modification, and transport

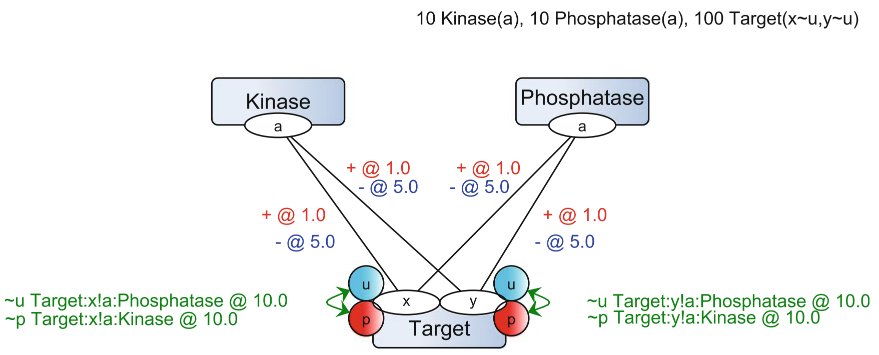

10 Kinase(a), 10 Phosphatase(a), 100 Target(x~u,y~u)
Kinase
a
Phosphatase
a
+ @ 1.0
- @ 5.0
+ @ 1.0
- @ 5.0
+ @ 1.0
- @ 5.0
+ @ 1.0
- @ 5.0
u
p
x
y
u
p
Target
~u Target:x!a:Phosphatase @ 10.0
~p Target:x!a:Kinase @ 10.0
~u Target:y!a:Phosphatase @ 10.0
~p Target:y!a:Kinase @ 10.0

Fig. 5 An extension of the basic contact graph mapping the 12 rules (and their rates) of the model to the agents and their sites. States are shown explicitly, and the preconditions for their modification annotate the *double-headed arrow* between them. Agent interactions are annotated with the rate of binding (*positive, in red*) and unbinding (*negative, in blue*); as these are very basic binding and unbinding rules, no further annotations are needed. The entirety of this visualization may be used to reconstruct the textual model as shown in Fig. 3

between different model compartments. Each edge in this visual formalism corresponds to one or more rules. In turn, each annotation on an edge describes the precondition (if any) and rate to one of these rules, such that said rule may be reconstructed wholly from the annotation. For example, two agents may bind with each other in two different rules dependent on their phosphorylation state; in the visualization, this would be represented as a single edge between the two agents but annotated twice, once for each rule and attached rate. Reversing this process thus allows us to recover the entirety of the model from the visualization, minus simulation-specific observables. This extended visualization is less useful when the model involves complex rules combining two or more of the above effects, however, and is currently defined manually (with plans for partial automation) in .graphml format (using the yED tool) from the underlying Kappa file.

To summarize the above: a Kappa model is a collection of agents (sets of sites that may hold state) and their rate-controlled interactions in the form of rules, which may be visualized concisely in the form of a contact graph. Given a set of initial conditions and observables, one may then execute this model to generate a stochastic simulation that tracks agent and mixture evolution over time, modeling external influences via perturbations and observing the causal properties of this evolution via stories.

A variety of tools exist for modeling in Kappa. Foremost amongst these is KaSim [5], which may be used to simulate and analyze a Kappa model defined directly in textual form (as displayed in Fig. 3). Under development for many years, the features embodied in KaSim are well-developed and user-tested. A recent alternative is LMS-Kappa [6], an embedded domain-specific language written in Scala: users may write Kappa models using a Java-compatible object-oriented functional programming language with Java-like syntax to generate the rules. LMS-Kappa also eases the process of supplementing the core language with modular extensions (e.g., geometric constraints on the assembly of complexes) and makes the creation of Kappa models more accessible and more flexible from a programmer's perspective. Another recent effort is LBS-Kappa [7], which attempts to capture the modularity inherent in common biological models to achieve concise representations, as an implementation of the Language for Biochemical Systems introduced in Chapter 5 of this book.

With our introduction to Kappa now complete, we move on to some practical applications of the modeling language to synthetic biology. We highlight the manner in which models in Kappa may capture biological interactions at both the genetic and the protein–protein interaction levels through a series of case studies:

- A unidirectional switch based on nitrosylase induction in *Saccharomyces cerevisiae*, building on the kinase-phosphatase model introduced in this section.
- The repressilator in *Escherichia coli* formed from BioBrick parts, introducing the Kappa BioBrick Framework.
- A light-mediated extension to said repressilator developed by the University of Edinburgh team during iGEM 2010, to demonstrate how to build on the Kappa BioBrick Framework to create a model combining both protein and genetic interactions.

All models, including the introductory model described in this section, are available at RuleBase (http://rulebase.org), an online repository for rule-based models; the models from this chapter are located at http://rulebase.org/users/884. Prospective modelers will also find other examples of biologically interesting models there.

2 Modeling a Synthetic Switch

We begin exploring the role of the Kappa modeling language in synthetic biology by applying it to a real-life example of a synthetic switch. The design of this example flows naturally from the kinase-phosphatase model described in Subheading 1, and follows the idea of controlling devices based on Goldbeter-Koshland kinetics [8] known to possess ultrasensitive toggle-like properties in specific parameter ranges. Concretely, we use inducible promoters to control the two "arms" of the Goldbeter-Koshland loop, represented previously by Kinase and Phosphatase and in the example to follow as TRXh5 and GSNO.

The constitutively expressed NPR1 transcription cofactor is an integral coordinator [9] of the multi-layered cellular defence system in *Arabidopsis* (Fig. 6). In resting cells, it is subject to *S*-nitrosylation at a known amino acid (Cys156) by the GSNO protein acting

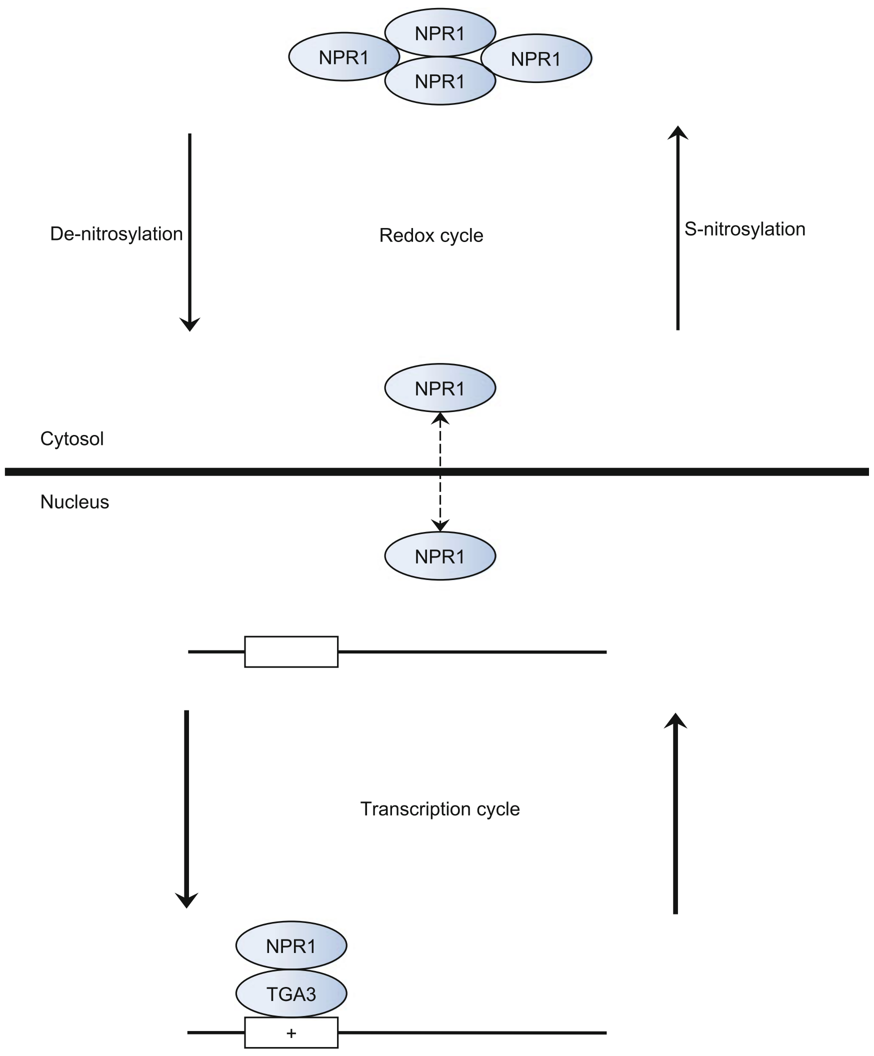


Fig. 6 Under resting conditions, the *S*-nitrosylation of NPR1 by GSNO promotes formation of the NPR1 oligomer. Pathogen recognition causes ambient changes to cellular redox potential, which promotes monomer release. Nucleic monomeric NPR1 interacts and forms a complex with transcription factors at target promoters to activate them

to promote the assembly of the NPR1 oligomer. Modification to the intracellular redox environment by invasive pathogens, and the subsequent action of the enzymes TRXh5 and NTRA, releases the NPR1 monomer instead. Monomeric NPR1 interacts with the TGA subclass of transcription factors essential for activating immune defence genes, thus stimulating the cell's genetic response to pathogen invasion.

Investigating this plant immune cascade allows us to identify system function influencing the outcome of defence signaling. In particular, much of the processes surrounding NPR1 function

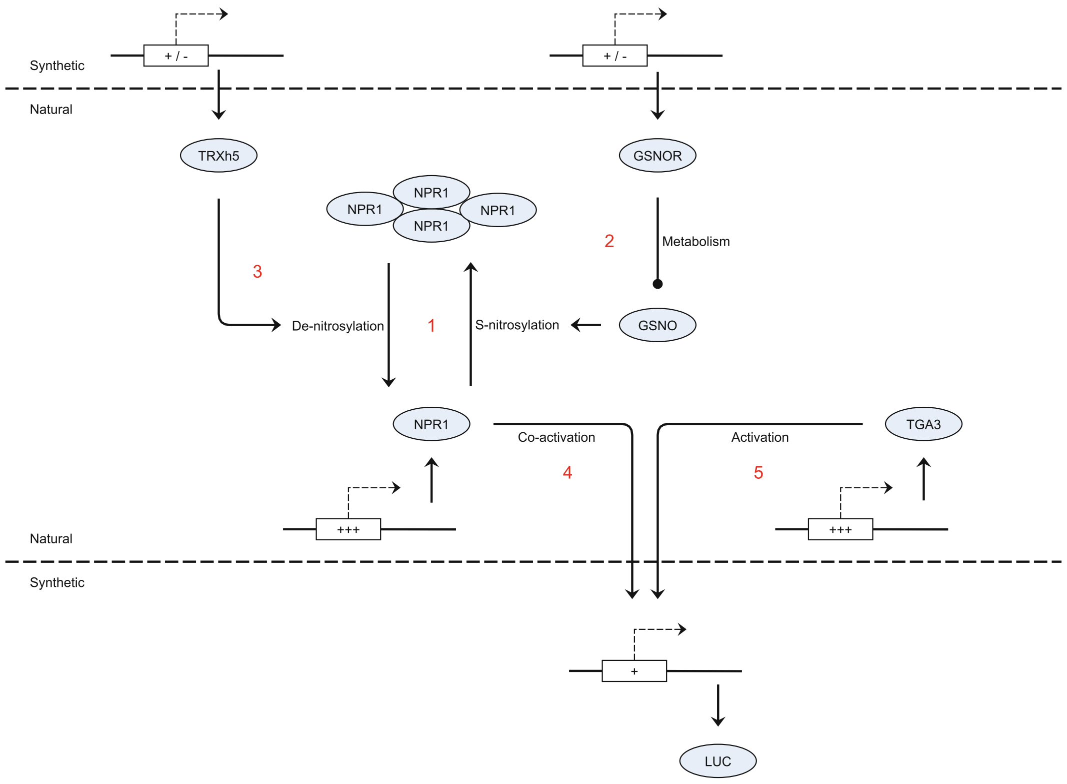


Fig. 7 A schematic of the unidirectional synthetic protein switch. The labels on the promoters indicate inducible by an outside chemical compound (+/−), constitutively expressed (+++), and activated as pathway output (+) respectively. Although the system itself is natural (albeit transcribed from *Arabidopsis* to yeast), the inducible promoters and the reporter are synthetic. Note the structural similarities to the basic kinase-phosphatase model—a target (NPR1) that switching between conformations (oligomeric to monomeric) under the influence of TRXh5 (forwards) and GSNO (backwards). (1) Under resting conditions, GSNO promotes oligomer formation of constitutively expressed NPR1. (2) Selectively activating GSNOR increases the rate at which GSNO is metabolized, thereby reducing *S*-nitrosylation of NPR1 and indirectly limiting oligomer formation. (3) Activation of TRXh5 (together with NTRA) de-nitrosylates the NPR1 oligomer and promotes monomer release. (4) Monomeric NPR1 interacts with constitutively expressed TGA3, readying it for activity as a transcription factor. (5) This then activates the expression of a LUC reporter gene

are not fully understood. Work by John Moore at the University of Edinburgh [10] approaches this problem by creating a synthetic protein circuit based on a theoretical interpretation of plant immunity, using a *S. cerevisiae* yeast chassis engineered to be amenable to redox manipulation.

By designing a synthetic circuit (Fig. 7) with selectively inducible inputs in the form of TRXh5 and the GSNO reductase GSNOR, we specifically promote the reduction of NPR1 to its monomeric form. This creates a unidirectional "on" switch with the standard luciferase reporter gene as the circuit output. It is not a true toggle switch as described by Gardner et al. [11], since there is no inducible mechanism for directly turning the circuit "off" again,

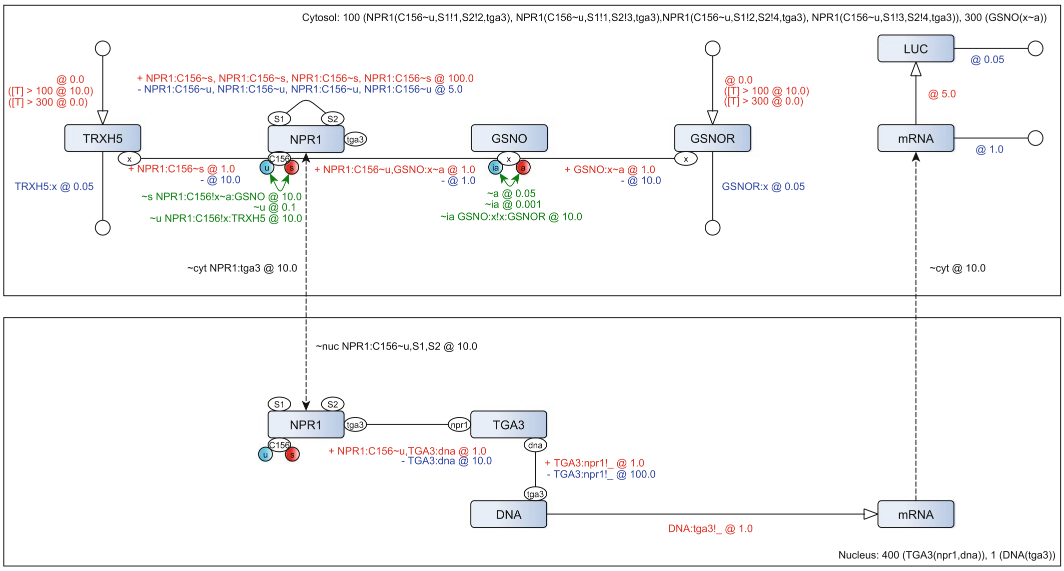

Cytosol: 100 (NPR1(C156~u,S1!1,S2!2,tga3), NPR1(C156~u,S1!1,S2!3,tga3),NPR1(C156~u,S1!2,S2!4,tga3), NPR1(C156~u,S1!3,S2!4,tga3)), 300 (GSNO(x~a))
@ 0.0
([T] > 100 @ 10.0)
([T] > 300 @ 0.0)
+ NPR1:C156~s, NPR1:C156~s, NPR1:C156~s, NPR1:C156~s @ 100.0
- NPR1:C156~u, NPR1:C156~u, NPR1:C156~u, NPR1:C156~u @ 5.0
S1
S2
TRXH5
NPR1
tga3
GSNO
GSNOR
LUC
mRNA
x
C156
u
s
+ NPR1:C156~s @ 1.0
- @ 10.0
TRXH5:x @ 0.05
~s NPR1:C156!x~a:GSNO @ 10.0
~u @ 0.1
~u NPR1:C156!x:TRXH5 @ 10.0
+ NPR1:C156~u,GSNO:x~a @ 1.0
- @ 1.0
ia
a
~a @ 0.05
~ia @ 0.001
~ia GSNO:x!x:GSNOR @ 10.0
+ GSNO:x~a @ 1.0
- @ 10.0
@ 0.0
([T] > 100 @ 10.0)
([T] > 300 @ 0.0)
GSNOR:x @ 0.05
@ 0.05
@ 5.0
@ 1.0
~cyt NPR1:tga3 @ 10.0
~cyt @ 10.0
~nuc NPR1:C156~u,S1,S2 @ 10.0
TGA3
npr1
dna
+ NPR1:C156~u,TGA3:dna @ 1.0
- TGA3:dna @ 10.0
+ TGA3:npr1!_ @ 1.0
- TGA3:npr1!_ @ 100.0
DNA
DNA:tga3!_ @ 1.0
Nucleus: 400 (TGA3(npr1,dna)), 1 (DNA(tga3))

Fig. 8 Visualizing the Kappa model of the synthetic immune pathway. GSNO, NPR1, and TGA3 are assumed to be constitutively expressed at rest, with GSNO promoting formation of the NPR1 oligomer. In addition to the visualization concepts described previously, we introduce the notion of compartments and transport between them (indicated by the *dashed arrows*), creation and degradation of agents (indicated by *source and sink nodes*) and the perturbation language (indicated by the *bracketed annotations on the edges leading from the source nodes*). At simulation time T $=$ 100, GSNOR and TRXh5 are inserted into the system via perturbation, de-nitrosylating NPR1 and promoting its monomeric form instead. As described in the text, monomeric NPR1 then translocates to the nucleus, combining with TGA3 to activate the expression of the reporter gene. The transcribed mRNA is transported back to the cytosol, producing LUC output

but rather a gradual return to the preinduction state as the inducible inputs filter out of the system.

We then develop a Kappa model to provide a mathematical representation of the system and to facilitate further investigation via model perturbations to supplement experimental analysis. This model is visualized in Fig. 8. The model visualizations described in Subheading 1 are extended further here to incorporate compartment information in a rudimentary fashion (again, modeled as a location site holding state either "~cytosol" or "~nucleus"). The compartments are supplemented with the initial populations of the agents present.

We begin our modeling case study by assuming the constitutive expression of NPR1, GSNO, and TGA3 proteins. Initially in the cytosolic compartment, the C156 site of the NPR1 agent favors its *S*-nitrosylated "~s" state due to the action of GSNO. GSNO and NPR1 can bind if NPR1 is de-nitrosylated (its "C156" site is in its "~u" state) and GSNO is active (its "x" site is in its "~a" state); once bound, another rule allows NPR1 to *S*-nitrosylate by switching state from "~u" to "~s" on its "C156" site. This ensures that NPR1 retains its oligomeric form, represented by four NPR1 agents forming a tetrameric ring. In turn, this prevents NPR1 from leaving the cytosol (since our transport rules state that nucleic transport of NPR1 can only occur when it is de-nitrosylated and its "S1" and "S2" sites are unbound, i.e., it is in monomeric form) and activating the downstream reporter in the nuclear compartment. These rules are fully visualized in Fig. 8.

The implementation illustrates an important adage: the information contained in the model is directly related to what is known of the system and the desired level of abstraction. For example, we make the choice in this example to model the NPR1 oligomer as a tetrameric ring of four separate agents rather than simply a state (oligomer vs. monomer) of the individual agent. This choice has further consequences on system dynamics, as we shall see shortly.

TRXh5 and GSNOR agents do not exist in the initial population of the cytosol, but are introduced from simulation time 100 via a model perturbation substituting for the expression of the relevant synthetic construct (the perturbation is represented in parentheses on the creation edge in Fig. 8). This influx is turned off again at simulation time 300, by which time enough TRXh5 and GSNOR are introduced to toggle the system switch. TRXh5 directly de-nitrosylates NPR1, switching the state of the "C156" site to "~u," whilst GSNOR inactivates GSNO and prevents it from acting on NPR1 to reverse this change; the second effect takes place over a slightly longer timescale than the first. Cumulatively, the two proteins stimulate the release of the monomeric form of NPR1, which we choose to model as the oligomeric NPR1 breaking up into its constituent parts once all four NPR1 agents are de-nitrosylated. The system thus behaves explosively as a large number of NPR1 monomers are released in a short time; in contrast, if we had chosen to model the conformation of NPR1 as a state as described previously, then we would have seen a more gradual change. Again, although mathematical models are often helpful in describing hitherto unknown behavior, the system can only behave according to the assumptions we make when we model it.

Monomeric NPR1 is free to transport to the nucleic compartment, where it interacts with the TGA3 transcription factor to activate the reporter gene. The mRNA thus produced is transported back to the cytosol, where in the final stage of the model cascade it is translated into the LUC reporter protein—the output of the synthetic circuit.

Switching off TRXh5 and GSNOR induction causes the system to return slowly to its preinduction state as the two protein populations are filtered out by constitutive degradation. NPR1 returns to its oligomeric form, spurred by the *S*-nitrosylating action of reactivated GSNO, and the reporter gene is turned off. Hence we have a unidirectional switch structure controllable by the induction described previously.

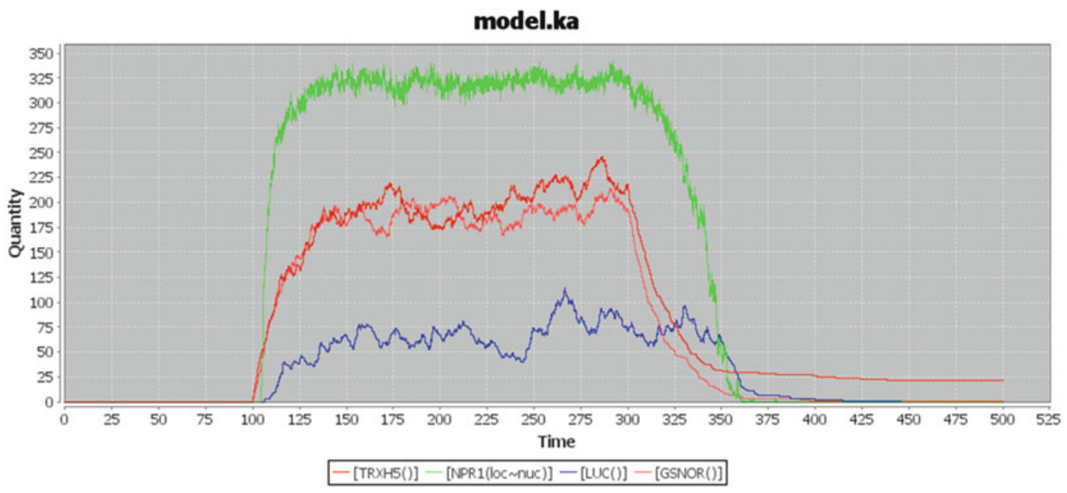


Fig. 9 Stochastic simulation of the NPR1 oligomer-to-monomer switch modeled in Kappa. Plot demonstrating that nuclear-localized monomeric NPR1 and subsequent reporter gene expression is dependent on induction of GSNOR and TRXh5. GSNOR and TRXh5 are switched on at simulation time T = 100 and off at T = 300. Units (time and quantity) are arbitrary

In vivo studies of the synthetic circuit have determined that it takes 2 h for GSNOR/TRXh5 to appear once induced, 4–5 h for the threshold level of de-nitrosylated NPR1 to be reached, 90 min for the cytosolic oligomer to convert to a nucleic monomer, and a further 2–3 h for LUC protein creation. Unfortunately these abstract observations are insufficient in themselves to fully specify the kinetic rates that determine model dynamics. Hence, as a first step we simply guess at these rate constants in the actual model, although simultaneously we maintain the initial protein populations as faithful to observed biology as possible.

As a result, at this stage simulations of the model (Fig. 9) display highly qualitative dynamics that are useful for tracking causality but not for reproducing the in vivo temporal behaviors described above. Given more careful characterisation and analysis, our model is poised to make pertinent observations regarding the sensitivity of the NPR1 oligomer–monomer switch to the concentrations of TRXh5 and GSNO in the system, as well as the responsiveness (how long it takes to complete stimulation response) and strength (the proportion of oligomeric NPR1 converted to monomeric form) of the switch. For the time being, it serves as an example of how we may model simple synthetic circuits, and in particular the protein interactions that may compose such circuits, in Kappa.

Now that we have demonstrated how to proceed from toy examples (kinase-phosphatase) to full-fledged biological models, our next question is how to tackle structured synthetic biology based on BioBrick parts.

3 The Kappa BioBrick Framework

3.1 Introducing the Kappa BioBrick Framework

Modular methodologies for modeling structured synthetic biological systems, based on the BioBrick standard [12] and formalized by systems of ordinary differential equations, were first explored by Marchisio and Stelling in 2008 [13]. Other tools such as TinkerCell [14] facilitate modelers looking to incorporate important principles such as stochasticity and analysis paradigms including parameter scanning into the modeling of BioBrick parts. We introduce in this section such a methodology, designed to assist the modeling in Kappa of circuits based on BioBrick parts.

The Kappa BioBrick Framework, originally laid down by the third author in 2009, differs from the above by incorporating the advantages of rule-based modeling with the description of dynamic BioBrick parts. Given a specification of a device constructed from BioBrick parts, it provides rules describing how these gene constructs are processed by the transcription and translation machinery of the cell (RNA polymerases and ribosomes). Such a framework allows the modular formalization of individual functional units within the system and their composition into more complicated devices and systems. It also meshes well with incremental strategies for modeling synthetic biological systems. Finally, the structure of the Kappa BioBrick Framework corresponds with efforts to standardize the characterisation of BioBrick parts, utilizing measures such as Polymerases Per Second (PoPS, the rate of transcription defined as the number of times that an RNA polymerase molecule passes a specific point on DNA per second) or Ribosomal Initiations Per Second (RiPS, the level of translation as the number of ribosome molecules that pass a point on mRNA each second).

Our approach is similar to later work by Marchisio et al. [15] regarding the use of the BioNetGen language in a framework for the design of complex eukaryotic gene circuits (*see* Chapter 7). Unlike this, the Kappa BioBrick Framework does not specifically cater for synthetic biology based on mRNA regulation using small RNAs, or eukaryotic issues such as compartmentalisation or RNA interference, although these may of course be added by the modeler. An additional difference is that Marchisio et al. utilize the Model Definition Language to provide an interface between modules; the Kappa BioBrick Framework relies instead on the inherent compositionality of the rules in Kappa.

We provide further details of a preliminary implementation of the Kappa BioBrick Framework in the conclusion of this chapter. For now, we begin describing the Kappa BioBrick Framework by considering the four functional categories of BioBrick parts: promoters, coding sequences, ribosome binding sites (RBS), and terminators.

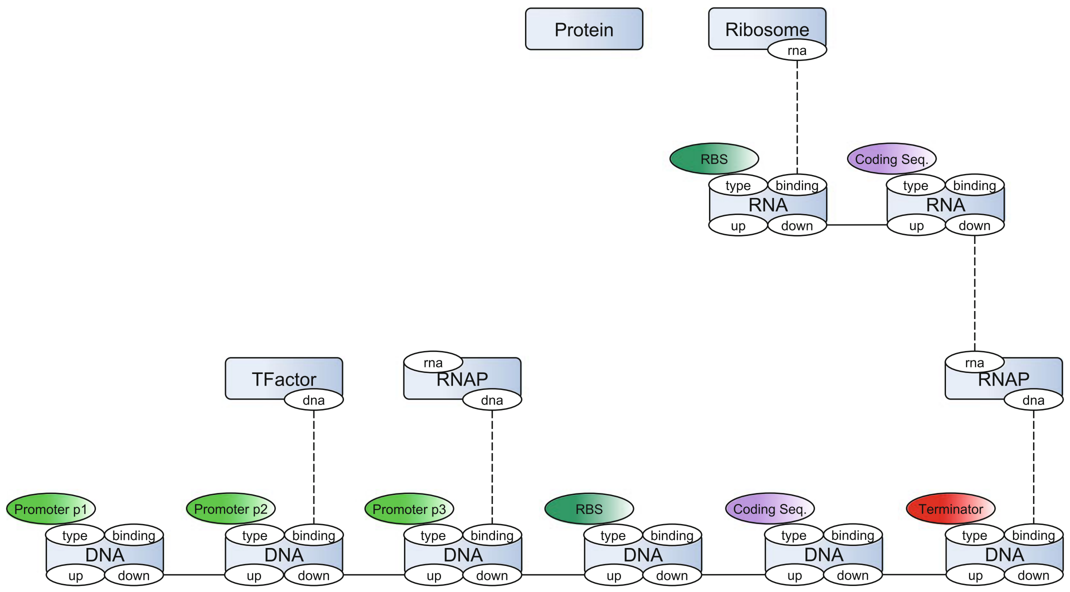
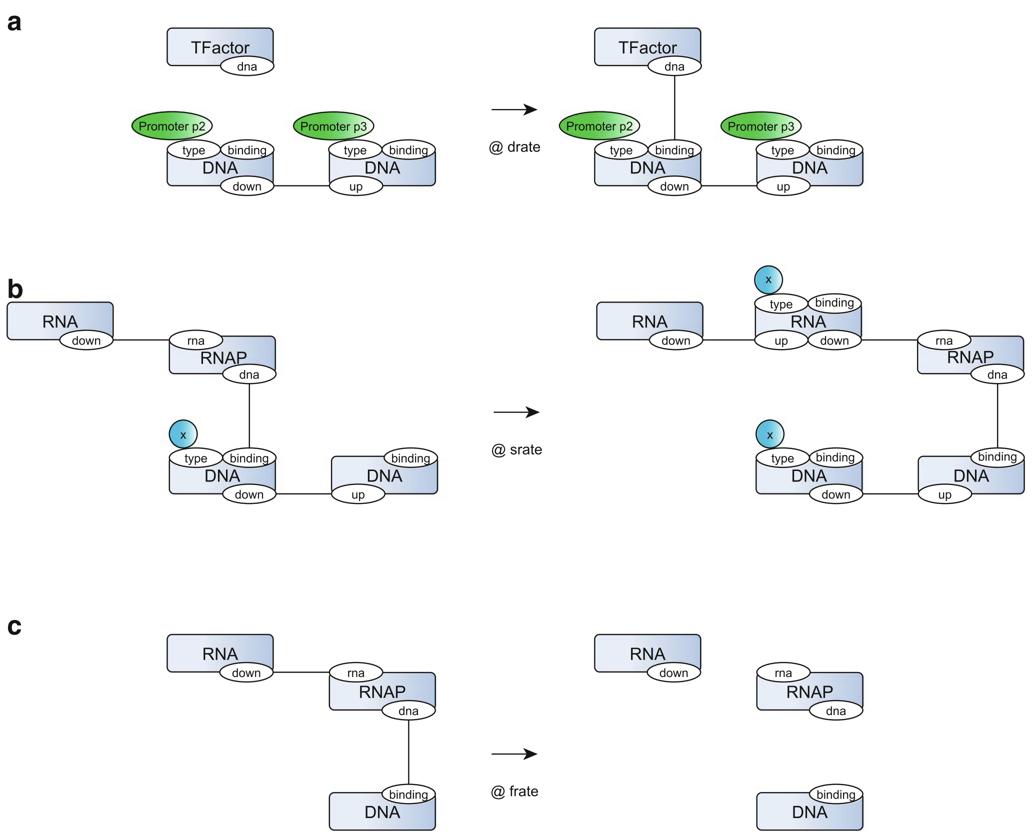


Fig. 10 The agents involved in the Kappa BioBrick Framework. At bottom, a chain of DNA agents (linked by their upstream and downstream sites) representing the BioBrick parts; each part would have a unique identifier contained as a state to the "type" site. mRNA is also represented as such a chain, transcribed from the DNA by RNAP and subject to translation by Ribosomes. The TFactor and Protein agents are placeholders automatically generated by the framework, and must be manually refined into the corresponding protein

A BioBrick part in the framework consists of one or more DNA agents connected in a chain and tagged with a unique identifier, for example adopted from the Registry of Standard Biological Parts (http://partsregistry.org/). Each BioBrick part also has an RNA representation defined in a similar manner. Given a set of BioBrick parts, the framework will automatically generate these DNA and RNA agents, along with RNA polymerases (RNAP) and Ribosomes involved in transcription and translation, and placeholder Transcription Factor and Protein agents to be manually refined by the user (Fig. 10). Organizing the agents in this manner allows us to represent the transcriptional and translational interactions modularly, without placing any limitations on protein behavior (which may vary a great deal more) beyond activity as a transcription factor.

The framework also generates a concise and complete set of rules (with associated kinetic rates) necessary to describe the activity of these BioBrick parts in an idealized space-homogeneous chassis. Once the rules for a virtual part have been established, it can be composed with other virtual parts in a modular manner analogous to the use of actual BioBrick parts in synthetic biology. The paragraphs that follow may be thought of as a recipe with which to create a set of Kappa rules for a certain part.

The crux of the framework lies in three basic rule templates: "docking," "sliding," and "falloff," shown in Fig. 11. Every rule



```
'A (Docking)' DNA(type~p2,binding,down!1), DNA(type~p3,binding,up!1), TFactor(dna) ->
                   DNA(type~p2,binding!2,down!1), DNA(type~p3,binding,up!1), TFactor(dna!2)   @ drate
'B (Sliding)' DNA(type~x,binding!2,down!1), DNA(binding,up!1), RNAP(dna!2,rna!3), RNA(down!3) ->
                   DNA(type~x,binding,down!1), DNA(binding!2,up!1), RNAP(dna!2,rna!3),
                   RNA(type~x,binding,down!3,up!4), RNA(down!4)                               @ srate
'C (Falloff)' DNA(binding!1), RNAP(dna!1,rna!2), RNA(down!2) ->
                   DNA(binding), RNAP(dna,rna), RNA(down)                                     @ frate
```

Fig. 11 Instances of the trinity of rules in the Kappa BioBrick Framework: "docking," "sliding," "falloff." (A) The "docking" rule illustrates "transcription factor binding" rule for a promoter from Table 1: the binding of a transcription factor to the correct binding region on the promoter BioBrick part, given that there is no RNAP present (note that the binding region on the downstream DNA agent is free). (B) The "sliding" rule illustrates a "translation" rule for a protein coding sequence: an RNAP agent moving along the DNA chain and transcribing the appropriate RNA as it does so. (C) The "falloff" rule illustrates both universal RNAP falloff and the falloff for a terminator: the dissociation of RNAP from an unspecified DNA part (the terminator falloff rule would have a higher rate constant than the universal rule), and the simultaneous release of the constructed RNA chain (there may be any number of RNA agents upstream of the one shown)

generated by the framework, bar the universal RNA degradation rule, is instantiated from one of these templates. At the transcriptional level:

- The "docking" rule template (Fig. 11a) defines the mechanism of transcription factor and RNA polymerase binding to BioBrick promoter parts. The regulatory effect of transcription factors on BioBrick promoters are simply refinements of these "docking" rules (Fig. 12).

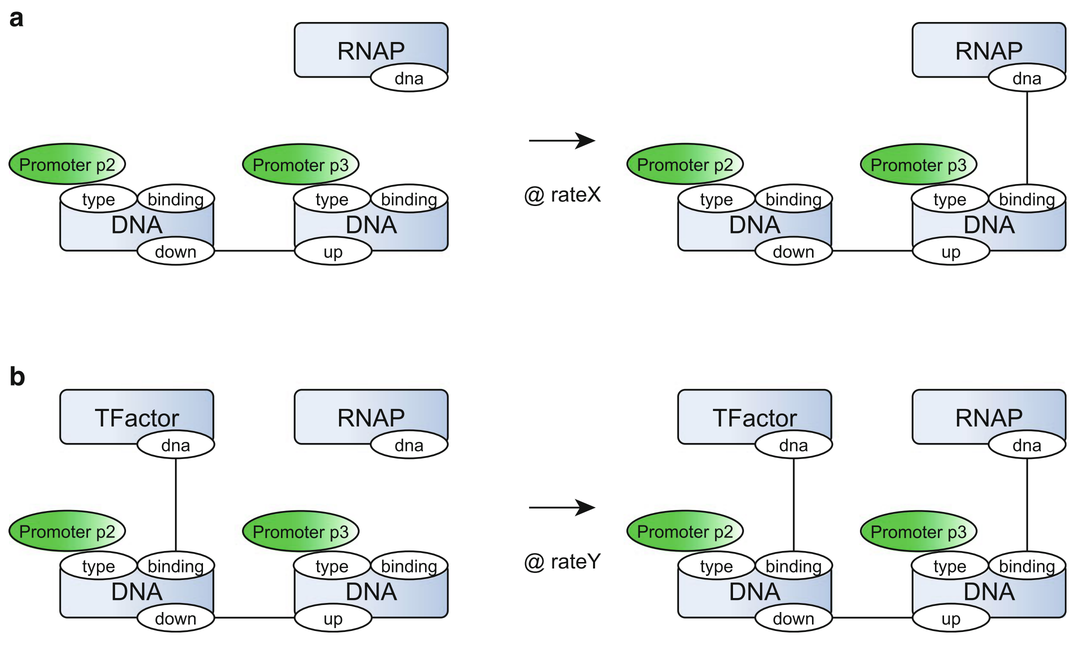


Fig. 12 Adapting the "docking" rules to model transcription factor regulation upon promoter activity. (**a**) The binding of RNAP to a promoter whose regulatory site is free, governed by a rate constant rateX. (**b**) The binding of RNAP to a promoter whose regulatory site is bound to a generic transcription factor, similarly governed by a rate constant rateY. The transcription factor is an activator in rateX is negligible and rateY is significant; conversely, if rateX is significant but rateY negligible, then the transcription factor is a repressor. By manipulating the exact values of rateX and rateY we are able to define leaky promoters or promoters of varying strengths. Finally, we can represent promoters with multiple regulatory binding sites, for example cooperative regulation, simply by elongating the promoter part (increasing the number of DNA agents involved) and specifying the effect of multiple transcription factors upon RNAP attraction

- All BioBrick DNA agents require associated rules that describe the transcription of the part from DNA to RNA caused by RNA polymerase; these are the "sliding" rules (Fig. 11b).
- "Falloff" rules (Fig. 11c) deal with the detachment of RNA polymerase and transcription factors from the chain of DNA agents representing BioBrick parts. Although this may in theory happen on any BioBrick part, BioBrick terminators have a higher falloff rate due to their function in preventing further transcription downstream.

Similarly at the translational level, ribosomes may "dock" at ribosome binding sites, may "slide" across protein coding sequences to translate the appropriate protein, and may "falloff" from the RNA chain. The framework also describes the potential degradation of RNA agents at a uniform rate, unlike DNA agents which are assumed to be immutable barring user-defined rules.

The rate constants associated with these framework rules correspond to values that would ideally be known to a rich database of synthetic BioBrick parts. For example, the effective rates of

Table 1
The rules generated by the Kappa BioBrick Framework at both transcriptional and translational levels

	Transcription	Translation
Promoter	**Transcription factor binding (*)** *Transcription factor unbinding (*)* **RNA polymerase binding (*)** ***Transcription initiation*** ***Transcription readthrough***	
Ribosome binding site	***Transcription***	Ribosome binding
Protein coding sequence	***Transcription***	***Translation initiation*** ***Translation (*)***
Terminator	*Transcription termination* ***Transcription readthrough***	
Other	*RNA polymerase falloff*	*Ribosome falloff* RNA degradation

"Docking" rules are denoted in *bold*, "sliding" rules in *bold italics*, and "falloff" rules in *italics*. The promoter rules marked with an asterisk must be manually refined according to promoter structure (repressor or activator, number of transcription factor binding sites). The similarly marked protein coding sequence translation rule requires user input of the specific protein agent created

"sliding" rules would correlate with measurements of PoPS for transcriptional activity and RiPS for translational activity. "Docking" rules would relate to such measures as promoter and RBS strength: transcription factor affinity, RNAP attraction as a function of transcription factor binding, and ribosome affinity. The kinetic rate for "falloff" rules is determined by such factors as RNAP error rate and terminator efficiency.

It must be clearly stated that the above rules do not take into account the actions of the proteins after they are synthesized or any pathways that they may be involved in, bar their possible effect as a transcription factor. Such considerations (for example, protein degradation or kinase activity) are up to the individual modeler to incorporate separately in addition to rules generated by the framework. An example of such an effort is provided in Subheading 4 to follow. Of course, at this point the advantages of modeling in Kappa that we have already explored in Subheading 2 apply.

The rules of the Kappa BioBrick Framework may thus be categorized along three dimensions: the BioBrick part they are associated with, whether they are involved at the transcriptional or translational level of cellular machinery, and which of the three templates they are based on (Table 1). To illustrate them further, we now present a model of the hallowed Elowitz repressilator created from BioBrick parts [16].

3.2 The Repressilator in the Kappa BioBrick Framework

The repressilator combines three simple synthetic genes connected in a loop (Fig. 13), such that the product of each gene represses the next gene in the loop and is in turn repressed by the product of the previous gene. The output is oscillatory as the repression of one protein by the second allows the third to build up to repress the second, which in turn allows the first to accumulate to repress the third, and so forth (Fig. 14).

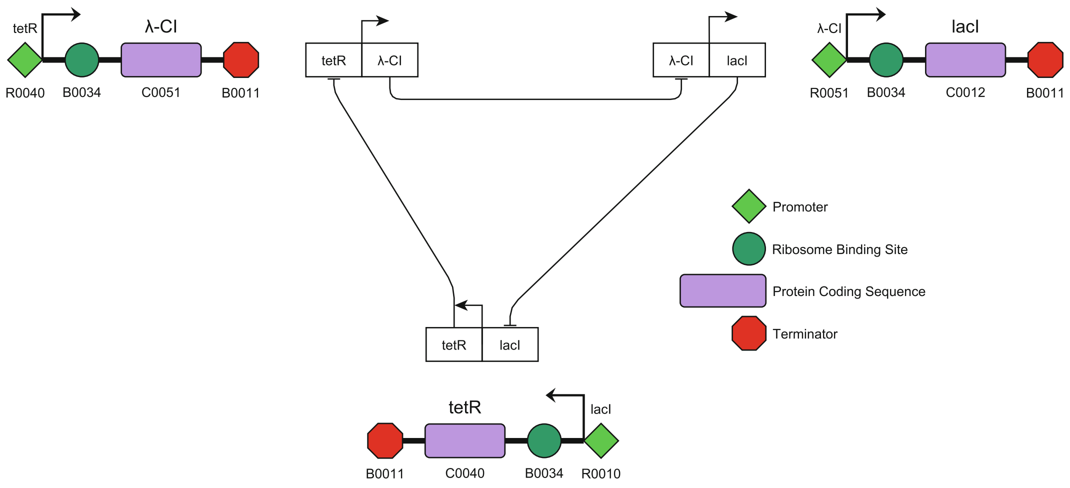


Fig. 13 The Elowitz repressilator constructed from BioBrick parts, with each repressor inhibiting production of the next repressor in the loop through the action of the appropriate promoter. Each part may be represented as a modular component (a set of agents and associated rules) in the Kappa BioBrick Framework, and the composition of a number of these parts creates a full circuit as shown. The central representation hides the RBS and terminator components of the BioBrick device, since we reuse the same parts throughout the model

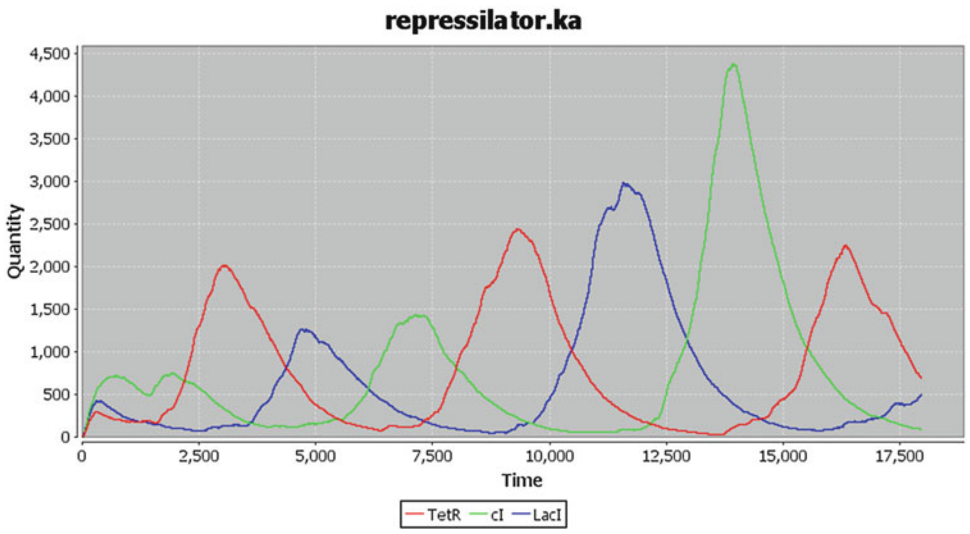


Fig. 14 Simulation results for the Elowitz repressilator modeled using the Kappa BioBrick Framework. The system stutters to begin with due to the initial conditions of the model favoring none of the three repressors, but as soon as one begins to assert dominance the system falls into its familiar oscillating pattern

Our model consists of seven types of agents (DNA, RNA, Ribosome, RNAP, and the three repressor proteins involved) and 61 rules. The agents and rules are composed as described previously for eight BioBrick parts: three protein coding sequences (LacI, TetR, and λ-CI), their corresponding promoters, and one each of a terminator and an RBS. Each promoter is designed to be cooperatively bound by up to two of its associated transcription factors (in this case a repressor), and thus is modeled as a concatenation of four DNA agents (two dedicated to repressor binding, one for RNAP binding, and a linker or spacer separating the promoter from any preceding part). Every other BioBrick part is modeled as a single DNA agent. The terminator and RBS are reused across all three devices.

In the paragraphs to follow, we describe the rules of the model according to the BioBrick parts they are associated to. Readers may find Table 1 useful in keeping track of the parts and their associated rules.

The first three rules of the model are RNAP and ribosome "falloff" rules that apply to all DNA agents (BioBrick parts), and an equally universal RNA degradation rule.

Each promoter requires up to $2n\,(2^{n-1}) + 2^n + 2$ rules, where n is the number of transcription factor binding sites, (2^{n-1}) represents the number of occupancy contexts in which a binding or unbinding can occur, and 2^n is the number of transcription factor binding configurations that the RNAP may depend on:

- Up to $n\,(2^{n-1})$ rules to describe the possibly cooperative binding of the transcription factor(s).
- Up to $n\,(2^{n-1})$ rules more to describe unbinding.
- Up to 2^n rules to describe the binding of the RNAP.
- A singleton rule describing initiation of the transcription process.
- A further singleton rule to deal with transcription readthrough by transporting any RNAP that arrives at the linker to the RNAP binding site (thus allowing the action of further rules).

In this model, $n = 2$ for all promoters, hence each promoter requires 14 rules to describe its behavior. All of the transcription factor binding and unbinding rules are combinatorial in the number of binding sites, in this case two, since (for example) the binding of a transcription factor at one site is conditional on whether or not the other site is already bound. Similarly, we model four rules to describe the binding of RNAP: one for when no transcription factors are bound, one each for when either of the sites are bound, and one for when both sites are bound. The eight rules describing the binding of various agents to the part are all variations on the "docking" base rule, the four rules describing unbinding are variants on the “falloff” base rule, and both transcription initiation and transcription walkthrough are based on the “sliding” base rule.

Of course, if the combinatorics of the promoter are known not to not fully affect its activity, for example the rate of RNAP binding, enumerating all occupancy contexts is possibly an overly detailed solution. Currently it is unclear how one might benefit from the potential regularities in promoter structure by saving on the cost of describing them.

The protein coding sequences each require three rules, all based on the “sliding” template:

- A rule for its transcription.
- A rule for initiating translation.
- A rule for the actual translation of the protein.

As may be surmised from the rule descriptions, only the first rule is active at the transcriptional level; the other two operate at the translational level.

The RBS is described via two rules:

- A transcription rule (“sliding”).
- A ribosome binding rule (“docking”).

Transcription of the RBS DNA agent creates an equivalent RNA agent, upon which the ribosome binding rule may then fire as the first step of the translation process.

The terminator also requires two rules:

- A “falloff” rule with enhanced rate to represent its efficiency at terminating transcription.
- Transcription readthrough (“sliding”) if its terminator function fails.

The first of these terminator rules trumps the generic RNAP “falloff” rule, thus making the terminator much more efficient at removing RNA polymerase from the DNA chain than any accidental falloff.

The proteins created in the model act as transcription factors as described in the promoter rules, but have no other activity beyond three rules describing their degradation. These final three rules are the only rules in the model not specified by the Kappa BioBrick Framework, but are required to produce the oscillations characteristic of the repressilator.

The initial conditions of the model contain one of each of the BioBrick devices shown in Fig. 13, as well as a population of RNA polymerase and ribosomes. By identifying the LacI, TetR, and λ-CI proteins as the observables of interest, simulation of the agents and rules described above generates a time course similar to that shown in Fig. 14.

Note, of course, that the framework as described above is not the only way to model BioBrick parts in Kappa; other, less modular formalisms may be of equal use to the modeler. Furthermore, by no means does it completely capture every interaction necessary in fine detail. As an example, RNA degradation process makes no distinction between exo- and endonucleases; it simply destroys RNA with free binding and downstream sites, which is roughly equivalent to half the activity of an exonuclease. A simple modification to the base framework would be to differentiate between the two. As another example, the transcription factors in the framework associate specifically with their binding site upon the promoter, whereas in reality they search for their appropriate binding sites by "sliding" along the DNA chain; a possible extension of the framework would be to take this action into account. A further extension might be to make use of the linker portion of the promoter and the promoter's transcription readthrough rule to represent the length of noncoding region between BioBrick devices. We might even refine the RNA agent to model the ability for multiple Ribosomes to exist on a single mRNA, thus fully distinguishing between PoPS and RiPS.

Current levels of implementation allow the skeleton of the framework—the unrefined "docking," "sliding," and "falloff" rules described above—to be automatically generated via LMS-Kappa. Given a suitably detailed source, such as an ideally populated Registry of Standard Biological Parts, one can imagine automating the modeling of such aspects as promoter structure (repressor or activator, number of transcription factor binding sites and their cooperativity if any) and rate information (transcription factor affinity, RBS and terminator efficiency, PoPS, and RiPS). However, we envisage that the modeler will always need to provide detailed protein interactions outside the scope of the framework so as to maintain the flexibility required of them.

These protein interactions are the subject of the next section, where we build upon the repressilator just described.

4 Extending the Repressilator: Light-Mediated Synchronization

The Kappa BioBrick Framework also provides a basis for modeling component devices that combine into a larger and more complex system. Individual modules may be developed and verified independently, simplifying the process of breaking down the overall effort into manageable units amenable to repeated cycles of refinement and extension. This is accomplished in a similar manner to the modularity of the individual BioBrick parts—individual devices, consisting of a set of agents, rules, and associated variables, are simply concatenated together into a single larger model. When composing in this manner, care must be taken that a specific biological entity (an agent and its sites) is depicted in the same way across different modules to ensure that it is shared as intended.

In addition, depending on the level of granularity desired, further rules may be used to describe the protein interactions necessary for the model (for example the activity of a signaling pathway linking a translation product to a transcription factor). These may be safely added independently of the Kappa BioBrick Framework and thought of as a modular component themselves. We proceed to elaborate on these points in the paragraphs to follow.

One of the weaknesses of the Elowitz repressilator is that without external regulation the system oscillations are extremely imprecise and do not last over time, so that the oscillations of multiple cells each running their own version of the repressilator rapidly desynchronise or die off. To address this problem, the 2010 Edinburgh iGEM team designed a light-mediated communication system based on the Elowitz repressilator (Fig. 15), involving the establishment of three independent channels of communication

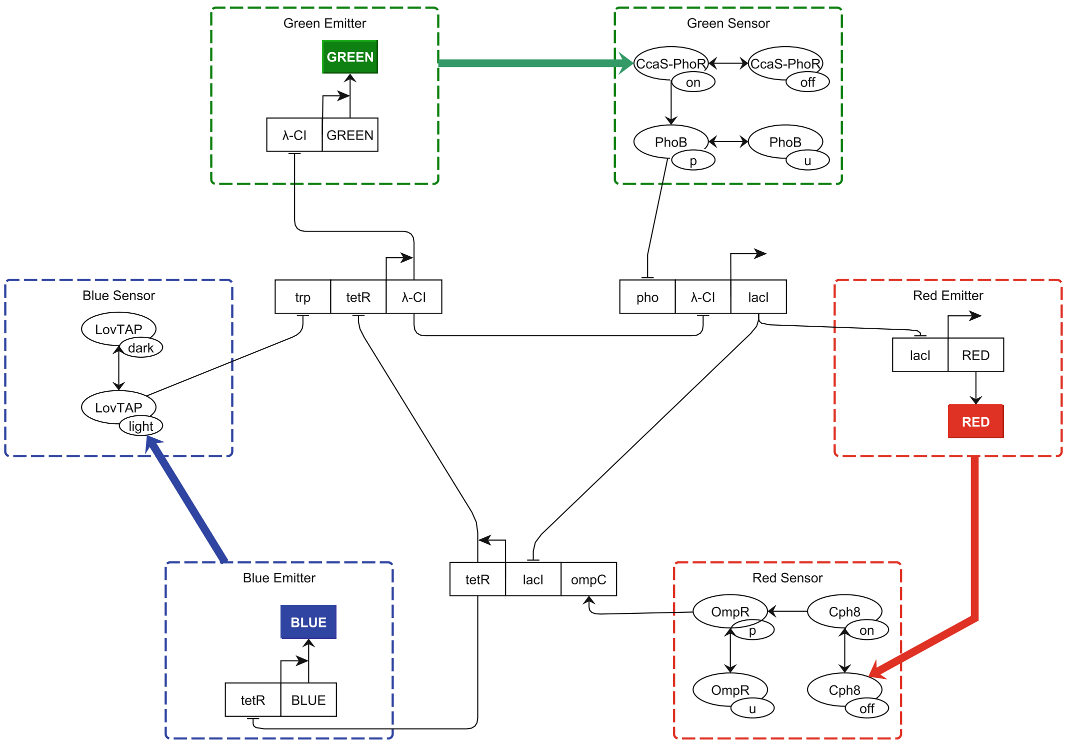


Fig. 15 Modeling light emitting and light sensing pathways coupled to an Elowitz repressilator. At the center the oscillating repressilator regulates the emission of light in the system. The light sensing pathways then provide input to this central regulator via a second promoter coupled to the same coding sequence, reinforcing or adjusting responses to synchronize the system. The central repressilator and the light emitting pathways are genetic components, built solely around the Kappa BioBrick Framework, but the light sensing pathways are protein interaction components that must need be manually specified by the modeler (*see* Fig. 16). (Reprinted from [17] with permission from Elsevier)

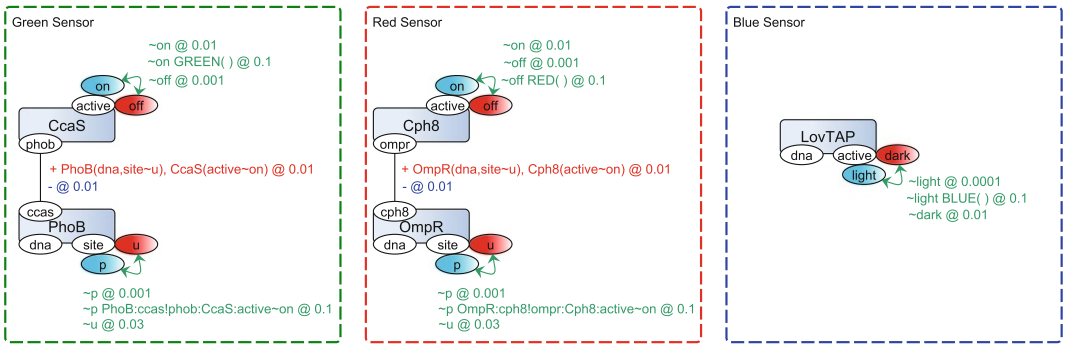


Fig. 16 Visualization of the light sensing pathways in the light-mediated repressilator. The red and green sensors are both two-component regulatory systems that induce changes in conformation when subjected to light of the appropriate wavelength. The green sensor is usually "off," but when subjected to green light activates CcaS, which binds to and phosphorylates PhoB, which then acts as a transcriptional repressor. In contrast, the red sensor is usually "on" and activating a downstream promoter; when red light is sensed, Cph8 is inactivated, which stops binding to OmpR and thus encourages its dephosphorylation, which in turn prevents it from acting as a transcriptional activator. In summary, incoming green light activates a repressor, as opposed to incoming red light which deactivates an activator instead; both thus have a symmetrical inhibitory effect on their coupled protein in the core repressilator. The blue light sensor is an allosteric hybrid protein with a simple single-component mechanism: activation via blue light switches LovTAP to its "light" configuration, whereupon it is free to act as an inhibitor upon the core repressilator

in different spectral wavelengths [17]. The goal of the project was to develop a multicellular system capable of self-reinforcing collective synchronization, with the eventual aim of enabling bacterial populations to interact with each other as well as with purely electronic systems via light.

As a first step to achieving the above, each gene product in the repressilator loop is used to repress the production of light of a particular wavelength. In other words, each wavelength is associated with the lack of an associated repressor (red light with lack of LacI, blue light with lack of TetR, and green light with lack of λ-CI).

In the meantime, the light sensing pathways (Fig. 16) provide input to the core repressilator to reinforce the oscillatory response. The model assumes that they are constitutively expressed, unlike the proteins generated by the core repressilator. For each of the three repressilator proteins, a second promoter is coupled from the light sensing pathway to the original promoter, such that inhibition from either repressor is sufficient to inhibit its production; this assumption is based on combinatorial promoter characterisation in Cox et al. [18]. The activated green and blue sensors explicitly inhibit LacI and λ-CI respectively, while the activated red sensor ceases promotion of TetR, hence inhibiting it implicitly.

If the effect of the light emitting pathways is to "broadcast" the current state of the host's repressilator to its neighbors, the proposed effect of the light sensing pathways is to "adjust" its state to match those of its neighbors instead. When activated, ideally the sensors either reinforce the current state of the repressilator in the host (because it is already synchronized with its neighbors) or modify it to bring it closer to the desired behavior. This communication occurs at a much faster rate than the transcriptional interactions of the core repressilator, but it is unclear at this stage (when approaching this system guided only by intuition) whether or not this synchronization will actually occur if implemented in vivo.

Modeling the light sensors and emitters in conjunction with the core repressilator, and analysis of the proposed system as a whole, is thus crucial to understanding it. This model of light communication is composed from seven components (the core repressilator, along with emitters and sensors for the three wavelengths), each of which we may consider a functional model in its own right. Each of these could then be validated individually before being combined into a single host, and then tested in both communicating and noncommunicating colonies.

The first iteration of development adopts the core Elowitz repressilator component (Fig. 17a) described previously in Subheading 3. We then add the light emitting components (Fig. 17b) to the model, producing oscillating light outputs linked to the oscillations of the core repressilator shown in Fig. 13. Biologically, these components were all synthetically created to purpose in the iGEM project: green light emissions based on the standard firefly (*Photinus pyralis*) luciferase enzyme which then underwent site-directed mutagenesis to create red light, while blue light emissions were developed from a bacterial luciferase from *Xenorhabdus luminescens*. Both the central repressilator and the light emitting components are formed solely from the BioBrick parts—the proteins produced and involved have no function other than as a transcription factor, and thus there is no necessity to manually model protein–protein interactions.

The next development iterations focus on the light sensing pathways. Similar to the light emission pathways, these were biologically engineered to purpose: the red light sensor was based on a bacterial phytochrome, the blue light sensor on a plant phototropin, and a novel fusion protein was designed as a green light sensor. Unlike the core repressilator and light emitting components, the proteins involved in the light sensing pathways are assumed to be constitutively expressed in the cell. Thus they are modeled purely on the protein interaction level without any use of the Kappa BioBrick Framework. By not allowing the produced transcription factors to bind to the repressilator (Fig. 17c), the response of the individual light-sensing pathways may be tested via perturbation analysis. Only when satisfied with their performance need this restriction be lifted (Fig. 17d), resulting in the complete model of the synthetic circuit.

Usually throughout the development of a model, rules must be refined and their accompanying kinetic parameters tuned to

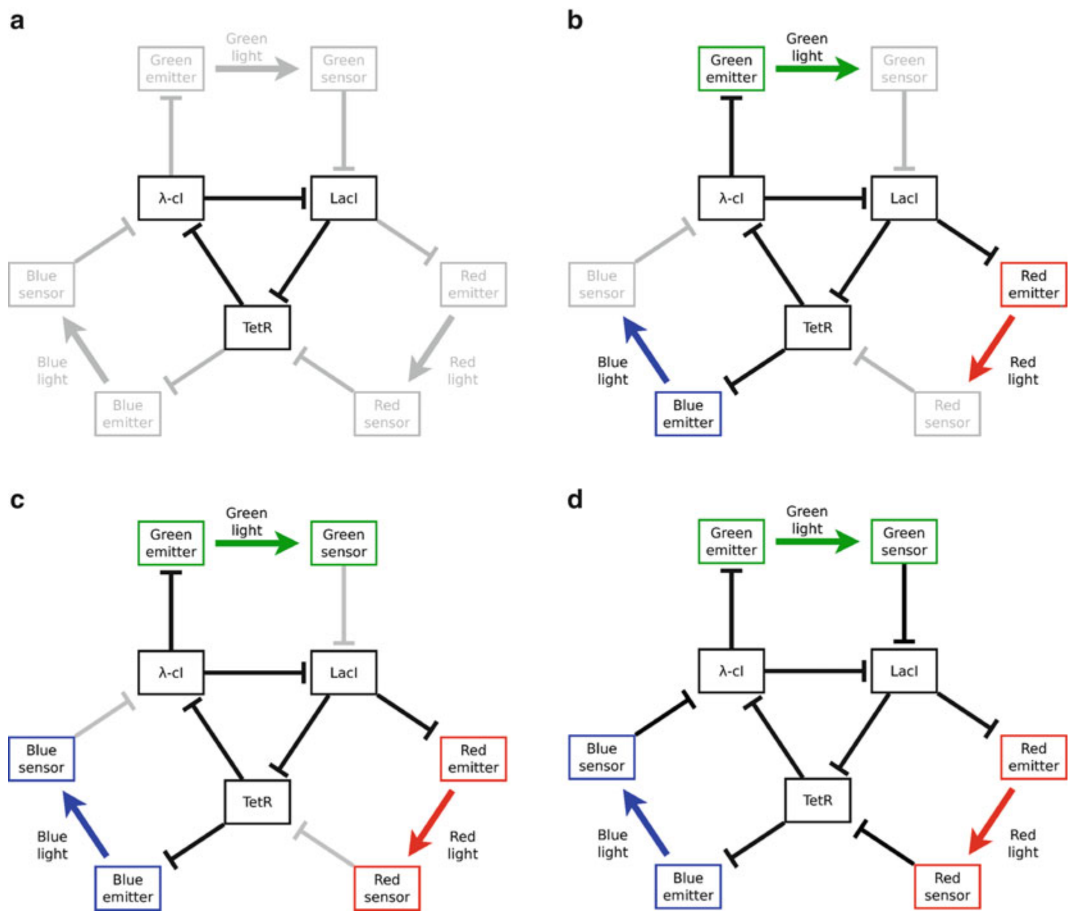


Fig. 17 Iterative development of the light communication system, from core repressilator to complete network, showing the modular nature of the system. (**a**) Core repressilator. (**b**) Light emission pathways. (**c**) Light sensing pathways. (**d**) The complete network. (Adapted from [17] with permission from Elsevier)

maintain the desired behavior (in this example, oscillations). Due to constraints on time and equipment, these rate parameters may be originally derived from in silico trial-and-error analysis rather than in vivo experimentation. The current version of KaSim also allows for the declaration of global variables that can be used to control the rates of multiple rules, thus affording another layer of modularity in the development of the model since rules with similar function (for example, readthrough transcription) may be controlled with fewer parameters. The latest version of the light-mediated repressilator model provided to the reader on RuleBase (http://rulebase.org) makes use of this capability.

Up to this point we have simply tied the BioBrick parts comprising the repressilator to the protein interactions of three simple signaling pathways. The intracellular model may now be extended further to simulate the behavior of an idealized virtual colony of

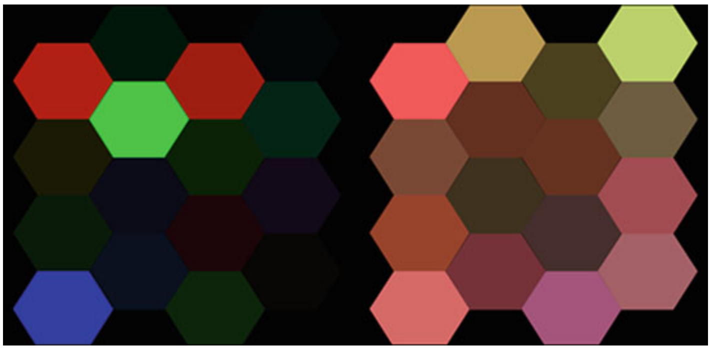


Fig. 18 A sample simulation snapshot of the multicellular light-mediated repressilator model. Isolated cells (noncommunicating) are shown on the *left*, each running a separate repressilator and broadcasting a clearly defined output wavelength independently of its neighbors. The communicating cells on the *right* show the effect of light-mediated synchronization on the overall state of each cell: their output wavelengths are muddier but more closely correlated

bacteria communicating with each other using the light produced within each cell. This may be done by utilizing the simplifying assumption that the bacteria are nonmotile and closely packed in a two-dimensional hexagonal biofilm. Additional rules to the model represent the communication of light between neighboring cells, each responsible for maintaining its own repressilator. A snapshot of a sample simulation is shown in Fig. 18, with isolated noncommunicating cells shown on the left and communicating cells on the right.

The light levels in each cell are recorded at each sample point during the simulation, along with the colony mean light levels and the standard deviation of the individual cell light levels from these colony means. Accurate average behavior is recorded by measuring results (Fig. 19) over sufficiently long simulations. These results show that a communicating colony has less time-averaged deviation in light levels between cells, and therefore increased synchronization.

So far in this chapter, we have explored a number of Kappa models in synthetic biology, from those based on protein–protein interactions to the purely genetic. We have attempted to highlight both the inherent strengths of adopting a rule-based approach to modeling, and the manner in which the Kappa BioBrick Framework structures the modeling of BioBrick parts whilst leaving room for extension and refinement. To conclude from here, we take a look at the future of rule-based modeling in Kappa, paying particular attention to the following questions: what is missing from the methods described above, and how we may fully exploit our advantages.

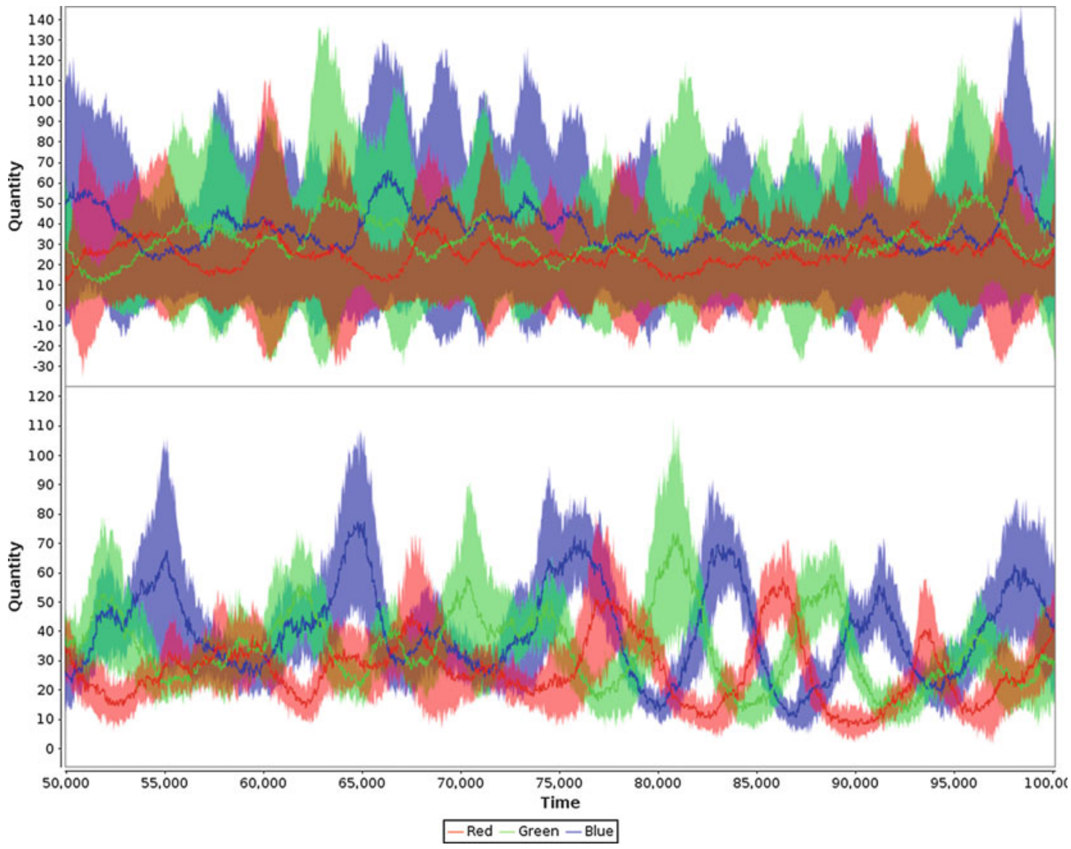


Fig. 19 Comparison of mean cell light levels in 4 × 4 colonies of cells, both isolated (*top*) and communicating (*bottom*) between cells in the virtual colony. Shaded areas on the graphs show standard deviation of cell light levels from colony mean. Communicating colony shows reduced time-averaged standard deviation (R:9.46, G:11.76, B:13.49) compared to the isolated colony (R:25.71, G:36.35, B:39.44), and hence increased coherence in cell activity levels. Units (time and quantity) are arbitrary. (Adapted from [17] with permission from Elsevier)

5 Looking to the Future

A fundamental challenge to synthetic biology is the engineering of biological parts with behavior that is well-defined in relation to other parts. This not only requires controlled and precise measurement protocols, but also a modeling language for the formalization of these interactions. The Kappa rule-based biological modeling language provides a means to this end—a set of rules describing the ways in which biological entities interact with other entities present in the system at a tunable level of detail.

The rule-based approach embodied by languages such as Kappa has a number of advantages over its alternatives, not least in the fact that it greatly reduces the combinatorial complexity of the system description in comparison to more traditional reaction-based models. Modular rules describe the functionality of individual BioBrick parts in an easy-to-understand manner, thus aiding comprehension of the underlying biology, and can be easily reused both in different contexts within a model and across multiple different models. The associated Kappa BioBrick Framework is well-suited to working with individual parts and this can only improve in the future with the development of dedicated support tools and databases. The modularity of the approach also makes it easy to apply an iterative development methodology to the problem, easing the process of building, analyzing, and understanding the model.

On the other hand, not all models can be constructed automatically from genetic components in the form of BioBrick parts, and the modeler input necessary to fill in this knowledge gap is non-trivial and may daunt a newcomer to the language. Furthermore, the framework itself is neither as readable nor as cleanly modular as we may wish. To address the second of these we turn to a pair of extensions to the Kappa language (their application in this manner first proposed by Elaine Murphy): meta-Kappa [19] and Colored Kappa.

Meta-Kappa introduces agent hierarchies in which children may inherit sites from their parents and may add new sites or replicate existing ones as well. Rules may be written with both parents and children, but all agents must be concretely specified (at the bottom level of the hierarchy) in the initial conditions of the model. The use of meta-Kappa is especially useful in concisely representing promoters with multiple binding sites for transcription factors.

Colored Kappa improves the modularity of rules reliant on the BioBrick part number (for example, the transcription rule depicted in Fig. 10) by allowing rules to contain variables, and rates of such a rule to be a function of said variables. Without Colored Kappa, we are forced to write a transcription rule for every BioBrick part in the system; with Colored Kappa, we can simply write a single rule conditioned on the BioBrick part number as a variable.

Both meta-Kappa and Colored Kappa have value in simplifying the manual user input to the Kappa BioBrick Framework. However, neither of them have yet been integrated with the framework, and it would be useful to accomplish this in the future.

Another extension to Kappa of value to the framework is Thermodynamic Kappa [20]. Thermodynamic Kappa helps us better express the relationship between forward and backward kinetic rates (i.e., the equilibrium constant), such as that given by the different affinities between the transcription and translation machinery with various portions of the DNA and RNA. These affinities realize the difference in potential energies between the various configurations. In Thermodynamic Kappa we can state these directly using energy patterns, that is, a list of connected agents with their associated energies. In the repressilator model this could be used to express the differences in the "docking," "sliding," and "falloff" equilibrium constants for the various DNA–RNAP or RNA–Ribosome complexes. For instance, by declaring that the energy for RNAP bound to a DNA terminator sequence is higher than that of RNAP bound to any other DNA part, we may implicitly state that RNAP dissociates from DNA more readily when sitting on top of a terminator. In the same way, cooperativity among transcription factors when binding to DNA can be described by a lower energy for the configuration in which two transcription factors are bound to DNA than when one of them is bound without the other.

An implementation of Thermodynamic Kappa based on the Metropolis algorithm [21] is available in LMS-Kappa. Further work is in progress to implement meta-Kappa and other extensions of Kappa not mentioned in this chapter on top of the LMS-Kappa core. Most importantly for the subject of this chapter, a preliminary implementation of the Kappa BioBrick Framework is also available as an extension in the standard distribution of LMS-Kappa (https://github.com/sstucki/lms-kappa). The extensibility of LMS-Kappa will eventually allow all of these language extensions to be adopted by the modeler according to the needs of the model, thus greatly contributing to even more flexible methodologies for modeling synthetic biology in Kappa.

What else can we think of for the future of Kappa modeling in synthetic biology? One possible step forward could be to extend the prokaryotic Kappa BioBrick Framework for eukaryotic purposes, allowing the strengths of Kappa to come into play for fields such as RNA-focused synthetic biology. We have already discussed in Subheading 3 the refinement of elements of the existing framework to better reflect known biology (for example the better models of RNA degradation) or to take better advantage of the benefits of Kappa in alleviating combinatorial complexity (for example in modeling the search process that occurs when transcription factors search for a suitable promoter binding site).

Finally, an exciting new domain of application is the concept of whole-cell modeling as exemplified by Karr et al. [22]. Whole-cell models combine multiple modeling approaches, such as ordinary differential equations and flux balance analysis, to integrate all of a cell's molecular components and their interactions in a single computational object. As applied to synthetic biology, we might begin by proposing a virtual chassis, which would allow us to test in silico how proposed BioBrick devices (modeled for example using the Kappa BioBrick Framework) affect the physiology of the host. A grander vision would then be to have a selection of not only different chassis (for example, yeast, *E. coli*, *Bacillus subtilis*, and so forth), but also different versions of chassis components (for example, signaling pathways) that would allow potential modelers to customize their model via plug and play.

Another application of the whole-cell modeling paradigm would be to test the Kappa BioBrick Framework's ability to account for resources (RNA polymerases and ribosomes) shared with other processes extant within the host. As stated throughout this chapter, Kappa allows for the refined modeling of transcriptional and translational activity, for instance facilitated search by transcription factors and multiple ribosomes upon mRNA. In a whole-cell model similar to that implemented by Karr et al., we might on the one hand use other modeling techniques such as flux balance analysis for metabolism, and delegate the interactions of DNA and RNA to a high resolution Kappa model. This would allow us to apply the best modeling techniques to their most appropriate usage, and tie them all together in a coherent whole.

To conclude this chapter, we would like to reiterate the concepts we have covered thus far. We have introduced the rule-based modeling language Kappa via a series of four examples: the first kinase-phosphatase model to acquaint readers with the basic Kappa language and visualizations thereof; the second to repeat this in a synthetic biological example of a unidirectional "on" switch to make concepts more concrete; the third to introduce the Kappa BioBrick Framework via the repressilator; and the fourth to extend this repressilator with protein-based light-mediated communication. We have also briefly introduced recent developments in the Kappa language, such as LMS-Kappa, aimed at implementing and extending the usability of the Kappa BioBrick Framework.

Of course, we should warn the reader that part of the excitement of synthetic biology is that it is a field in flux; therefore the methodology which we propose should itself be thought of as an open-ended process, more as a series of guidelines, than a closed standard. On the other hand, we hope that throughout this chapter we have convinced the reader that Kappa and the Kappa BioBrick Framework can be of help to a modeler with a computational background in understanding synthetic biology, thus providing fresh insight into this nascent and exciting discipline.

References

1. Danos V, Laneve C (2004) Formal molecular biology. Theor Comput Sci 325:69–110
2. Blinov ML, Faeder JR, Goldstein B et al (2004) BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. Bioinformatics 20:3289–3291
3. Bachman JA, Sorger P (2011) New approaches to modeling complex biochemistry. Nat Methods 8:130
4. Gillespie DT (1977) Exact stochastic simulation of coupled chemical reactions. J Phys Chem 81:2340–2361
5. Feret J, Krivine J (2013) KaSim3 reference manual. https://github.com/jkrivine/KaSim/blob/master/man/KaSim_manual.pdf?raw=true. Accessed 26 June 2013
6. Danos V, Honorato-Zimmer R, Stucki S (2013) KaSpace: a language for the combinatorial assembly of biological complexes. The First Annual Winter q-bio Meeting
7. Pedersen M, Phillips A, Plotkin G (2013) A high-level language for rule-based modelling. http://mdpedersen.azurewebsites.net/papers/lbs-kappa.pdf. Accessed 15 September 2014
8. Goldbeter A, Koshland DE (1981) An amplified sensitivity arising from covalent

modification in biological systems. Proc Natl Acad Sci U S A 78:6840–6844

9. Wang D, Amornsiripanitch N, Dong X (2006) A genomic approach to identify regulatory nodes in the transcriptional network of systemic acquired resistance in plants. PLoS Pathog 2:e123
10. Moore JW (2012) Foundation technologies in synthetic biology: tools for use in understanding plant immunity. PhD thesis, University of Edinburgh
11. Gardner TS, Cantor CR, Collins JJ (2000) Construction of a genetic toggle switch in *Escherichia coli*. Nature 403:339–342
12. Shetty RP, Endy D, Knight TF Jr (2008) Engineering BioBrick vectors from BioBrick parts. J Biol Eng 2:1–12
13. Marchisio MA, Stelling J (2008) Computational design of synthetic gene circuits with composable parts. Bioinformatics 24(17):1903–1910
14. Chandran D, Bergmann FT, Sauro HM (2009) TinkerCell: modular CAD tool for synthetic biology. J Biol Eng 3:19
15. Marchisio MA, Colaiacovo M, Whitehead E et al (2013) Modular, rule-based modeling for the design of eukaryotic synthetic gene circuits. BMC Syst Biol 7:42
16. Elowitz MB, Leibler S (2000) A synthetic oscillatory network of transcriptional regulators. Nature 403:335–338
17. Stewart D, Wilson-Kanamori JR (2011) Modular modelling in synthetic biology: light-based communication in *E. coli*. Electron Notes Theor Comput Sci 277:77–87
18. Cox RS, Surette MG, Elowitz MB (2007) Programming gene expression with combinatorial promoters. Mol Syst Biol 3
19. Danos V, Feret J, Fontana W et al (2009) Rule-based modelling and model perturbation. Lect Notes Comput Sci 5750:116–137
20. Danos V, Harmer R, Honorato-Zimmer R (2013) Thermodynamic graph-rewriting. Lect Notes Comput Sci 8052:380–394
21. Metropolis N, Rosenbluth AW, Rosenbluth MN et al (1953) Equation of state calculations by fast computing machines. J Chem Phys 21:1087
22. Karr JR, Sanghvi JC, Macklin DN et al (2012) A whole-cell computational model predicts phenotype from genotype. Cell 150:389–401

Chapter 7

Modular Design of Synthetic Gene Circuits with Biological Parts and Pools

Mario Andrea Marchisio

Abstract

Synthetic gene circuits can be designed in an electronic fashion by displaying their basic components—Standard Biological Parts and Pools of molecules—on the computer screen and connecting them with hypothetical wires. This procedure, achieved by our add-on for the software ProMoT, was successfully applied to bacterial circuits. Recently, we have extended this design-methodology to eukaryotic cells. Here, highly complex components such as promoters and Pools of mRNA contain hundreds of species and reactions whose calculation demands a rule-based modeling approach. We showed how to build such complex modules via the joint employment of the software BioNetGen (rule-based modeling) and ProMoT (modularization). In this chapter, we illustrate how to utilize our computational tool for synthetic biology with the in silico implementation of a simple eukaryotic gene circuit that performs the logic AND operation.

Key words Standard Biological Parts, Pools of signal carriers, Genetic modules, Rule-based modeling, Circuit design

1 Introduction

The construction of biological circuits requires a set of basic components and the definition of a shared input/output. The latter is what permits components' composition into more complex devices and, eventually, circuits. Circuit's elements should be well characterized. This implies the knowledge of their transfer function that permits the calculation of the output that corresponds to any given input. When all the basic components are associated with a predictive mathematical description, the model of an entire circuit arises from the composition of the models of each of its units.

According to these ideas, which come from electrical engineering, Drew Endy [1] suggested considering the Standard

Biological Parts as fundamental genetic components for bacterial circuits. They represent DNA traits with a well-defined function in transcription or translation. At the MIT Registry they are divided into categories such as promoters, ribosome binding sites (RBS), coding regions (both for proteins and small RNAs), and terminators. Beside them, the fluxes of RNA polymerases (PoPS, Polymerase Per Second) and ribosomes (RiPS, Ribosome Per Second) were indicated as shared input/output representing a biological counterpart of the electrical current. Since RNA polymerases and ribosomes handle an exchange of information among circuit components they were referred to as *common signal carriers*.

Computational design and simulations of synthetic gene circuits require that each single Standard Biological Part is described by a model that represents the interactions that take place inside the Part itself. Furthermore, each Part needs to be *composable*, i.e., it has to be associated with an interface that handles the computation and the exchange of fluxes of signal carriers with the rest of the circuit. Therefore, Parts have to be independent *modules*. Different Parts have diverse complexity that should be reflected properly by a model description. Promoters, for instance, can be regulated by several transcription factor proteins. They bind the DNA independently or cooperatively and either recruit RNA polymerase or compete with it. Regulatory proteins can bind many operators. Therefore, a promoter model might require a high number of species and reactions colliding with the problem of the *combinatorial explosion*.

We have recently proposed a way to model Standard Biological Parts such that they are independent composable modules associated with an exhaustive description of the species and the reactions they involve [2]. This demands the use of two software tools: ProMoT (Process Modeling Tool, available at http://www.mpi-magdeburg.mpg.de/projects/promot) [3] and BioNetGen (www.bionetgen.org) [4]. ProMoT allows—through the Model Definition Language, MDL [5]—the representation of complex systems as a network of modules that communicate via the exchange of fluxes; BioNetGen permits the computation of the species and reactions present in a biological system by means of a rule-based modeling approach. ProMoT, furthermore, provides a user with a graphical interface where gene circuits can be designed in a drag and drop manner, as in electronics, making this task easy and intuitive.

In this chapter we describe in detail how to use our ProMoT add-on for synthetic biology. Concepts and technicalities are explained by illustrating how to design a simple eukaryotic gene circuit that mimics the Boolean AND function.

2 Methods

2.1 A New Method for Circuits' Modeling with Parts and Pools

Though made of Standard Biological Parts, synthetic gene circuits are often represented by models that neglect this intrinsic and entire transcription units are frequently considered as basic modules instead. If they lead to protein production, they are associated with two ordinary differential equations, one for the mRNA and the other for the protein dynamics. Transcription and translation regulation are described, for instance, by Hill functions that lump complex interactions into two different parameters only: the Hill coefficient, which quantifies the regulatory factors' cooperativity, and the Hill constant, which corresponds to the concentration of the regulatory factors necessary either to decrease (repression) or to enhance (activation) the mRNA or protein production of one half of their maximum value. If, on the one hand, this modeling approach has the advantage of limiting the number of kinetics parameters, on the other hand it does not provide a detailed description of the interactions that take place at DNA and mRNA level. Moreover, the dynamics of molecules such as RNA polymerases and ribosomes is not even considered explicitly.

Starting from the ideas of Drew Endy, we proposed a new way of designing and modeling synthetic gene circuits [6]. In our framework, Standard Biological Parts are basic, independent modules. Each Part is described by both a model based on full mass-action kinetics and an interface that handles the exchange of fluxes of common signal carriers with other circuit components. Moreover, we decided to consider another three molecules as common signal carriers: transcription factors, small RNAs, and chemicals because of the role they play in regulating both transcription and translation. In a circuit scheme, signal carriers are associated with new entities, the *Pools* (their icons—together with all the other symbols used throughout this chapter—are shown in Fig. 1). Pools represent the place where free molecules of signal carriers are stored. These molecules determine the circuit performance and are biological *potentials*. Therefore, Pools can be considered as *bio-batteries*. Moreover, Pools of transcription factors and small RNAs are interfaces between transcription units, whereas Pools of chemicals are interfaces between the cell and the surrounding environment (*see* Fig. 2). Pools can also have a different function when they are used as containers for some specific (e.g., enzymatic) reactions [7]. In this case, they are independent of any signal carrier.

2.2 Parts and Pools Within the ProMoT/BioNetGen Framework

ProMoT permits the handling of biological Parts and Pools as independent modules of genetic networks. Parts and Pools are described into separate MDL files. They are organized in three sections: *modules*, *terminals*, and *links*. Following the notation in [8], modules for bio-circuit components are: *storage-intras*,

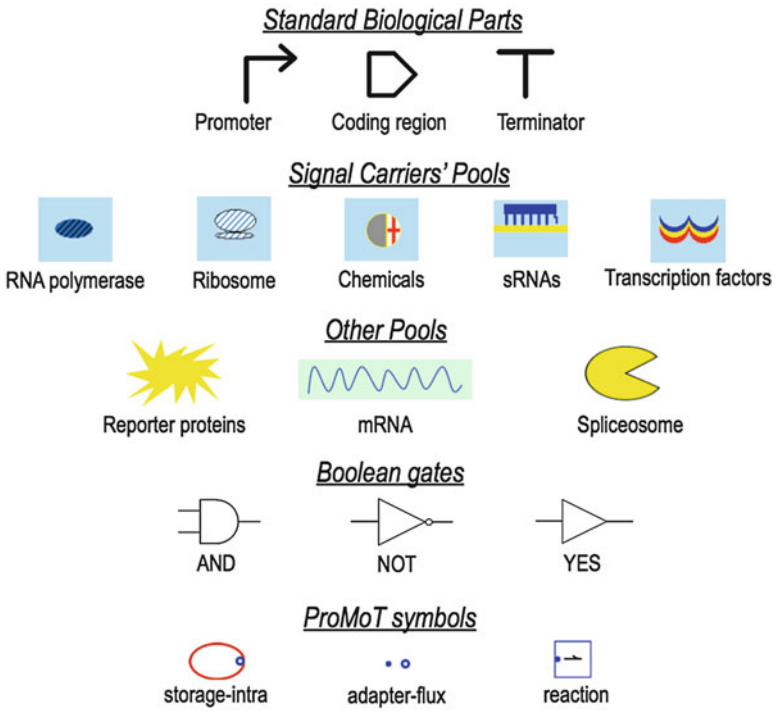


Fig. 1 Symbols. Standard Biological Parts are represented by the SBOL [17] icons, whereas Boolean gates are symbolized as in electronics. Pools' pictures come from our previous publications [2, 6]. Molecules not present in figure (e.g., the dicer enzyme and the RISC complex) are not associated with a symbol, in a circuit scheme, but just with their name. ProMoT symbols contain input/output terminals

adapter-fluxes, and *reactions*. Storage-intras represent species; adapter-fluxes are entities that handle the exchange of information between circuit's components (namely, fluxes and species concentrations); reactions are modules for the biochemical interactions among species. Terminals constitute the interface of Parts and Pools. In a circuit scheme, terminals are wired to each other to establish communication between modules. Inside each component, they are connected to adapter-fluxes. Links, finally, are the internal wires for each Part and Pool, i.e., the connections between storage-intras, adapter-fluxes, and reactions.

Within this framework, it is straightforward to model Parts and Pools with full mass-action kinetics. Moreover, adapter-fluxes permit the calculation and the exchange of signal carrier's fluxes as requested by our modeling method.

Parts such as promoters and the bacterial RBS together with the mRNA Pool for eukaryotic cells can contain many species and reactions depending both on the number of regulatory factors acting on them and on their DNA/mRNA binding sites. In this

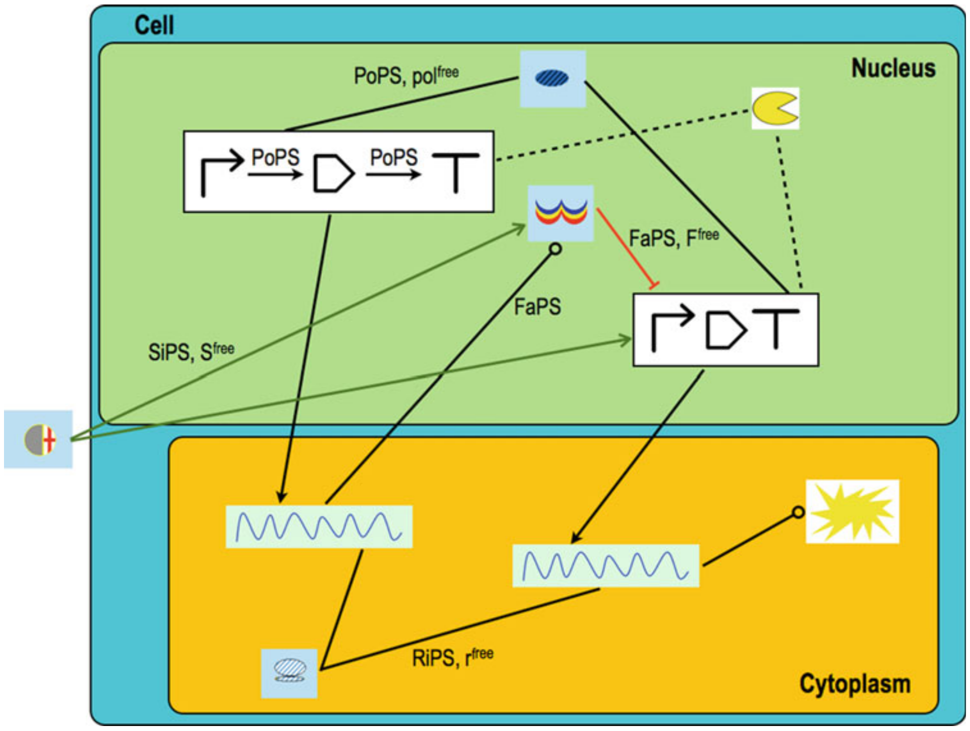


Fig. 2 Signal carriers' fluxes and Pools. Pools' role in a gene circuit scheme is illustrated via a simple eukaryotic one-step cascade. Each transcription unit is made of three Parts (enclosed in a *black rectangle*) through which RNA polymerases flow. Both units are connected to the RNA polymerase Pool with which they exchange a PoPS flux and from which they are constantly updated (during a simulation) about the concentration of available molecules of RNA polymerases (pol^{free}). The first transcription unit encodes for a repressor that acts on the promoter of the second transcription unit and regulates the production of a reporter protein. The connection point between the two units is the nuclear Pool where repressor molecules are stored after being transcribed into the corresponding mRNA Pool in the cytoplasm. FaPS (Factor Per Second) is the general name for a flux of transcription factors. Here we have a FaPS flux from the cytoplasm to the nucleus and another one exchanged between the repressor Pool and the promoter of the second transcription unit. Repressors are inactivated by an external signal, a chemical, whose molecules (S^{free}) are present in a Pool, placed outside the cell, which makes a connection between the circuit and the environment. Chemicals interact with both free and bound-to-DNA repressors. Therefore, chemicals' Pool exchanges a SiPS (Signal Per Second) flux with both the repressor Pool and the second transcription unit's promoter. In the cytoplasm, the ribosome Pool plays the same role as the RNA polymerase Pool in the nucleus. It is connected to the two mRNA Pools (the repressor and the reporter one) with which it exchanges a RiPS flux and to which it communicates the concentration of available ribosomes (r^{free}). The Pools of the spliceosome and the reporter proteins (the former placed in the nucleus, the latter in the cytoplasm) complete the circuit design. Though they are not considered as signal carriers, they communicate with the connected Parts and Pools via fluxes and concentration values as well. This kind of information exchange is symbolized, in this figure, either via a straight or a dashed line. *Black arrows* represent transcription; *black lines* ending with a circle, translation. The *red line ending with an orthogonal bar* stands for transcription repression, whereas the *green arrows* mean transcription activation

case, a proper computation of species and reactions is achieved via the software BioNetGen that implements a rule-based modeling approach. Biological systems are described in a very abstract way by specifying: (1) the kind of molecules they contain and all the states these molecules can assume (*molecule type*); (2) the species present at the beginning of the computation and their corresponding concentrations (*seed species*); (3) the *rules* that govern species interactions together with the associated kinetics parameter values. From this input, BioNetGen derives a full description of the species and interactions that make up the system.

ProMoT and BioNetGen have complementary features. ProMoT permits a clear definition of interacting biological modules, BioNetGen provides an exhaustive computation of species and biochemical reactions. We merged these two characteristics in order to generate complete models of highly complex genetic Parts and Pools. Our software requires an input file where a Part's (Pool's) structure and kinetic parameter values are indicated. For instance, the input file of a promoter contains the number of operators which are present along the promoter sequence, if they bind repressor or activator proteins, the interaction between the regulatory factors (cooperative or not), the effect of chemicals on these proteins. This information is then converted into a BNGL (BioNetGen Language) file that serves as an input for a call to BioNetGen. It returns a Part (Pool) description in terms of species and reactions. However, fluxes of signal carriers are missing and the module still lacks an interface. Therefore, the output file of BioNetGen is parsed and translated by our software into an MDL file that finally encodes a complete model for Parts (Pools). This file can be loaded into ProMoT and the corresponding module can be used as a component inside a genetic circuit.

2.3 Parts and Pools Generation

Our ProMoT add-on for Synthetic Biology is a collection of Perl and Python scripts. Each script generates either a single Part (Pool) or one Part and one Pool if they are strictly connected to each other (such as the eukaryotic coding region and the corresponding mRNA Pool). Parts and Pools are encoded into MDL files. Promoters, RBSs, and the eukaryotic mRNA Pool are accompanied by a further MDL file that contains a specification for the reactions they host. Parts and Pools models are built according to the content of their input files. In the following we explain how to fill in each Part and Pool input file properly.

2.3.1 Promoter (promoter.inp)

Promoter name: a string that specifies the name of the MDL file encoding for the promoter model. This string, joint to "_REACTIONS_LIB", also gives the name to the MDL file that contains the promoter reactions.

Repressors' number: an integer greater than or equal to zero.

Operators per repressor: each repressor acting on the promoter can bind either a single or a group of operators. Accepted values are integers greater than or equal to one. They are separated by commas (e.g., 1, 2, 2). If the repressors' number is equal to zero, this entry is automatically set to zero as well.

Repressors: names of the repressors that bind the promoter. Each name is made of the repressor kind ("Ra": active repressor or "Ri": inactive repressor) followed by an integer to unequivocally identify both the repressor and the group of operators it binds to. Repressors' names are separated by commas. Furthermore, repressors are enumerated in a crescent order (e.g., Ra1, Ri2, Ra3). This allows the assigning of higher numbers to the operators that are more distant from the TSS (Transcription Start Site). In synthetic promoters, indeed, repressor operators are placed between the TSS and the TATA box.

Chemicals binding repressors: two kinds of chemicals are allowed: inducers ("I"), and corepressors ("C"). Following the notation in [9], inducers bind and deactivate active repressors (I + Ra), whereas corepressors bind and activate inactive repressors (C + Ri). Since active repressors do not need to interact with chemicals to bind the DNA, they can be associated also with "N" that stands for no signal. Each chemical type ("I", "C", and "N") is accompanied by an integer that indicates the repressor with which the signal interacts (e.g., N1, C2, I3). Chemicals' names are separated by commas.

Activators' number: an integer greater than or equal to zero, as in the repressors' case.

Operators per activator: one or more operators are associated with each activator. These integer values are separated by commas (e.g., 2, 2).

Activators: activators are of two kinds: active ("Aa") and inactive ("Ai"). The former are able to bind the DNA in their original configuration, the latter require a structural modification caused by the action of a chemical. The name of an activator contains the activator's kind and integer—like in the repressor case (e.g., Ai1, Aa2). Activators' operators are placed upstream of the TATA box. Analogously to repressors, activators should be numbered in ascending order: the higher the integer, the further from the TATA box the activator binds.

Chemicals binding activators: the same three possibilities as in the repressors' case: "I", "C", and "N". Inducers bind inactive activators (I + Ai), whereas corepressors target active activators (C + Aa). Active activators can be associated with no signal ("N") as well (*see* **Note 1**). Chemicals require the specification of the activator they bind (integer label—e.g., I1, N2).

Repressors' cooperativity: a list of integers separated by commas (e.g., 2, 3). Each integer represents a repressor whose molecules bind cooperatively to their group of operators. Inside a group of repressor operators, we assume that the closest to the TSS is the strongest one, i.e., it has the highest affinity towards the corresponding repressor. All the other operators have a lower affinity that, however, increases when the adjacent operator (on the right, i.e., with a smaller integer label) is occupied by a repressor protein. Only homo-cooperativity is taken into account, i.e., repressors do not cooperate if they bind different operator groups. When no repressor shows any cooperativity, one should simply write "no".

Activators' cooperativity: a list of integers separated by commas, as in the repressor case. However, in a group of activator operators, the most distant from the TATA box is the one to which the highest affinity is assigned. Moreover, in presence of cooperativity, the rightmost operator within a group must be occupied in order to recruit RNA polymerase on the DNA. If no activator binds cooperatively, "no" has to be written.

P_free: promoter concentration (*see* **Note 2**).

Alpha-repressors: binding rate constant between a repressor and the corresponding operators. The value is unique in absence of cooperativity, otherwise three values have to be specified: (1) the binding rate constant with the strongest operator; (2) the binding rate constant with the all the other (weak) operators without cooperative effects; (3) the binding rate constant with the weak operators in presence of cooperative effects. These three values are enclosed in round brackets and separated by semicolons. Values that refer to different repressors are separated by commas (e.g., 1e9, (1e9; 1e6; 1e7), (1e8; 1e5; 1e8)).

Beta-repressors: repressors' unbinding rate from their operators. It follows the same rules as alpha-repressors (e.g., 10, (0.1; 10; 2), (1, 10, 1.5)—*see* **Note 3**).

Alpha-activators, beta-activators: see alpha-repressors and beta-repressors.

K_d-repressors: repressors' decay rate. One value has to be specified for each repressor acting on the promoter. Different values are separated by commas.

K_d-activators: decay rates for promoters' activators.

Gamma-repressors: rate constant associated with the binding of an inducer to a repressor bound to an operator. This reaction is irreversible. A single value has to be specified for each repressor. If an active repressor is not under the action of inducers, its corresponding gamma has to be set to zero.

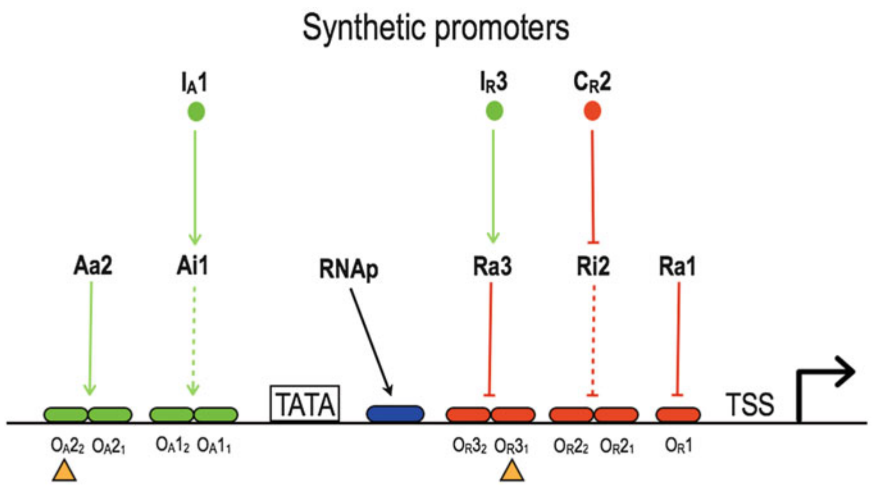


Fig. 3 Synthetic promoters. This synthetic promoter is regulated by three repressors and two activators. One repressor (Ri2) and one activator (Ai1) are inactive (*dashed lines*) in their ground conformation. Therefore, they need to be bound by a chemical to get access to the DNA. In particular, Ri2 needs a corepressor (C_R2—but a software user has to write C2 only in the input file) to switch off transcription and Ai1 requires an inducer (I_A1—see above) to recruit RNA polymerase (RNAp) on the DNA. Both repressors and activators are numbered in crescent order with respect to their distance from the TSS and the TATA box, respectively. RNA polymerase site (*blue spot*) is placed between the repressors' and activators' operators (*red* and *green spots*, respectively). Ri2, Ra3, Ai1, and Aa2 bind groups of two operators. Operators belonging to the same group are supposed to be equivalent unless the corresponding transcription factor proteins bind to them cooperatively. Here, Ra3 and Aa2 show a cooperative behavior. An *orange triangle* indicates the operator that has the stronger affinity towards the respective transcription factor. Operators' names specify which repressor/activator they bind and their position within a group. For instance: 0_R3_2 is the label for the second operator inside the group of operators that bind repressor number 3 (Ra3 in our case). The meaning of the different *lines*/*arrows* is explained in Fig. 2

Gamma-activators: binding rate constant of corepressors to the corresponding activators when the latter are bound to promoter operators. Active activators that are not associated with any chemical demand gamma to be equal to zero.

K1: binding rate constant between RNA polymerase and its promoter binding site.

K_1: RNA polymerase's unbinding rate from the promoter.

K2: transcription initiation rate. It represents the promoter strength.

K2_lk: transcription leakage rate.

Compartments: "y" (yes) for eukaryotic cells, "n" (no) for bacterial cells.

Local directory: path to the directory where the Parts and Pools scripts are stored.

BioNetGen directory: path to the directory where the BioNetGen executable files are stored.

See Fig. 3 for a representation of a synthetic promoter.

2.3.2 Coding Region and Mature mRNA Pool (EU_coding.inp)

Gene name: coding region's name. This name is given to the MDL file that contains the model for the coding region. A separate file ("m_mRNA_*gene name*.mdl") is assigned to the Pool of mature mRNA transcribed from this Part. The library file encoding for the reactions that take place into the mRNA Pool is called "*gene name*_REACTIONS_LIB.mdl".

Riboswitches' number: an integer value greater than or equal to zero that specifies how many riboswitches are placed on the 5′-UTR (UnTranslated Region) of the mRNA's coding region.

Riboswitches' kind: five different types of riboswitches are taken into account: S (single aptamer); TON (Tandem aptamers, hOmo-effectors, No cooperativity); TEN (Tandem aptamers, hEtero-effectors, No cooperativity); TOC (Tandem aptamers, hOmo-effectors, Cooperativity); TEC (Tandem aptamers, hEtero-effectors, Cooperativity—*see* **Note 4**). Strings for the kind of riboswitches are separated by commas (e.g., tec, toc).

Effectors per riboswitch: as with every chemical in our model, effectors can be of two different kinds: inducers ("I") if they activate translation, and corepressors ("C") when, on the contrary, they repress translation. Single-aptamer riboswitches are controlled by a unique effector; tandem riboswiches are bound by two different chemicals (hetero-effectors) or a single one (homo-effectors). Each effector is specified by its kind followed by an integer label (*see* **Note 5**); two different effectors binding the same tandem riboswitches should be enclosed between round brackets and separated by a semicolon. Effectors binding different riboswitches are separated by commas (e.g., (I1; I2), C3).

siRNAs' number: an integer value greater than or equal to zero to define how many small interfering RNAs bind the mRNA at the 3′-UTR (repressing, in this way, translation).

K_fd: fast degradation decay rate; mRNA decays at this rate when bound to the complex RISC-siRNA.

Sites per siRNA: each siRNA can bind more than a single site along the mRNA 3′-UTR. Integer values corresponding to different siRNAs should be separated by commas (e.g., 2, 1).

Polymerase leakage: "y" (yes) if the promoter connected to the coding region contains operators, "n" (no) if this promoter is constitutive.

Gene length: to be expressed as base-pair number. It is used to calculate both RNA polymerase and ribosome elongation rate.

K1r: ribosomes' binding rate constant to the mRNA.

K1_r: ribosomes' unbinding rate from the mRNA.

K2r: translation initiation rate.

Theta riboswitch: binding rate constant between the effectors and the corresponding aptamers. One value is required for single aptamer riboswitches (S) and tandem riboswitches homo-effectors without cooperativity (TON); two values for tandem riboswitches hetero-effectors without cooperativity (TEN—one rate constant for each effector) and for tandem riboswitches homo-effectors with cooperativity (TOC—the first value is the binding rate without cooperativity effects; the second, bigger one, refers to the cooperative binding. The two aptamers are equivalent, i.e., they have the same affinity towards their common effector); three values for tandem riboswitches hetero-effectors with cooperativity (TEC—the first value is the binding rate constant to aptamer number 1—the strongest one; the second value is the binding rate constant to aptamer number 2—the weakest one—when number 1 is free; the third value is the binding rate constant to aptamer number 2 when number 1 is bound by the corresponding effector, i.e., with cooperativity effects. The second value should be lower than the third one). Whenever more than a single value is required, different rate constants are enclosed in round brackets and separated by semicolons (e.g., (1e9; 1e6; 1e8), (1e5; 1e7)).

Csi riboswitch: effectors' unbinding rate from riboswitches' aptamers. It follows the same rules as for theta riboswitch (e.g., (0.1; 10; 1), (100; 10)).

Theta siRNA: binding rate constant between small interfering RNAs and the mRNA. A unique value per siRNA is required. Values are separated by commas (e.g., 1e6, 1e5).

Csi siRNA: siRNAs' unbinding rate from the mRNA. Also in this case, a single value is required for every siRNA (e.g., 0.1, 0.01).

K2r_lk: translation leakage rate.

K1y: binding rate constant between the spliceosome (Y) and the immature mRNA.

K_1y: spliceosome unbinding rate from the immature mRNA.

K2y: splicing rate.

Km: mRNA maturation rate. It lumps all the mRNA maturation processes in the nucleus and the mature mRNA transport in the cytoplasm.

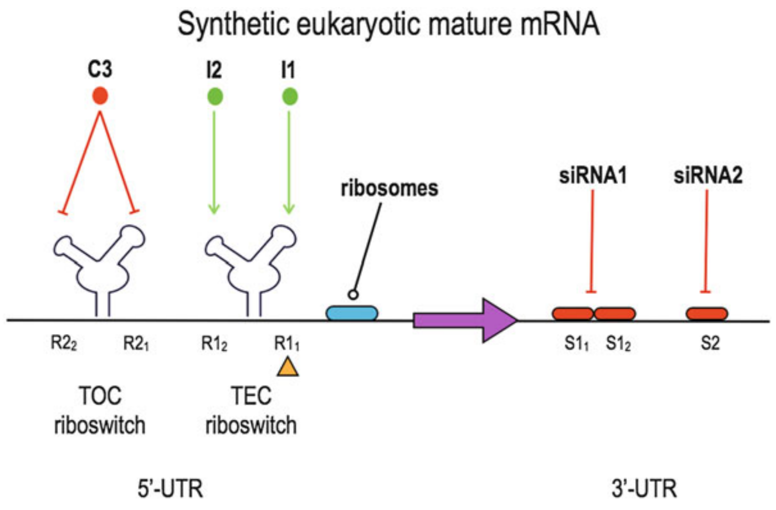


Fig. 4 Synthetic eukaryotic mature mRNA. The mRNA here represented hosts two tandem riboswitches on the 5′-UTR and two groups of siRNA binding sites (red spots) on the 3′-UTR. Aptamers can lie in two states: *on*, if they allow ribosome binding (i.e., translation), and *off*, otherwise. Riboswitch 1 (R1, kind: TEC) is "off" in its ground state and gets activated by two inducers (I1 and I2) that bind cooperatively. Aptamer 1 has stronger affinity towards its effector as indicated by the orange triangle. Riboswitch 2 (R2, kind: TOC) is, in contrast, always "on" unless at least one of its aptamers is bound by the corepressor C3. Since siRNAs can only repress translation, via RNA interference, this gene (*purple arrow*) is synthesized only when both I1 and I2 are present and C3 together with siRNA1 and siRNA2 are absent. The *blue spot* represents the site where ribosomes bind. As for the various *arrows* and *lines*, their meaning is explained in Fig. 2

Zeta_r: protein synthesis rate. It represents the rate at which ribosomes leave the mRNA releasing a new protein.

K_tr: protein nuclear import rate.

K_dp: protein decay rate (*see* **Note 6**).

Local directory: path to the directory that contains the scripts for Parts and Pools.

BioNetGen directory: path to the directory that contains the BioNetGen executable files.

Product_name: name of the protein encoded by the coding region (*see* **Note 7**).

Promoter name (link): name of the promoter connected to the coding region (*see* **Note 8**).

In Fig. 4, a scheme for the mature mRNA corresponding to a synthetic eukaryotic coding region Part is provided (*see* **Note 9**).

2.3.3 siRNA Coding Region and Pool (sirna.inp)

siRNA name: a string that identifies the two MDL files associated with an siRNA. One contains the model for the siRNA coding Part, the other for the corresponding cytoplasmic Pool. The former is named "*siRNA name*_coding.mdl", the latter: "*siRNA name*_-pool.mdl".

Polymerase leakage: this entry has to be set to "y" (yes) if the promoter connected to the siRNA Part contains operators, otherwise it should be "n" (no).

siRNA length: specified as base pairs number, it is used to calculate RNA polymerase elongation rate.

K1d: binding rate constant between the dicer enzyme and the siRNA (*see* **Note 10**).

K_1d: dicer enzyme unbinding rate from the siRNA.

K2d: splicing rate.

Km: siRNA maturation time.

K1risc: binding rate constant of the RISC complex to the siRNA.

K_1risc: RISC complex unbinding rate from the siRNA.

Promoter name (link): name of the promoter connected to the siRNA Part (*see* **Note 8**).

2.3.4 Terminator (EU_terminator.inp)

Terminator name: a string for the name of the MDL file containing the terminator model.

mRNA k_d: mRNA decay rate. This value is passed to a mature mRNA Pool or to an siRNA Pool (*see* **Note 11**).

Zeta: RNA polymerase unbinding rate from the DNA.

2.3.5 Chemicals' Pool (sigpool.inp)

Signal name: a string that appears in the name of the Pool of chemicals (or environmental signals): "*signal name*_pool.mdl".

Connected to an input source: if this entry is set to "y" (yes), chemicals are produced into a Part or a Pool present in the gene circuit. Otherwise, chemicals are supposed to be generated into their Pool with a rate constant *k_s*.

Sig_free: initial concentration of chemicals' molecules.

K_s: chemicals' generation rate constant in (M/s).

K_d: chemicals' decay rate.

Compartments: "y" (yes) for eukaryotic cells, "n" (no) for bacterial cells.

2.3.6 Transcription Factor's Pool (tfpool.inp)

Tf name: a string to which the suffix "_pool" is added to form the name for the MDL file that encodes the model for a transcription factor Pool.

Type: two kinds of transcription factors are allowed: monomers ("m") and dimers ("d").

Tf state: in its native configuration a transcription factor can be either active ("a") or inactive ("i").

Signals: "y" (yes) if the transcription factor interacts with chemicals, "n" (no) otherwise (*see* **Note 12**).

Tf free: initial concentration of transcription factors.

K_d: transcription factor decay rate (*see* **Note 13**).

Delta: dimerization rate constant.

Epsilon: dimer separation rate (into monomers).

Lambda: binding rate constant between chemicals and transcription factors.

Mu: chemicals' unbinding rate from the transcription factors.

Compartments: "y" (yes) for eukaryotic cells, "n" (no) for bacterial cells (*see* **Note 14**).

2.3.7 Other Pools (EU_pools.inp)

Pool kind: to be chosen among: "polymerase", "ribosome", "spliceosome", "dicer", and "risc".

Free molecule concentration: initial concentration of the chosen molecules. This parameter is set to a default value by writing "d" (*see* **Note 15**).

2.3.8 Sum Pools (EU_sum.inp)

Pool kind: to be chosen among: "polymerase", "ribosome", "spliceosome", "dicer", "risc", and "reporter".

Reporter name: string corresponding to the name of the reporter protein (*see* **Note 7**). This entry is ignored if "Pool kind" is different from "reporter".

A *sum pool* is a special type of pool. It serves only to reduce the terminals' number of a device when it is made of several parts. For instance: if a bio-device contains many promoters, they can be all connected to an "RNA polymerase Sum Pool" placed inside the device. In this way, the device itself will have a single terminal connected to the circuit's RNA polymerase pool.

To illustrate how to design a eukaryotic synthetic gene circuit with Parts and Pools, we consider a four-gate biological Boolean network (*see* Fig. 5). The scheme of this circuit is generated by our tool for digital gene circuit automatic design [10, 11] and performs a logic AND operation (*see* **Note 16**) on its two input chemicals: an

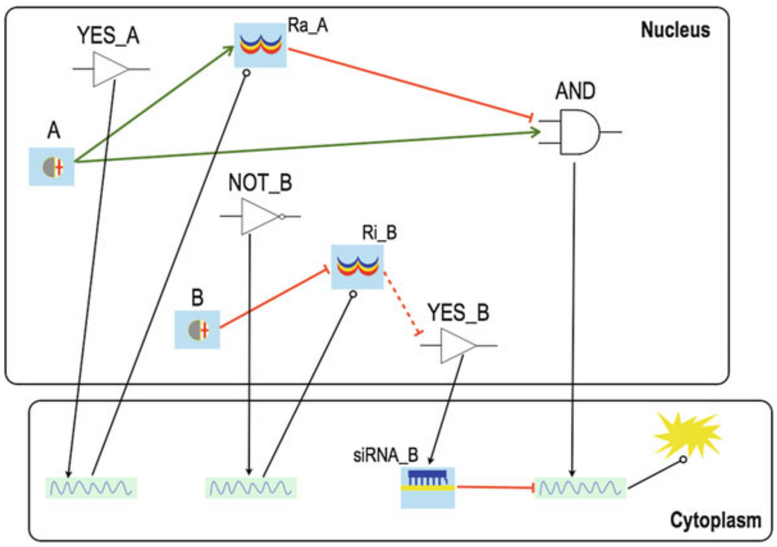


Fig. 5 A gene network carrying out the AND Boolean function. In the circuit scheme we omitted, for the sake of simplicity, the Pools of common signal carriers such as RNA polymerases and ribosomes together with the Pools of dicer enzyme, the spliceosome, and the RISC complex. Since the two input chemicals acts on transcription factors, their Pools are placed into the cell nucleus

2.4 Circuit Design: A Small Gene Network Mimicking the AND Boolean Function via Transcription and Translation Regulation

inducer (*A*) and a corepressor (*B*). Among the Boolean devices we find *YES_A*, which couples the signal *A* to an active repressor (*Ra_A*) and *YES_B*, which associates the other input, *B*, with a small interfering RNA (*siRNA_B*). This operation requires an intermediate step since chemicals do not interact directly with siRNAs. Therefore, *siRNA_B* transcription is controlled by the inactive repressor *Ri_B* (expressed by *NOT_B*) that is activated by corepressor *B*. Circuit design is completed by the *AND* gate where the reporter protein expression (circuit's output) is regulates by *Ra_A*, on the transcription side, and by *siRNA_B*, on the translation side. *A* prevents *Ra_A* from binding the DNA and *B* represses *siRNA_B* transcription after binding and activating *Ri_B*. Therefore, the *AND* gate can produce a reporter protein only in the presence of the two input chemicals, as required.

Eight different Parts (three promoters, three coding regions for proteins, one coding region for siRNAs, and one terminator) are necessary to assemble the four bio-gates. Besides this, this circuit requires two chemicals' Pools (representing the inputs), two repressors' Pools, one reporter Pool (for the output), three mRNA Pools, one siRNA Pool, and the "shared" RNA polymerase, ribosome, spliceosome, dicer enzyme, and RISC complex Pools.

The circuit design can be organized in the following steps: (1) Parts' generation and composition into Boolean gates; (2) Pools' generation and nucleus design; (3) cytoplasm design; (4) nucleus and cytoplasm linking. We suppose that this circuit is hosted in mammalian cells. The choice of parameter values is based on our reference work [2].

2.4.1 YES_A and NOT_B Gate Design

These two gates have an almost identical design since they differ only for the kind of repressor they produce. Each gate is made of three Parts: a constitutive promoter, a coding region, and a terminator.

The constitutive promoter (*p_const*) interacts with the sole RNA polymerase and therefore requires the specification of only few parameters into the "promoter.inp" file (*see* **Note 17**). After running the script "promoter_rules_BNGL.py", four files are generated:

"p_const.mdl" that contains the promoter's model; "p_const_REACTION_LIB.mdl", where an abstract description of the promoter's reactions is present; "p_const.bngl" that encodes rules for a call to BioNetGen; "p_const.net", i.e., the BioNetGen output file with the list of species and reactions present inside the promoter Part.

The coding region for both repressor proteins (named *gene_a* and *gene_b*) is a gene whose mRNA is not regulated by any siRNA or riboswitch (*see* **Note 18**). The script "EU_coding_and_matmRNA_BNGL.py" produces five new files. "Gene_a.mdl" ("gene_b.mdl") describes the coding region Part, whereas "m_mrna_gene_a.mdl" ("m_mrna_gene_b.mdl") encodes the model for the Pool of the mature mRNA corresponding to *gene_a* (*gene_b*). This Pool will be placed in the cell cytoplasm; "m_mrna_gene_a.bngl" ("m_mrna_gene_b.bngl") and "m_mrna_gene_a.net" ("m_mrna_gene_b.net") are the BioNetGet input and output file, respectively. Finally, the reactions associated with the mRNA Pool find an MDL representation into "m_mrna_gene_a_REACTIONS_LIB.mdl" ("m_mrna_gene_b_REACTIONS_LIB.mdl").

The last part necessary to design *YES_A* and *NOT_B* gate is the terminator (*see* **Note 19**). "EU_terminator.py" generates a single MDL file: "term.mdl".

With all the necessary MDL files, the two Boolean gates can be designed in a drag and drop way within the ProMoT Graphical User Interface. First, one has to load, via the *ProMoT Browser*, the file "NEW_LIB.mdl" (*see* **Note 20**). This file contains a call to the ProMoT libraries necessary to handle Parts' and Pools' models based on storage-intras, reactions, and adapter-fluxes. Furthermore, it adds to these libraries some biochemical reactions. After that, it is the turn of the file "p_const_REACTIONS_LIB.mdl" and, finally, one can load the MDL files associated with the three Parts. At this point, a new module called, for instance, *tu_yes_a* and associated with the whole *YES_A* gate (*see* **Note 21**) can be created (*see* **Note 22**). To open the ProMoT canvas (the *Visual Editor*) it is necessary to double click on *tu_yes_a*. The canvas is organized into two panels. On the left one, one can place the Part's icons (*see* **Note 23**), whereas the right one serves to design the circuit scheme. The area for circuit design can be enlarged after selecting *Set Module Size...* from the *Edit* menu of the *Visual Editor*. Parts are then dragged from the left panel to the right one and dropped here. Afterwards, corresponding Parts' terminals have to be connected to each other. There are three types of Parts' terminals that handle fluxes and concentration of signal carriers (or other molecules): *in* and *exc* (full blue circles) and *out* (empty blue circles). Some Parts have also *in* and *out* terminals to send or receive parameter values (green circles). An *out* terminal of a Part is connected to an *in* terminal of another Part or to an *exc* (or, in some cases, an *in*) terminal of a Pool. In general, *exc* terminals are connected to each other. Inside the module *tu_yes_a*, the promoter terminal *out_pol* should be connected to the coding region terminal *in_pol*; the coding region *out_pol* and *in_k_d* to the terminator *in_pol* and *out_k_d*, respectively (*see* **Note 24**). As their name suggests, *out/in_pol* handle the exchange of PoPS, whereas *out/in_k_d* allow the terminator to pass to the coding region the value of the mRNA decay rate. Terminals cannot remain without any link. Therefore, the unconnected terminals of the three Parts become gate terminals and will find a connection into the whole circuit with terminals belonging to other gates or Pools. Promoter *p_const*, for instance, has a "free" *exc_pol* terminal that later will be connected to a terminal (with the same name) on the RNA polymerase Pool. Keeping pressed the CTRL button, one should click on the *p_const* icon on the canvas to open a new menu window. By selecting *Connect Terminals* and, then, *Propagate 'exc_pol'* a gate terminal (called *exc_pol* as well) appears on the left side of the canvas. This operation should be repeated on each Part to convert every unconnected terminal into a gate one (*see* **Note 25**). Finally, the gate design is saved by choosing *Save* from the *Model* menu on the *Visual Editor* (*see* Fig. 6). An MDL file with the gate model is created after highlighting *tu_yes_a* on the *ProMoT Browser* and selecting *Save selected Classes* from the *File* menu. We name the new file as "tu_yes_a.mdl" (*see* **Note 26**). As for *NOT_B*: the only difference is that its coding region is called *gene_b*—and the new module will be named *tu_not_b*.

2.4.2 YES_B Gate Design

YES_B (*tu_yes_b* as a ProMoT module) is made of a promoter regulated by the repressor *Ri_b* (once bound and activated by the corepressor input *B*), a coding region for small interfering RNAs, and a terminator. Differently from the constitutive promoter *p_const* described above, a *YES_B* promoter needs to host operators in order to be regulated by transcription factor proteins. Let us suppose that three operators are present along the *p_ri* sequence and that the repressors bind them without cooperativity (*see* **Note 27**). This promoter structure and regulation determines the

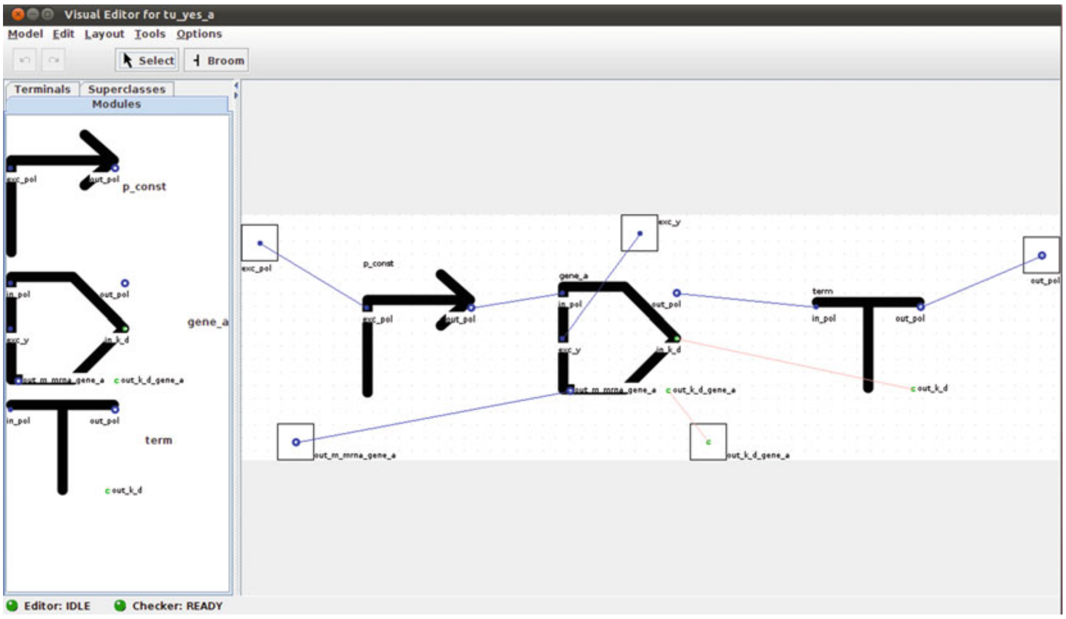


Fig. 6 A transcription unit with ProMoT. This is how the *YES_A* gate appears on the ProMoT *Visual Editor*. Parts' icons are displayed on the *left panel*, whereas the gate is drawn on the canvas. Device terminals are enclosed in *squares* and connected to the corresponding Part terminals

presence of 14 species (5 are adapter-fluxes in the MDL code) and 46 reactions, whereas a constitutive promoter contains only 4 species and 3 reactions.

The script "siRNA.py" generates both the Part ("sirna_b_coding.mdl") and the corresponding cytoplasmic Pool file ("sirna_b_pool.mdl") related to the siRNA coding region (*see* **Note 28**). As for the terminator, one can reuse the "term.mdl" model generated for the above two gates.

The gate design proceeds as illustrated above. One has first to load the promoter's reaction file ("p_ri_REACTIONS_LIB.mdl"), then the promoter file ("p_ri.mdl"), the "sirna_b_coding.mdl" file, and the "term.mdl" file (if not loaded previously). After creating the new module *tu_yes_b*, the three Parts are connected as explained above for *tu_yes_a* and *tu_not_b*. Notice, however, that *p_ri* has three extra terminals (*see* **Note 29**), each of them becoming a gate terminal. Overall, *tu_yes_b* gate has eight terminals.

2.4.3 AND Gate Design

The most complex gate of our illustrative circuit is the one that mimics the AND Boolean behavior. The input *A* (inducer) acts on the gate promoter by inactivating the repressor *Ra_A*; the input *B*, in contrast, regulates the production of the reporter protein by repressing (via *Ri_B*'s activation) the transcription of *siRNA_B* that binds the *AND* gate mRNA. Let us suppose that the *AND* gate promoter, *p_ra*, contains two operators to which *Ra_A* proteins bind cooperatively (for a total of 11 species and 22 reactions—*see* **Note 30**). *AND* gate's coding region, *gene_and*, is transcribed into an mRNA where translation is regulated via RNA interference. Symmetrically to *Ra_A*, we assign two binding sites to *siRNA_B* as well (*see* **Note 31**). This implies the presence of 22 reactions and 12 species into *gene_and* mRNA Pool (against the 9 reactions and 7 species inside *gene_a* and *gene_b* mRNA Pools). *AND* gate design requires the procedure followed for the previous three gates. This transcription unit (*tu_and* module) has overall nine terminals. It exchanges both active and inactive repressors with *Ra_A* Pool via the terminals *exc_ra1* and *exc_ri1*, respectively. Indeed, *Ra_A* enters the promoter and can be there inactivated, by an inducer, when it is bound to the DNA. Inactivated repressors go back to their Pool where they can dissociate from the inducers and get active again (this reaction is excluded from the promoter model).

2.4.4 Chemicals', Transcription Factors', and Reporter's Pools

For the sake of simplicity, we assume that chemicals act on their repressor targets only inside the nucleus. Therefore, we place both *A* and *B* input signal Pools in this cellular compartment (*see* **Note 32**). Pools' models are created by the script "sigpool_g.pl" (files: "a_pool.mdl" and "b_pool.mdl"). Only one terminal (*exc_sig*) is present on each Pool (*see* **Note 33**).

We assume that both circuit's repressors, *Ra_A* and *Ri_B*, dimerize. Their Pools (files: "ra_a_pool.mdl" and "ri_b_pool.ml") are generated by the script "tfpool_g.pl" (*see* **Note 34**). Each Pool has four terminals: *exc_tf_a* and *exc_tf_i*, which are connected to the promoter where the repressor acts; *exc_sg*, which puts the repressor Pool in communication with the corresponding input signal Pool; *in_ra_a* (or *in_ri_b*), which is connected to the mRNA Pool where the transcription factor is synthesized.

Finally, the reporter protein Pool file ("rep_pool.mdl") is produced by the script "reporter_pool.py" whose input file is very similar to the one for the transcription factors (*see* **Note 35**). This Pool contains a unique terminal, *in_rep*, connected to the mRNA Pool where the fluorescent protein translation is carried out. Since fluorescence is, in general, not localized in the nucleus, we place the reporter protein Pool in the cell cytoplasm.

2.4.5 Other Pools

MDL files for Pools of molecules such as RNA polymerases ("pol_-pool.mdl"), ribosomes ("rib_pool.mdl"), the splicesome ("y_pool.mdl"), the dicer enzyme ("dicer_pool.mdl"), and the RISC complex ("risc_pool.mdl") are written by a single script called "EU_pools.py". The input file "EU_pools.inp" requires the specification of the molecule type and initial concentration only. Each Pool has a single terminal of *exc* type (*see* **Note 33**). RNA polymerase, spliceosome and dicer enzyme Pool are in the nucleus, whereas ribosome and RISC complex Pool find place in the cytoplasm.

Now we have all the circuit components: gene Parts assembled into Boolean gates and Pools. We can move on and design, separately, the nucleus and the cytoplasm. Finally, we will connect them to close our synthetic gene circuit.

2.4.6 Compartments' Design

To design the portion of the circuit located in the cell nucleus, one has to load, in order, the following MDL files on the *ProMoT Browser*: the REACTIONS_LIB files associated with the three promoters; the eight MDL files containing the model for the twelve Parts (*p_const* is used twice, *term* four times); the four gates' files; the seven MDL files encoding for the nuclear Pools (*A*, *B*, *Ra_A*, *Ri_B*, RNA polymerase, dicer enzyme, and spliceosome—*see* **Note 36**). First, it is necessary to create a module for the nucleus. We will call it *nucleus_and*. By double clicking on *nucleus_and* one opens, as usual, the *Visual Editor*. After modifying the "module size", one has to drag to and drop on the canvas the four Boolean gates and the seven Pools that make the circuit's nucleus. Then, they have to be connected to each other.

Gates are not connected to one another directly. They are linked to Pools only: RNA polymerase Pool, since this signal carrier is the responsible for gate transcription; spliceosome or dicer enzyme Pools, depending on the RNA sequence transcribed from the gate; mature mRNA and siRNA Pools that contain their final RNA products and are placed into the cytoplasm; transcription factor and chemical Pools whose content exerts transcriptional control on the promoters' gates.

Each gate is connected to the RNA polymerase Pool by linking both gate's terminals *exc_pol* and *out_pol* to the only terminal present on *pol_pool* (it is called *exc_pol* as well—*see* **Note 37**).

tu_yes_a, *tu_not_b*, and *tu_and* are connected to the spliceosome Pool (*see* **Note 38**). This connection requires linking gates' and spliceosome Pool's terminals called *exc_y*. *tu_yes_b* produces siRNAs, therefore it has to be connected to the dicer enzyme Pool. The link goes from the gate terminal *exc_d* to *exc_dicer* on the Pool. *tu_yes_b* promoter is regulated by *Ri_B* upon activation by the corepressor *B*. Therefore, *tu_yes_b* terminal *exc_ra1* is linked to *exc_tf_a* on *Ri_B* Pool, whereas the gate terminal *exc_c1r* is connected to *exc_sig* on signal *B* Pool (*see* **Note 39**). Furthermore, *B* and *Ri_B* Pools should be put in communication by linking the terminals *exc_sig* (on *b_pool*) and *exc_sg* (on *ri_b_pool*). Gate *AND* is controlled by *Ra_A* repressor. The inducer *A* can enter the gate too and inactivate *Ra_A* molecules bound to the *p_ra* promoter. Hence, *tu_and* and *ra_a_pool* have a double connection to handle the flux of both active and inactive repressors. This requires a link between the terminals *exc_ra1* (*AND* gate) and *exc_tf_a* (*Ra_A* Pool) as well as a wire between *exc_ri1* (*AND* gate) and *exc_tf_i* (*Ra_A* Pool). *A* Pool has to be connected to both *Ra_A* Pool (link between *exc_sig* and *exc_sg* as seen above) and the *AND* gate (wire from *exc_sig* again and *exc_i1r* on *tu_and*).

All the remaining gates' and Pools' unconnected terminals become compartment terminals and will find their counterpart on the cytoplasm module. Both *tu_yes_a* and *tu_not_b* provide two nuclear terminals: *out_m_mrna_gene_a* (*out_m_mrna_gene_b*) and *out_k_d_gene_a* (*out_k_d_gene_b*). The former handles the fluxes of mature mRNA to the cytoplasm, the latter permits the communication of the mRNA decay rate, determined by the gate terminator, to the cytoplasmic mature mRNA Pool. Together with two analogous terminals, the *AND* gate shows a third unconnected one: *out_pol_lk_p_ra*. This terminal will be connected to the reporter protein's mature mRNA Pool passing to it a flux of PoPS due to the promoter *p_ra* leakage (*see* **Note 40**). *tu_yes_b* has an unconnected terminal due to promoter leakage as well (*out_pol_lk_-p_ri*), together with other two that handle information about *siR-NA_B* (*out_sirna_b* and *out_k_d_sirna_b*). They will be connected to the *siRNA_B* Pool in the cytoplasm. Each repressor Pool has a free terminal too: *in_ra_a* and *in_ri_b*. They gets a FaPS flux (Factor Per Second) from their corresponding mature mRNA Pools in the cytoplasm.

After propagating all the unconnected gates' and Pools' terminals, the nucleus design is complete and *nucleus_and* module can be saved into a separate MDL file.

Cytoplasm design requires the MDL files of seven Pools: three mature mRNA Pools (associated with *gene_a*, *gene_b*, and *gen-e_and* transcription), the *siRNA_B* Pool, the RISC complex one, the reporter protein one, and the ribosomes one. Before the mRNAs and the *siRNA_B* Pools, one has to load the corresponding REACTIONS_LIB files. After instancing a new module on the *ProMoT Browser* (*cytoplasm_and*), the seven Pools can be displayed, as usual, on the *Visual Editor*. The ribosomes Pool is connected to each mature mRNA Pool via a wire between the respective *exc_rib* terminals; the RISC complex Pool is joined to the *siRNA_B* via a link between the two *exc_risc* terminals; *siRNA_B* Pool and the Pool of mature mRNA encoding for the reporter protein (*m_mrna_gene_and*) are bridged by a wire between their terminals: *exc_sirna_b* and *exc_s1*, respectively (*see* **Note 41**); finally, *m_mrna_gene_and* needs a connection to *rep_pool* to send it a flux of fluorescent proteins. This is achieved with a link between *out_rep* (on the mRNA Pool) and *in_rep* (on the reporter Pool). All the other Pools' terminals are propagated to cytoplasmic terminals. *m_mrna_gene_a* and *m_mrna_gene_b* have three unconnected terminals: one receives a flux of mature mRNA from the corresponding gate in nucleus (*in_m_mrna_gene_a/b*); another receives—from the gate's coding regions—the mRNA decay rate (*in_k_d_gene_a/b*); the last one, finally, sends a FaPS flux to the associated repressor Pool in the nucleus (*out_ra_a/out_ri_b*). Three analogous terminals are also present on *m_mrna_gene_and*, together with a fourth unconnected one, *in_pol_lk_p_ra*, that get a PoPS leakage flux from the *AND* gate (*see* **Note 40**). Finally, *siRNA_B* Pool has three free terminals as well: *in_sirna_b*, *in_k_d_sirna_b*, and *in_pol_lk_p_ri*. Also in this case, each terminal gets information from the nucleus: the first one, a flux of siRNAs; the second one, the siRNA decay rate; the last one, a PoPS leakage flux. With the propagation of all these unconnected terminals to compartmental terminals, the cytoplasm design is over and the new module can be saved into a new file named “cytoplasm_and.mdl”.

2.4.7 Closing the Circuit: Compartment Connection

Both *nucleus_and* and *cytoplasm_and* have been designed as ProMoT modules. They have to be converted into another kind of object, called *compartments*, in order to specify their volumes. This task is carried out by the python script “compartment_parser.py”. It takes three inputs: the name of the MDL file associated with the module to be converted, a value for the compartment volume, and the type of compartment. They are specified into the input file “Compartment.inp” (*see* **Note 42**). By running “compartment_-parser.py” on *nucleus_and* and *cytoplasm_and*, some new MDL files are written (*see* **Note 43**). The most significant ones are named “ALL_nucleus_and.mdl” and “ALL_cytoplasm_and.mdl”. Their content is a complete description of the compartments since they embrace all the compartment’s components: devices with genetic Parts, Pools, and the corresponding biochemical reactions.

The final step to close our synthetic circuit is the connection of the two compartments into a new ProMoT object that belongs to the *sbml-model* class. This can be achieved via the script “link_compartment.py”. This script reads its three input values from the file “Link_compartment.inp” (*see* **Note 44**). A unique file, called “ALL_cell_and.mdl”, is created (*see* **Note 45**): it contains an instance of the new sbml-model object *cell_and* together with a description of all its components (taken from “ALL_nucleus_and.mdl” and “ALL_cytoplasm_and.mdl”). “ALL_cell_and.mdl“ can be loaded on the *ProMoT Browser* directly, i.e., without any other MDL file having been previously loaded. *Cell_and* is under the *sbml-model* folder (which is, in turn, under *module*). It can be visualized on the *Visual Editor* just by double clicking on it, as usual. Here, it appears as two connected compartments. By double clicking on each of them, one can see their components (gates and Pools). By double clicking on nuclear Boolean gates, one can have a look at their Parts. By further double clicking on Parts and Pool, one can have a visual representation of the species, reactions, and fluxes involved in them. Finally, by double clicking on storage-intra (species) and reactions, one can change concentrations or parameter values (*see* **Note 46**).

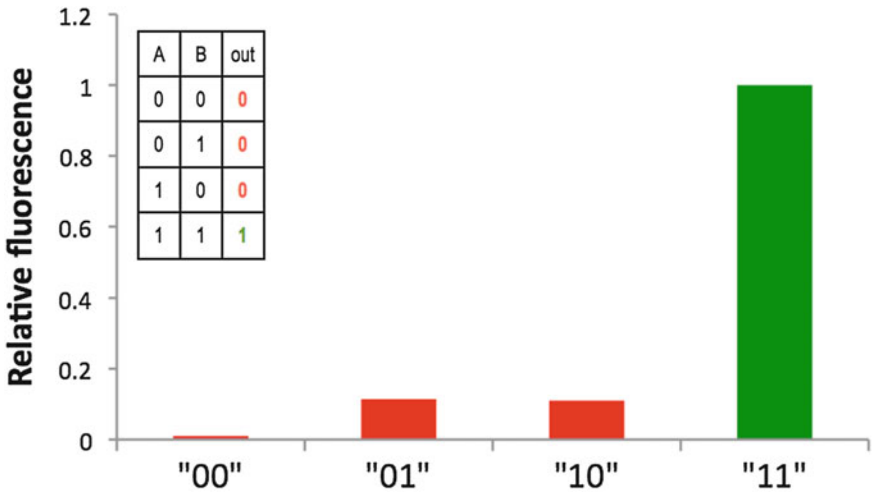


Fig. 7 AND gate simulations. Deterministic simulations of our circuit that mimics AND Boolean behavior are performed with COPASI [18]. First, the system reaches a steady state in absence of chemicals (48-h simulation). Then, signals' concentrations are changed (switched to 0.01 M) according to the four truth table entries (on the *x* axis). The circuit reaches a second steady state after 96 h of simulation. Reporter protein concentration is taken as a circuit output and the data in figure are normalized to the highest output (logic 1) occurred when both *A* and *B* input chemicals are present ("11" on the circuit truth table—*see* the *insert*)

Since the circuit design is concluded, one can now export *cell_and* to a format suitable for simulations such as Matlab (Mathworks, Nantucket/MA) and SBML [12] (*see* **Note 47**). In Fig. 7, results from deterministic simulations on our illustrative digital circuit are shown.

3 Notes

1. Chemicals' kinds refer to the action of these small molecules directly on transcription rather than on the transcription factor proteins they bind. Inducers allow transcription by either inhibiting a repressor or turning on an activator; corepressors switch off transcription by either activating a repressor or turning off an activator.
2. "Free" refers to the initial promoter state where all the operators are free and RNA polymerase is not present on its binding site.
3. To properly represent cooperativity, the first value of alpha ("alpha_strong") should be greater than the second ("alpha_weak"). Moreover, "alpha_weak" should be lower than the third value ("alpha_cooperativity"). As for beta: "beta_strong" should be lower than "beta_weak" that, in turn, should be

greater than “beta_cooperativity”. These values are checked by the software and a warning message is printed if these rules are not respected. In this way, *partial cooperativity* can also be simulated as in the Repressilator model [13].

4. Effectors are chemicals that bind riboswitches’ structures called aptamers and change their structure so that ribosome binding can be either obstructed or facilitated.
5. Two effectors cannot have the same integer label. This label has nothing to do with the riboswitch the effectors bind; it serves only to unequivocally identify each effector.
6. In principle, this rate should be the same specified into the corresponding protein pool.
7. This entry is necessary for the script “link_compartment.py” that generates the links between nucleus and cytoplasm (*see* Subheading 2.4.7).
8. Also this entry is required to run the script “link_compartment.py” (*see* **Note 7**).
9. The bacterial RBS has an input file (“rbs.inp”) similar to the one for the eukaryotic coding part. However, bacterial small RNAs can both activate and inhibit translation and, therefore, are of two kinds: locks and keys, following the notation in [14].
10. In the siRNA model, the dicer enzyme plays the same role as the spliceosome in the mRNA model.
11. In our bacterial Part models, terminators do not determine the mRNA decay rate. Moreover, bacterial terminators require a further parameter, the readthrough rate (*eta*), since RNA polymerase might not detach the DNA at the terminator and pass directly to the adjacent transcription unit. Notice, however, that if you use promoters and RBSs coming from the rule-based modeling approach (presented in this chapter) to build bacterial circuits, you have to set the value of *eta* to zero in every terminator (and, as a consequence, to ‘n’—no—every readthrough flux).
12. Inactive transcription factors have to be bound by chemicals in order to act on the DNA.
13. One can use the same decay rate for both free transcription factors (inside their nuclear Pool and into the mature mRNA Pool where they are produced) and transcription factors that are bound to the DNA into a promoter.
14. The reporter protein pool input file (**reporter_pool.inp**) has the same entries as the transcription factor pool input file except for “signals” and the parameters “lambda” and “mu”. Since reporter proteins represent, normally, the circuit output, they are not supposed to interact with any chemicals—used as circuit inputs.

15. Default values are: RNA polymerase, 1.3e−7; ribosomes, 5.0e−8; spliceosome, 1.3e−7; dicer enzyme, 1.3e−7; RISC complex, 5.0e−8. All these values are in M. Throughout the Notes units are never specified. Rates are always expressed in $s^{(-1)}$, whereas rate constants are generally in $M^{(-1)}$ $s^{(-1)}$.
16. A two-input AND gate gives 1 when both inputs are equal to 1, otherwise the gate's output is 0. In synthetic gene digital circuits, 0 and 1 correspond to low (or even null) and high molecules concentrations, respectively.
17. Promoter name: p_const—repressors' number: 0—activators' number: 0—p_free: 4.4e−12—k1: 1e5—k_1: 1—k2: 0.5—compartment: y. Moreover, the local directory and the BioNetGen directory have to be set properly. All the other parameters are ignored.
18. In the input file "EU_coding.inp" one has to set: gene name: gene_a (gene_b)—riboswitches' number: 0—siRNAs' number: 0—polymerase leakage: n—gene length: 1,500—k1r: 1e6—k_1r: 0.01—k2r: 0.02—k1y: 1.5e3—k_1y: 0.0017—k2y: 0.033—km: 5.5e−4—zeta_r: 0.5—k_tr: 0.0083—k_dp: 2.8e−5—product name: Ra_A (Ri_B). Once again, both the local and BioNetGen directories have to be set properly.
19. The "EU_terminator.inp" entry can be set as follows: terminator name: term—mRNA k_d: 3.8e−5—zeta: 31.25.
20. Choose *Open* from the *File* menu and then *Model*.... Alternatively, one can use the shortcut CTRL + O.
21. It is recommendable to use transcription units as basic devices for circuit design. In order to use our script "compartment_parser.py" to generate the MDL file for the nucleus, the name of every transcription unit module should start with the prefix "tu_".
22. Open the folder *structural-modeling-entity*, click on *module* and select *Add subclass*... from the *Edit* menu.
23. Click on the Part's name on the *ProMoT Browser*: the Part's icon will appear on the screen. Drag it to and drop it on the left panel of the *Visual Editor*. Keeping the CTRL button pressed, click on the icon and select *Default Class* from the menu window that pops up. In this way, the icon will be present on the *Visual Editor* when you start the design of a new transcription unit.
24. Click on a terminal, a white cross appears. Drag it to the corresponding terminal on a different icon and release the mouse button: a wire between the two terminals is created.
25. When this is achieved, no Part appears enclosed in a red frame.
26. To associate *tu_yes_a* with the YES gate icon, open the "tu_yes_a.mdl" file with a text editor and add a new line *:icon "yes.png"* just under *:super-classes ("module")*. After saving the change, reload the file into ProMoT.

27. In the "promoter.inp" file one should write: promoter name: p_ri—repressors' number: 1—operators per repressor: 3—repressors: Ri1—chemicals binding repressors: C1—activator number: 0—repressor cooperativity: no—p_free: 4.4e-12—alpha-repressors: 1e9—beta-repressors: 10—k_d-repressors: 2.8e−5—gamma-repressors: 1e6—k1: 1e5—k_1: 1—k2: 0.5—k2_lk: 5.0e−5—compartment: y. Both local and BioNetGen directory have to be set properly.
28. The "siRNA.inp" input file entries should be set as follows: siRNA name: sirna_b—polymerase leakage: y—siRNA length: 20—k1d: 1.5e3—k_1d: 0.0017—k2d: 0.033—km: 5.5e−4—k1risc: 3.0e7—k_1risc: 0.017—promoter name (link): p_ri.
29. One terminal (*out_pol_lk_p_ri*) is due to the leakage of RNA polymerase. As a gate terminal, it will be connected to the *siRNA_B* Pool in the cytoplasm since RNA polymerase leakage has the effect to increase the amount of *siRNA_B* directly. Another terminal, *exc_Ra1*, will be connected to the *Ri_B* Pool. Here, repressors *Ri_B* get activated by corepressors *B* and, only then, can leave the Pool and interact with the DNA. Do not confuse *Ri_B* Pool with *Ra_A* one. As explained above, inside a promoter Part transcription factors are named according to their type (*Ra*, *Ri*) to which a label (an integer) is added. These names have nothing to do with the names the same transcription factors have into their Pools. The third extra terminal, *exc_c1r*, will be connected to the signal *B* Pool.
30. *p_ra* requires the following "promoter.inp" configuration: promoter name: p_ra—repressors' number: 1—operators per repressor: 2—repressors: Ra1—chemicals binding repressors: I1—activator number: 0—repressor cooperativity: 1—p_free: 4.4e-12—alpha-repressors: (1e9;1e7;1e9)—beta-repressors: (9;224;9)—k_d-repressors: 2.8e−5—gamma-repressors: 1e6—k1: 1e5—k_1: 1—k2: 0.5—k2_lk: 5.0e−5—compartment: y. Finally, both local and BioNetGen directory have to be set properly.
31. The coding region input file entries should be set as follows: gene name: gene_and—riboswitches number: 0—siRNA number: 1—k_fd: 2e−3—sites per siRNA: 2—polymerase leakage: y—gene length: 1,500—k1r: 1e6—k_1r: 0.01—k2r: 0.02—theta siRNA: 1e7—csi siRNA: 0.01—k2r_lk: 1.0 e−5—k1y: 1.5e3—k_1y: 0.0017—k2y: 0.033—km: 5.5 e-4—zeta_r: 0.5—k_tr: 0.0083—k_dp: 2.8e−5—product name: rep—promoter name (link): p_ra. Moreover, the specification of both local and BioNetGen directories is needed.
32. The file "sigpool.inp" should be filled in the following way: signal name: a (b)—connected to an input source: n—sig_free: 0 (0.01)—k_s: 0—k_d: 0—compartment: y. In the circuit

simulations, we let the system reach the steady state in absence of chemicals and then chemical's concentration is changed to 0.01 M. For simplicity, we assume that chemicals are not degraded.

33. There is no limit to the number of terminals to which a single terminal can be connected. For instance, an *exc* terminal belonging to a signals' Pool is connected to as many terminals as there are Parts and Pools that host interactions with these chemicals.

34. Into the file "tfpool.inp" one should set: tf_name: Ra_a (Ri_b)—type: d—tf state: a (i)—signals: y—tf free: 0—k_d: 2.8e−5—delta: 1e9—epsilon: 10—lambda: 1e6—mu: 1e−3—compartment: y.

35. Here we assume that reporter proteins do not form dimers. Hence, the "reporter_pool.inp" file should be filled in as follows: reporter name: rep—type: m—initial concentration: 0—k_d: 2.8e−5—compartment: y. Values for delta and epsilon are ignored.

36. If one wants to build a circuit in different steps, in order to avoid loading each time too many MDL files, one can save his under-construction circuit into a single MDL file containing all its components. To do that, one has first to highlight, on the *ProMot Browser*, all the Parts and Pools he wants to save and then choose *Save selected classes* from the *File* menu. We suggest saving each Part and Pool into a separate MDL file in any case, this might be helpful for future circuit modifications.

37. Multiple-terminal connections are handled by ProMoT by creating a new entity, called node, that is represented by an empty, blue circle.

38. All the steps that lead to mRNA maturation take place inside the coding region Part (*gene_a*, *gene_b*, and *gene_and*) of each gate.

39. Similar to what is pointed out in **Note 29**, there is no correspondence between the repressor and chemical name in their Pools and in the Part on which they act. Chemical *B* is called, for instance, *c1r* into *tu_yes_b* since it acts on repressor (*r*) number *1*.

40. This PoPS flux is fictitious: it serves only to increase the amount of mature mRNA into its corresponding Pool and does not represent a real current of RNA polymerases.

41. s1 means siRNA number 1.

42. By writing "dn" or "dc" as *volume* into "Compartment.inp" a default value will be assigned to the nuclear or cytoplasmic volume. The former is set to 3.74e-13l [15], the latter to 9.7e-13l [16]. The input file should be specified with the

MDL extension ("nucleus_and.mdl" and "cytoplasm_and.mdl" in our case). As for the compartment type: 'n' (nucleus) and 'c' (cytoplasm) are the accepted characters.

43. Every transcription unit is rewritten into a new MDL file where each Part contains the required specification of the nuclear volume. Therefore, we have four new MDL files: "volume_tu_yes_a.mdl", "volume_tu_yes_b.mdl", "volume_tu_not_b.mdl", and "volume_tu_and.mdl". Moreover, two other new MDL files contain the description of the compartment class objects. They are called: "compartment_nucleus_and.mdl" and "compartment_cytoplasm_and.mdl"

44. "Link_compartment.inp" entries require to be set as follows: Nucleus name: nucleus_and—Cytoplasm name: cytoplasm_and—SBML model name: cell_and.

45. Two other new MDL are written into the working directory. One is "cell.mdl", where the whole cell is a usual ProMoT module. This file represents the input for another python script, named "SBML_parser.py", which is invoked by "link_compartment.py" directly. "SBML_parser.py" converts *cell_and* module into a ProMoT sbml-model and produces the file "sbml_model_cell_and.mdl".

46. For instance, in order to change the transcription initiation rate of the promoter *p_ra* into the *AND* gate one should double click, in the order, on *nucleus_and*, *tu_and*, and *p_ra*. To visualize *p_ra* content better, it is necessary to change the Part layout (click on *Layout* and choose, for instance, *Radial Layout*). The transcription initiation rate is the parameter that corresponds to the reaction named *k2_13* (every kinetic parameter is followed by an integer to avoid ambiguities in the reactions' names). By double clicking on *k2_13* reaction's icon, a new window is opened. It is named *Details for 'k2_13'*. The panel *Slot Variables* is organized into folders. Folder *k2_13* contains three subfolders (one for each terminal: *a*,*b*, and *c*) and three variables. *K1* variable represents the kinetic parameter associated with the reaction and its current value should be 0.5 s^{-1} (*see* **Note 30**). After double clicking on it, one can type a new value. To make this modification permanent, one has to click on *Save And Close* on the bottom of this window.

47. Highlight *cell_and* on the *ProMoT Browser*, then choose *Export* from the *File* menu and finally select either *Output to Matlab...* or *Export to SBML....* Before the export operation, it is advisable to check that no error is present in the circuit scheme. Open *cell_and* in the *Visual Editor* and choose *Check For Consistency* from the *Tools* menu. If everything is correct, you will have in the circuit's model as many variables as the equations' number.

Acknowledgment

This work was partially supported by the EU FP7 project ST-FLOW (contract 289326).

References

1. Endy D (2005) Foundations for engineering biology. Nature 438:449–453
2. Marchisio MA, Colaiacovo M, Whitehead E, Stelling J (2013) Modular, rule-based modeling for the design of eukaryotic synthetic gene circuits. BMC Syst Biol 7:42
3. Mirschel S, Steinmetz K, Rempel M, Ginkel M, Gilles ED (2009) PROMOT: modular modeling for systems biology. Bioinformatics 25:687–689
4. Faeder JR, Blinov ML, Hlavacek WS (2009) Rule-based modeling of biochemical systems with BioNetGen. Methods Mol Biol 500:113–167
5. Ginkel M, Kremling A, Nutsch T, Rehner R, Gilles ED (2003) Modular modeling of cellular systems with ProMoT/Diva. Bioinformatics 19:1169–1176
6. Marchisio MA, Stelling J (2008) Computational design of synthetic gene circuits with composable parts. Bioinformatics 24:1903–1910
7. Marchisio MA, Stelling J (2009) Synthetic gene network computational design, In *Proc.* IEEE Int Symp Circuits Syst ISCAS 2009:309–312
8. Saez-Rodriguez J, Kremling A, Gilles ED (2005) Dissecting the puzzle of life: modularization of signal transduction networks. Comput Chem Eng 29:619–629
9. Lewin B (2000) Genes VII. Oxford University Press, New York
10. Marchisio MA, Stelling J (2013) Simplified computational design of synthetic gene digital circuits. In: Kulkarni V, Raman K, Stan G (eds) System theoretic and computational perspectives in systems and synthetic biology. Springer, New York
11. Marchisio MA, Stelling J (2011) Automatic design of digital synthetic gene circuits. PLoS Comput Biol 7:e1001083
12. Hucka M et al (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics 19:524–531
13. Elowitz MB, Leibler S (2000) A synthetic oscillatory network of transcriptional regulators. Nature 403:335–338
14. Massé E, Escorcia FE, Gottesman S (2003) Coupled degradation of a small regulatory RNA and its mRNA targets in Escherichia coli. Genes Dev 17:2374–2383
15. Maul GG, Deaven L (1977) Quantitative determination of nuclear pore complexes in cycling cells with differing DNA content. J Cell Biol 73:748–760
16. Fujioka A, Terai K, Itoh RE, Aoki K, Nakamura T, Kuroda S, Nishida E, Matsuda M (2006) Dynamics of the Ras/ERK MAPK cascade as monitored by fluorescent probes. J Biol Chem 281:8917–8926
17. Galdzicki M, Rodriguez C, Chandran D, Sauro HM, Gennari JH (2011) Standard biological parts knowledgebase. PLoS One 6:e17005
18. 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 22:3067–3074

Chapter 8

Computationally Guided Design of Robust Gene Circuits

Najaf A. Shah and Casim A. Sarkar

Abstract

The inability to rationally design and construct circuits that robustly enable complex behaviors is perhaps the most fundamental challenge in synthetic biology. While systems modeling can aid this process and help reduce the space of design strategies, the unavailability and dynamic variability of kinetic parameters limits the utility of such models. Here, we present a general approach that employs an exhaustive enumeration of network architectures to suggest topologies that robustly enable a desired behavior.

Key words Synthetic circuit design, Systems modeling, Network topology, Topology search

1 Introduction

The difficulty in engineering circuits that are both complex enough to yield desired dynamic behaviors and sufficiently reliable to be deployed in medical and industrial applications is perhaps the most significant hindrance to progress in synthetic biology. Despite progress in the identification, creation, and characterization of a large number of new parts as well as advances in genetic manipulation techniques over the past decade, the circuit design–implement–test cycle remains largely unchanged in that most functioning synthetic circuits are constructed not via rational and streamlined application of engineering principles, but through a laborious process that involves assembling ad hoc combinations of parts and iterating until the desired behavior is obtained [1, 2].

Systems modeling and simulation can greatly aid the circuit construction process by elucidating specific architectures for robustly implementing the desired functionality, especially in instances involving nonlinear or emergent behavior. However, the canonical approach to modeling rarely leads to designs that yield the desired functionality without extensive optimization. This lack of predictivity is primarily due to the fact that models assume values for kinetic parameters (e.g., the dissociation rate constant for a transcription factor or the turnover number for a kinase) which

Mario Andrea Marchisio (ed.), *Computational Methods in Synthetic Biology*, Methods in Molecular Biology, vol. 1244,
DOI 10.1007/978-1-4939-1878-2_8, © Springer Science+Business Media New York 2015

are often incorrect; more importantly, these effective parameter values are modulated by intrinsic and extrinsic perturbations (e.g., temperature and growth rate), and are hence under constant flux.

Here, we describe a simple and general approach to designing synthetic circuits that yields network architectures that are parametrically robust in that the architectures exhibit the desired behavior over a large subspace of parameter values. The basic method involves exhaustively enumerating all three-component network topologies, translating each topology into a systems model, simulating each model under a large number of random, biologically reasonable parameter sets, and finally scoring the performance of each topology in yielding the desired behavior. This approach was introduced by Ma et al. [3] to study networks enabling adaptation, and has since been extended and applied to study networks enabling switch-like behavior [4].

2 Topology Search Procedure

Although this method can be adapted to study network architectures robustly enabling any well-defined behavior, we describe here the various steps in the context of a specific behavior, ultrasensitivity to an external stimulus. This systems-level property enables cells to establish a stimulus threshold such that a small increase in stimulus concentration at this threshold leads to a dramatic increase in the output of the system. Mathematically, an ultrasensitive response in a stimulus-activated system is one in which the output (e.g., the phosphorylation of a signaling protein) shifts from 10 % to 90 % of the maximum response, with less than an 81-fold increase in stimulus concentration [5].

2.1 Exhaustively Enumerate Minimal Networks

To keep the analysis tractable, we recommend studying minimal networks of three components. A three-component network can be sufficiently complex to yield a variety of dynamic behaviors, and the entire space of three-component networks is amenable to simulation and further analysis. Due to the combinatorial nature of the topology space, increasing the number of components to four or five would lead to a dramatically enlarged set of networks that may be impractical to exhaustively simulate (and experimentally implement); moreover, the incremental benefit is likely to be insignificant for many behaviors.

1. Start with three abstract network components, labeled A, B, and C. A is considered the input component, C is considered the output component, and B is an additional regulatory component.
2. Allow each component to interact with every other component and itself. In a network of three components, there can be a

maximum of nine unique interactions such that each interaction involves an actor component (the node from which the connection originates) and a target component (the node at which the connection terminates). Furthermore, each interaction can be positive, negative, or non-existent. The biological meaning of these interactions depends on the specific type of actor component involved (*see* **step 5** below). For instance, an enzyme B could activate C, inactivate C, or have no effect on C. There are nine possible interaction connections, and three different types of interactions, yielding a universe of $3^9 = 19{,}683$ unique network topologies.

This step can be implemented as follows. Allocate a matrix, M, with nine elements, one for each of the possible interactions:

$$\begin{pmatrix} M_{A\to A} & M_{A\to B} & M_{A\to C} \\ M_{B\to A} & M_{B\to B} & M_{B\to C} \\ M_{C\to A} & M_{C\to B} & M_{C\to C} \end{pmatrix}$$

Each element in M can take on one of three values: 0 (indicating the absence of that particular interaction), 1 (indicating a positive interaction), and 2 (indicating a negative interaction). Hence, enumerating all such possible matrices yields all possible network topologies.

3. As an optional step, remove network topologies in which there are no direct or indirect interaction routes by which the input component A can impact the output component C, since in these cases the output component is independent of the input component.

   Specifically, filter out all matrices from the previous step in which $M_{A\to B}$ and $M_{A\to C}$ are both 0 (i.e., topologies lacking an interaction in which A acts on B or C)

4. Each component can exist in one of two states: an active form, which can exert an action, and an inactive form, which cannot. Both active and inactive components can exist as monomers or be bound to another component. For clarity, here we denote the active form of a component by appending an asterisk; for instance, active B is written as B^*.

5. Identify each component as being either an enzyme or transcription factor. From a modeling view, most components of synthetic circuits can be identified as either of the following.

   (a) Enzyme: a component which binds to a target component and catalyzes its interconversion. For example, if B is an enzyme and there is a negative link from B to C, B^* can inactivate C^* by supporting its conversion into C. Alternatively, if there is a positive link from B to C, B^* can activate C by supporting its conversion into C^*. Thus, an enzyme does not change the total concentration of its target; rather, it alters the fraction of this concentration that is in the active state.

(b) Transcription factor: a component which synthesizes the inactive form of its target. For instance, if B is a transcription factor and there is a positive link from B to C, B^* can upregulate the synthesis of the inactive form of C. Alternatively, if there is a negative link from B to C, B^* downregulates the synthesis of C. Thus, a transcription factor modulates the total concentration of its target; it does not directly act on target molecules to change their activity state.

Allocate a three-element vector, I, specifying the identities of the components. For example, the following vector specifies that A and B are enzymes, and C is a transcription factor:

$$I = [\mathrm{E}, \mathrm{E}, \mathrm{T}]$$

As is the case biologically, the identity of a component within a network has a very significant impact on the spectrum of behaviors observed over the entire space of networks. For instance, in our study of ultrasensitivity, we found that networks of two enzymes and one transcription factor (with enzymes A, B and transcription factor C) were as a group significantly more robust than networks of three transcription factors (with transcription factors A, B, C). Through post-hoc analysis, we discovered that this enhanced robustness was due in large part to zero-order ultrasensitivity, which is an enzyme-specific phenomenon. Given that the optimal combination of component identities may vary for different applications, we recommend repeating the exhaustive topology search with different combinations of identities of the three components.

2.2 Build a Systems Model for Each Network Topology

The previous step yields a large set of topologies, where each of the three components (A, B, and C) is identified as an enzyme or a transcription factor. In the current step, convert network topologies into a set of ordinary differential equations (ODEs) by applying the following general procedure for each network topology.

1. A network has three components, each of which can exist in an active and an inactive form, yielding six distinct species.
2. Network components are subject to basal synthesis, degradation, activation, and inactivation processes. Hence, irrespective of the topology, each network model consists of a system of ODEs with at least the following terms:

$$
\begin{aligned}
\frac{\mathrm{d}A}{\mathrm{d}t} &= b_{\mathrm{syn},A} - k_{\mathrm{deg},A}A - k_{P,A}P\frac{A}{A+K_{P,A}} + k_{Q,A}Q\frac{A^*}{A^*+K_{Q,A}} + \cdots \\
\frac{\mathrm{d}A^*}{\mathrm{d}t} &= -k_{\mathrm{deg},A}A^* + k_{P,A}P\frac{A}{A+K_{P,A}} - k_{Q,A}Q\frac{A^*}{A^*+K_{Q,A}} + \cdots \\
\frac{\mathrm{d}B}{\mathrm{d}t} &= b_{\mathrm{syn},B} - k_{\mathrm{deg},B}B - k_{P,B}P\frac{B}{B+K_{P,B}} + k_{Q,B}Q\frac{B^*}{B^*+K_{Q,B}} + \cdots \\
\frac{\mathrm{d}B^*}{\mathrm{d}t} &= -k_{\mathrm{deg},B}B^* + k_{P,B}P\frac{B}{B+K_P} - k_{Q,B}Q\frac{B^*}{B^*+K_{Q,B}} + \cdots \\
\frac{\mathrm{d}C}{\mathrm{d}t} &= b_{\mathrm{syn},C} - k_{\mathrm{deg},C}C - k_{P,C}P\frac{C}{C+K_{P,C}} + k_{Q,C}Q\frac{C^*}{C^*+K_{Q,C}} + \cdots \\
\frac{\mathrm{d}C^*}{\mathrm{d}t} &= -k_{\mathrm{deg},C}C^* + k_{P,C}P\frac{C}{C+K_{P,C}} - k_{Q,C}Q\frac{C^*}{C^*+K_{Q,C}} + \cdots
\end{aligned}
$$

Here, b_{syn} is a basal synthesis parameter, k_{deg} is a degradation parameter, k_P and k_Q are background activation and inactivation terms, and P and Q are background activation and inactivation enzymes.

3. The ellipses in the above set of equations refer to topology-specific interactions between network components, which are modeled using mass-action kinetics. Although use of the Michaelis–Menten approximation would yield simpler models, the underlying assumptions of this approximation can be violated by the broad variations in parameter values and network topologies that are required in this approach. Indeed, a comparison of Michaelis–Menten and mass-action kinetic models in identifying robustly ultrasensitive networks resulted in qualitatively different results (Shah and Sarkar, unpublished results).

   In a mass-action modeling context, an enzyme binds to its target, forming an intermediate complex that can subsequently either dissociate or yield the interconverted target along with the unchanged enzyme. Hence, the systems model must also account for these intermediate complexes. Since there are six species, and each species can complex with every species (including itself) in the universe of network topologies, this yields a total of $n(n+1)/2 = 6 \times 7/2 = 21$ additional binary complex species. For a given network, only a small subset of these complex species will be relevant; however, explicit accounting of all $21 + 6 = 27$ species simplifies model implementation in code.

   Each interaction between components falls into one of the following categories:

   (a) *Activation by an enzyme*: in this case, the active version of an enzyme component (e.g., B^*) complexes with the inactive version of its target component (e.g., C). This complex (W) can either dissociate, releasing the two original species ($B^* + C$), or yield the original active version of the enzyme along with the newly active version of the target component ($B^* + C^*$).

$$B^* + C \xrightarrow{k_{0,B^*C}} W$$

$$W \xrightarrow{k_{1,B^*C}} B^* + C$$

$$W \xrightarrow{k_{2,B^*C}} B^* + C^*$$

$$\begin{aligned}
\frac{\mathrm{d}C}{\mathrm{d}t} &= -k_{0,B^*C}B^*C + k_{1,B^*C}W + \cdots \\
\frac{\mathrm{d}W}{\mathrm{d}t} &= k_{0,B^*C}B^*C - k_{1,B^*C}W - k_{2,B^*C}W + \cdots \\
\frac{\mathrm{d}C^*}{\mathrm{d}t} &= k_{2,B^*C}W + \cdots \\
\frac{\mathrm{d}B^*}{\mathrm{d}t} &= -k_{0,B^*C}B^*C + k_{1,B^*C}W + k_{2,B^*C}W + \cdots
\end{aligned}$$

(b) *Inactivation by an enzyme*: in this case, the active version of an enzyme component (e.g., B^*) complexes with the active version of its target component (e.g., C^*). This complex (W) can either dissociate, releasing the two original species ($B^* + C^*$), or yield the original active version of the enzyme along with the inactive version of the target component ($B^* + C$).

$$B^* + C^* \xrightarrow{k_{0,B^*C^*}} W$$

$$W \xrightarrow{k_{1,B^*C^*}} B^* + C^*$$

$$W \xrightarrow{k_{2,B^*C^*}} B^* + C$$

$$\begin{aligned}
\frac{\mathrm{d}C^*}{\mathrm{d}t} &= -k_{0,B^*C^*}B^*C^* + k_{1,B^*C^*}W + \cdots \\
\frac{\mathrm{d}W}{\mathrm{d}t} &= k_{0,B^*C^*}B^*C^* - k_{1,B^*C^*}W - k_{2,B^*C^*}W + \cdots \\
\frac{\mathrm{d}C}{\mathrm{d}t} &= k_{2,B^*C^*}W + \cdots \\
\frac{\mathrm{d}B^*}{\mathrm{d}t} &= -k_{0,B^*C^*}B^*C^* + k_{1,B^*C^*}W + k_{2,B^*C^*}W + \cdots
\end{aligned}$$

(c) *Augmented synthesis by a transcription factor:* in this case, the active version of the transcription factor component (e.g., B^*) upregulates the inactive version of its target species (e.g., C).

$$\frac{\mathrm{d}C}{\mathrm{d}t} = v_{BC}\frac{(B^*)^{n_{\mathrm{H}}}}{(B^*)^{n_{\mathrm{H}}} + \left(K_{\mathrm{syn},BC}\right)^{n_{\mathrm{H}}}} + \cdots$$

$K_{\text{syn},BC}$ is the concentration of B^* at which the rate of synthesis of its target species C is half-maximal. The synthesis Hill coefficient, n_H, should be treated as a constant parameter and, in the default scenario, should be set to a value of 1. However, if there is evidence from parts characterization data or if it is known that the effective synthesis interaction is nonlinear with respect to the transcription factor concentration, n_H can be set to a higher value. Repeating the entire analysis under different constant values of n_H can also be informative.

(d) *Repressed synthesis by a transcription factor:* in this case, the active version of the transcription factor component (e.g., A^*) inhibits all non-basal synthesis of its target component (e.g., synthesis of C by B^*).

$$\frac{\mathrm{d}C}{\mathrm{d}t} = v_{BC}\frac{(B^*)^{n_\text{H}}}{(B^*)^{n_\text{H}} + \left(K_{\text{syn},BC}(1 + A^*)\right)^{n_\text{H}}} + \cdots$$

4. If the target behavior of interest involves input from an external stimulus, it can be accommodated by adding a few more terms to the model. For example, the following reactions describe how to model A as a receptor that is activated upon binding of a ligand stimulus, S.

$$\frac{\mathrm{d}A}{\mathrm{d}t} = -k_{0,SA}SA + k_{1,SA}A^* + \cdots$$
$$\frac{\mathrm{d}A^*}{\mathrm{d}t} = k_{0,SA}SA - k_{1,SA}A^* + \cdots$$

As an example, the network topology depicted in Fig. 1 includes enzymatic activation (of C by A), enzymatic inactivation (of B by A and of C by B), transcriptional upregulation (of C by itself), transcriptional repression (of A by C), and activation of a receptor (A) by an external stimulus.

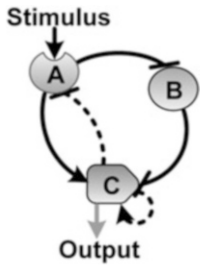


Fig. 1 Example network topology. In this topology, *A* and *B* are enzymes and *C* is a transcription factor. *A* enzymatically activates *C* and inactivates *B*. *B* enzymatically inactivates *C*. *C* transcriptionally upregulates itself and represses *A*. *A* receives the input *Stimulus* in the form of a ligand, and the output, *C*, drives the response of the system [4]

2.3 Sample Sets of Parameter Values

To prepare the systems models built in the previous step for simulation, apply parameter sets to each network model. The following approach is based on Latin hypercube sampling [6].

1. Define the sampling depth, D, or the number of different parameter sets that will be applied to each network topology. *See* Note 1 in Section 3 for a discussion on how to choose a value for D.
2. Group all possible parameters by type (e.g., complex dissociation rates), and for each parameter group, define a range of biologically reasonable values over which the parameters in that group will be sampled.
3. Divide the range for each parameter to yield a sequence of D values, spaced equally on a logarithmic scale.
4. Assemble all parameter values into a matrix such that each column represents a different parameter (and hence the matrix has D rows).
5. For each column in the matrix, separately permute the rows, leaving the rest of the matrix intact.

2.4 Implement and Simulate Network Models

Due to the size of the search space, it would be impractical to enumerate all networks and build models for each individually; instead, these steps should be done programmatically. We recommend the following general strategies.

1. Network enumeration can be practically implemented as follows. A three-component network topology can be represented by a 3×3 interaction matrix, M. The entry at $M(i,j)$ represents the interaction by i on j. Each entry can take on one of three unique values (representing activating, inactivating, or non-existent interactions). Enumerating all possible matrices yields 19,683 unique matrices, or network topologies.
2. Model building can be implemented using a single function that is organized as follows. This function takes as input:
   (a) An interaction matrix, M, specifying the network topology (*see* Subheading 2.1, **step 2**)
   (b) A vector, I, specifying the identities of the components (*see* Subheading 2.1, **step 5**)
   (c) A vector of parameter values, specifying the kinetic constants of the various interactions (i.e., a single row from the permuted parameter value matrix discussed in Subheading 2.3)
   (d) A vector of instantaneous species concentrations

   The general algorithm implemented by the function is as follows:

   (a) Initialize a new vector to hold the instantaneous rates for each species.
   (b) Update the rate vector with basal synthesis, degradation, and background activation/inactivation terms.

(c) Iterate through all positions in the interaction matrix and, for interactions that exist, evaluate the relevant terms and update the rate vector accordingly.

Hence, to simulate a single network topology under given parameter values, this function can simply be passed to a numerical solver, along with the interaction map, parameter values, and the initial concentrations of species. The solver will repeatedly call the function to get instantaneous rates, and the model building procedures will be opaque to the solver.

The following pseudocode outlines an approach for implementing the algorithm discussed above.

```
SpeciesRates = function GetRates(M, I, Param, SpeciesConcs)
{
        allSpecies <- [AA, AB, AC, AA*, AB*, AC*, A*A, A*B, A*C, A*A*, A*B*, A*C*, BB, BC, BB*,
        BC*, B*B, B*C, B*B*, B*C*, CC, CC*, C*C, C*C*]

        # initialize
        for specie in allSpecies
        {
                SpeciesRates[specie] = -SpeciesConcs[specie] * Param[kdeg,specie]
        }

        # basal synthesis, activation, and inactivation
        for specie in [A, B, C]
        {
                SpeciesRates[specie] += Param[syn,A] - Param[kP,specie] * Param[P] *
                SpeciesConcs[specie]/(SpeciesConcs[specie] + Param[KP,specie]) + Param[kQ,specie*]
                * Param[Q] * SpeciesConcs[specie*]/(SpeciesConcs[specie*] + Param[KP,specie])

                SpeciesRates[specie*] += - Param[kP,specie] * Param[P] *
                SpeciesConcs[specie]/(SpeciesConcs[specie] + Param[KP,specie]) - Param[kQ,specie*]
                * Param[Q] * SpeciesConcs[specie*]/(SpeciesConcs[specie*] + Param[KP,specie])
        }


        for interaction in M
        {
                [actor, target, type] <- M[interaction]

                # e.g. enzymatic activation
                if (I[actor] == E AND type == activation)
                {
                        complex <- actor*|target

                        SpeciesRates[actor*] += -Param[actor,target,k0] * SpeciesConcs[actor] *
                        SpeciesConcs[target] + Param[actor,target,k1] * SpeciesConcs[complex]

                        SpeciesRates[target] += -Param[actor,target,k0] * SpeciesConcs[actor] *
                        SpeciesConcs[target] + Param[actor,target,k1] * SpeciesConcs[complex]

                        SpeciesRates[target*] += Param[actor,target,k2] * SpeciesConcs[complex]

                        SpeciesRates[complex] += Param[actor,target,k0] * SpeciesConcs[actor] *
                        SpeciesConcs[target] - Param[actor,target,k1] * SpeciesConcs[complex] -
                        Param[actor,target,k2] * SpeciesConcs[complex]
                }

                ... # similarly for inactivation and transcription
        }

}
```

2.5 Evaluate Network Robustness

In this step, use the results to score each network individually for robustness in generating the behavior of interest. A simple way to arrive at a score for each network is to compute the proportion of simulations (from D) that yielded the behavior of interest. As an example, for a study of ultrasensitivity, one could track the number of stimulus–response curves that are monotonically increasing with respect to stimulus, steep ($n_H > 3$), and exhibit a meaningful contrast between the high and low points ($C^*_{hi} > (C^* + C)/2$).

If the behavior of interest involves an external stimulus or input, each of the D simulations for a single network would need to be repeated under a range of stimulus concentrations.

2.6 Rank Network Topologies

In this step, rank the network topologies by robustness score. Given the goal of using this analysis to infer general strategies for implementing synthetic circuits, the focus should not be on individual high-ranking topologies, but on general patterns. In this regard, visualizing the top 50–100 network topologies in a single heatmap (Fig. 2) can be very informative.

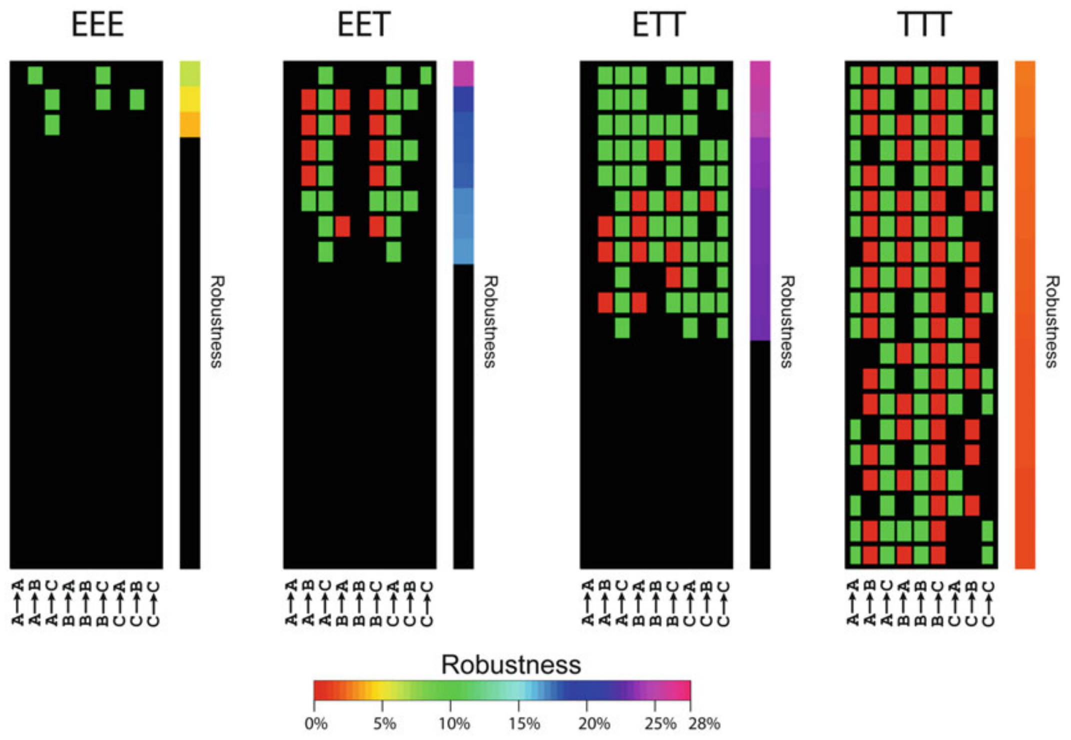


Fig. 2 Network topologies ranked by robustness in generating ultrasensitivity. Only network topologies ranking in the 100 most robust networks were included. Networks with additional, non-contributing interactions were filtered. EEE, EET, ETT, and TTT represent different compositional classes as described in the text. Within each of the four blocks (corresponding to the EEE, EET, ETT, and TTT compositional classes), each row denotes a network topology: *green rectangles* denote positive interactions (activation or synthesis upregulation), while *red rectangles* denote negative interactions (inactivation or synthesis suppression). For instance, the first row in the EEE block denotes a network in which enzyme *A* activates enzyme *B*, which in turn activates enzyme *C* [4]

After visualizing the top-ranked networks, we recommend performing an additional pruning step to filter networks with excessive interactions. This can be accomplished by comparing each network in the set of most robust networks to every other network in the set. If a network X is both a proper subnetwork of another network Y (if the interaction matrix for X, M_X, can be derived from M_Y by setting one or more elements within M_Y to zero, then X is a proper subnetwork of Y) and has the higher robustness score of the two, then the larger network can be eliminated from the list. This procedure removes network interactions that do not contribute to overall robustness.

2.7 Cluster Network Topologies

To help uncover less intuitive families of network topologies enabling the behavior of interest, filter the list of all network topologies to keep only those achieving above-threshold robustness scores. Each network topology in this list can be interpreted as a nine-dimensional vector, as described above. Define the similarity between two network topologies as the number of interactions in common minus the number of interactions that are different. Compute similarity for all pairs of networks in the list to yield a similarity matrix, which can subsequently be passed to a hierarchical clustering algorithm [7] to create visualizations in which more general topology families can be identified.

3 Notes

1. Parameter sampling
   The sampling depth, D, should be selected based on the behavior of interest. For switch-like behavior, we found $D = 1{,}000$ to be adequate; however, for other behaviors such as oscillations, D might have to be significantly higher to derive meaningful results from the analysis. We recommend starting with $D = 1{,}000$, running all simulations for the analysis, and then performing checks to assess whether the depth is adequate. One simple and informative check involves repeating the entire analysis for a small subset of interesting networks 100 times, and then assessing the distribution of robustness scores for each network for convergence.

   Although it is impossible to select unbiased parameter ranges that are truly representative of the biological parts and system of interest, the above procedure provides a solution that is applicable in a variety of contexts. Of course, the range for a particular parameter can be further constrained, or a nonuniform sampling approach can be used if reliable experimental data is available.

References

1. Kwok R (2010) Five hard truths for synthetic biology. Nature 463:288–290
2. Purnick PEM, Weiss R (2009) The second wave of synthetic biology: from modules to systems. Nat Rev Mol Cell Biol 10:410–422
3. Ma W, Trusina A, El-Samad H et al (2009) Defining network topologies that can achieve biochemical adaptation. Cell 138:760–773
4. Shah NA, Sarkar CA (2011) Robust network topologies for generating switch-like cellular responses. PLOS Comput Biol 7:e1002085
5. Goldbeter A, Koshland DE (1981) An amplified sensitivity arising from covalent modification in biological systems. Proc Natl Acad Sci U S A 78:6840–6844
6. Iman RL, Davenport JM, Zeigler DK (1980) Latin Hypercube Sampling (Program User's Guide). Technical Report SAND79-1473, Sandia National Laboratories, Albuquerque, NM
7. Hastie T, Tibshirani R, Friedman J (2013) The elements of statistical learning. Springer, New York

Chemical Master Equation Closure for Computer-Aided Synthetic Biology

Patrick Smadbeck and Yiannis N. Kaznessis

Abstract

With inexpensive DNA synthesis technologies, we can now construct biological systems by quickly piecing together DNA sequences. Synthetic biology is the promising discipline that focuses on the construction of these new biological systems. Synthetic biology is an engineering discipline, and as such, it can benefit from mathematical modeling. This chapter focuses on mathematical models of biological systems. These models take the form of chemical reaction networks. The importance of stochasticity is discussed and methods to simulate stochastic reaction networks are reviewed. A closure scheme solution is also presented for the master equation of chemical reaction networks. The master equation is a complete model of randomly evolving molecular populations. Because of its ambitious character, the master equation remained unsolved for all but the simplest of molecular interaction networks for over 70 years. With the first complete solution of chemical master equations, a wide range of experimental observations of biomolecular interactions may be mathematically conceptualized. We anticipate that models based on the closure scheme described herein may assist in rationally designing synthetic biological systems.

Key words Synthetic biology, Computer-aided design, Multiscale models, Chemical master equation, Closure schemes

1 Introduction

Numerous synthetic gene circuits have been created in the past decade, including bistable switches, oscillators, inducible activators, and logic gates [1–6]. Applications abound from biofuel synthesis to antibiotic technologies [7–10]. Designing synthetic gene regulatory networks now takes advantage of an ever-expanding toolbox of molecular components and of developed technologies for inexpensively manipulating DNA sequences. Although recently developed designs of regulatable gene networks are ingenious, there are still numerous limitations in rationally engineering synthetic biological systems. This is especially true if targeted phenotypes are nonlinear, such as the ones observed in bistable or oscillatory gene networks.

Mario Andrea Marchisio (ed.), *Computational Methods in Synthetic Biology*, Methods in Molecular Biology, vol. 1244, DOI 10.1007/978-1-4939-1878-2_9, © Springer Science+Business Media New York 2015

Multiscale mathematical tools have been developed in an attempt to rationalize synthetic biology [11–18]. Multiscale models that expand on traditional mathematics developed by scientists and engineers are necessary to model kinetic and thermodynamic processes. Biological phenotypes can be reduced to networks of biomolecular interactions, with the time and length scales of these often spanning several orders of magnitude. Although the principles of thermodynamics, kinetics and transport phenomena apply to biological systems, these systems differ from traditional, industrial-scale chemical systems in an important, fundamental way: they are occasionally far from the thermodynamic limit. This theoretical limit is attained when the number of molecules of molecular species in the system increases toward infinity. However, the fact that biomolecular systems can be very far from the thermodynamic limit, with reactants/products numbering only very small numbers of molecules in the system, hinders the use of continuous-deterministic models. Indeed, using ordinary differential equations for simulating the reaction kinetics of these systems can be distinctly false, especially if experimentally observed nonlinearities are to be captured by the models. The need arises then for stochastic models that account for inherent, thermal noise, which is manifest as phenotypic distributions at the population growth/interaction levels.

This assessment is not new. The importance of modeling formalisms appropriate for systems away from the thermodynamic limit was recognized more than 50 years ago by McQuarrie, Moyal, and Oppenheim [19–23], among others. These physical chemists developed the chemical master equation (CME) that follows the time changes of the probability the state is at any point in the available state space.

Without the ability to solve the CME, scientists turned to approximations. In 1976, Daniel Gillespie developed a computer algorithm that could sample the master probability distribution with numerical simulations of networks of reactions [24, 25]. Although Gillespie's methods were not widely recognized for almost 20 years, his algorithms found fertile ground for development in efforts to model biological systems. Nowadays, a community of scientists and engineers is continually working to improve the computational efficiency and accuracy of algorithms that simulate chemical reacting systems [26–35].

In the following section, we introduce the chemical master equation and discuss the challenges faced when developing solutions. We then summarize algorithms based on Gillespie's stochastic simulation algorithm (SSA). We return to the master equation and present the recently developed zero-information closure scheme. Finally we present a simple example of a bistable reaction networks and the solution of the master equation. We conclude by speculating on the impact a closure scheme may have on computer-aided synthetic biology.

2 Methods for Multiscale Models of Biomolecular Systems

2.1 Chemical Master Equation

The chemical master equation (CME) is the mathematical foundation for modeling stochastic chemical reactions [19–23]. The common form is equivalent to the more general Chapman–Kolmogorov equation as applied to Markov processes. In particular, it will be a Markov chain with a discrete set of possible states, or "state space", occurring in continuous time. Here, states refer to numbers of molecules present in the system. In general, for time t, this will be a vector $\underline{X}(t) = [X_1 \ldots X_N]$, where X_i is number of molecules of the i-th chemical species with $i = 1, \ldots, N$.

Transitions between states of the Markov chain occur when a chemical reaction occurs. Reactions in biological systems may include covalent reactions, bindings, conformational changes, transcriptional elongation events, etc.

The general form of the CME is:

$$\frac{\partial P(\underline{X};t)}{\partial t} = \sum_{\underline{X}'} \left[T\left(\underline{X}\middle|\underline{X}'\right) P(\underline{X}';t) - T\left(\underline{X}'\middle|\underline{X}\right) P(\underline{X};t) \right] \quad (1)$$

Where $P(\underline{X};t)$ is the probability of being in state $\underline{X}$ at time t, and $T\left(\underline{X}\middle|\underline{X}'\right)$ is the transition probability of going from state $\underline{X}'$ to state $\underline{X}$ per unit time. The CME describes the dynamics of stochastic systems exactly, but has been until recently, for all but the simplest systems, mathematically intractable [19].

The reason analytical solutions to the chemical master equation remained elusive becomes clear when the master equation is recast in equivalent terms of probability moments—the probability distribution average, the variance, and so on:

$$\frac{\partial \underline{\mu}}{\partial t} = A\underline{\mu} + A'\underline{\mu}' \quad (2)$$

where $\underline{\mu}$ is the vector of moments up to order M and A is the matrix describing the linear portion of the moment equations. On the right, $\underline{\mu}'$ is the vector of higher-order moments, and the corresponding matrix A'. Generating the matrices in Eq. 2 can be performed either analytically [36, 37] or numerically [38]. For linear systems with only zeroth or first-order reactions, A' is empty. For other systems, A' is not empty and the set of ODEs becomes infinite, and thus intractable.

Consequently, the Gillespie stochastic simulation algorithm (SSA) and the Chemical Langevin Equation (CLE) formalism were developed to approximate the dynamic solution to the CME.

2.2 Stochastic-Discrete and Stochastic-Continuous Algorithms

The SSA utilizes Monte Carlo sampling to circumvent the mathematical difficulties inherent to stochastic simulation. Gillespie proved that, with an assumption of a well-mixed volume, chemical reactions can be defined as exponentially distributed random events. For example, given a reaction j with propensity $\alpha_j(\underline{X})$, which is analogous to the reaction macroscopic rate and that depends on the current state $\underline{X}$, the time to the next reaction is determined using a uniform random number (URN) as:

$$\tau_j = -\frac{\ln \mathrm{URN}}{\alpha_j(\underline{X})} \tag{3}$$

Using this characterization, individual trajectories through time can be generated for any arbitrary chemical reaction network. The exact probability distribution can be rebuilt from an ensemble of individual trajectories. This method was extended and made efficient by Gibson and Bruck [26] and is best described as a stochastic-discrete-space algorithm.

In principle, the SSA produces exact distributions at the limit of infinite simulated trajectories and thus can be made arbitrarily accurate. The primary drawback is that producing sufficient numbers of trajectories can be computationally taxing. For systems in which many reactions occur in short periods of time the computational load is substantial.

The common situation in which the Gillespie algorithm becomes prohibitively expensive is when the state space can be accurately described as a continuum. In such cases the number of reactions occurring in a short period of time can become effectively infinite. The chemical Langevin equation is then an alternative to the SSA that maintains accuracy but avoids the computational costs of a discrete-space algorithm. A Langevin equation is a stochastic differential equation (SDE) that adds a random variable to a deterministic system to produce fluctuations. For stochastic chemical systems the CLE is commonly written as:

$$\mathrm{d}\underline{X} = \sum_j \left[\alpha_j(\underline{X})\underline{v_j}\mathrm{d}t + \sqrt{\alpha_j(\underline{X})}\underline{v_j}\mathrm{d}w_j\right] \tag{4}$$

where $\alpha_j(\underline{X})$ is the propensity for reaction j as a function of the state $\underline{X}$, $\underline{v_j}$ is the stoichiometric vector for reaction j, and $\mathrm{d}W_j$ is a Wiener process, a continuous-time stochastic process, also called Brownian motion. The CLE is a stochastic-continuous-space algorithm.

The CLE is a good approximation of the CME when the state space is roughly continuous. The primary drawback is accuracy. When the continuous state space approximation does not hold, accuracy is not guaranteed.

2.3 Hybrid Algorithms

The drawbacks of SSA algorithms are especially important when considering biological systems. First, many biological models can be described as stiff in that there are multiple time scales involved. For biological systems, using the SSA can be time consuming, but using the CLE can be inaccurate. Second, in biological systems many reactions change in the course of a simulation. In particular, consider an oscillatory system in which components will alternate from nearly zero to a large number and back relatively quickly. Thus, any solution must be able to handle the determination of fast and slow reaction sets dynamically. The key to fast and accurate biological simulation is in selectively utilizing the appropriate algorithms.

Algorithms that utilize both the SSA and CLE are called hybrid stochastic algorithms. The reaction set is split into two regimes, fast reactions and slow reactions, and continuous-space stochastic algorithms are utilized, when possible, for the fast reactions. For stiff systems a hybrid algorithm must be able to handle disparate time scales efficiently and to dynamically characterize reactions as slow and fast. The Hybrid Stochastic Simulator for Supercomputers (Hy3S) is such an algorithm, designed specifically for biological model simulation on supercomputers [13].

The first aspect of Hy3S to be discussed is dynamic partitioning of a reaction network into a set of fast reactions and slow reactions, such that different algorithms can be applied to each set. Initially all reactions are defined as slow reactions and redefined as fast if two conditions hold. First, a reaction is fast if many reaction events occur in a small increment of time. This condition is defined by the parameter λ and mathematically defined as:

$$\alpha_j(\underline{X})\Delta t \geq \lambda \gg 1 \tag{5}$$

Second, for fast reactions the effect of the reaction on the reactants and products must be sufficiently small. This condition is described by a parameter ε, and mathematically defined as:

$$X_i(t) > \varepsilon \cdot \left|v_{i,j}\right| \tag{6}$$

where i is a reactant or product of reaction j. The common values for λ and ε are 10 molecules/s and 100 molecules, respectively [13].

The reactions classified as fast are then simulated using the Chemical Langevin Equation, while the slow reactions utilize the SSA. Stochastic differential equations like the CLE can be integrated using different algorithms depending on the desired accuracy. In the case of Hy3S two methods in particular are used, the Euler–Maruyama method and the Milstein method [13].

In order to maintain accuracy in both regimes a method for determining when a slow reaction, and which slow reaction, occurs is necessary. In the Hy3S suite this is done by tracking zero-crossing events for each slow reaction. The variable tracked is denoted by R_j where j is one of the slow reactions in a system. The governing equation is then:

$$\frac{\mathrm{d}R_j\big|_t}{\mathrm{d}t} = \alpha_j(\underline{X})\big|_t \tag{7}$$

with an initial condition of $R_j|t_o = \ln \mathrm{URN}_j$. Zero-crossing events occur when $R_j > 0$ at the end of a time step.

Numerous methods have been developed for reducing run times for complex stochastic system, such as quasi-steady state approximations [29], and dynamic integration stepping techniques [30].

In all, there are eight hybrid algorithms in the Hy3S suite depending on which integration method is used (Euler–Maruyama or Milstein), the time-step (Fixed or adaptive), and whether the algorithm is parallelized (serial or MPI). Along with these algorithms, a standard SSA (serial and MPI) is included making the total number of algorithms ten.

2.4 Zero-Information Closure of the Master Chemical Equation

Recently, we have presented a zero information (ZI) closure scheme for the chemical master equation of reaction networks that may include nonlinear reaction kinetics. This section explains the theoretical underpinnings of the zero-information moment closure method. The section starts with an outline of ZI-closure and then provides the algorithm for determining the maximum entropy distribution. It concludes with a Schlögl model example to clarify the calculation of several important equations [39, 40]. This section expands upon a previously described ODE solving scheme and steady-state determination method [39].

A closure scheme is used to determine the key relationship between higher and lower-order moments necessary to solve the moment equations in Eq. 2:

$$\underline{\mu}' = F\left(\underline{\mu}\right) \tag{8}$$

In most closure schemes $F(\underline{\mu})$ is an analytical expression related to a well-known characteristic equation. In the case of ZI-closure the closure scheme refers to the maximum-entropy distribution or most-likely distribution.

For a single component system with a probability distribution $p(x)$, the information entropy is defined as:

$$H = -\sum_{x=0}^{\infty} p(x) \ln p(x) \tag{9}$$

For an unconstrained system the resulting maximum-entropy distribution is simply uniform. However, values of the lower-order moments, $\underline{\mu}$, act as constraints on the system. The result is best solved using a Lagrange multiplier method (here assuming a simple component with M known lower order moments):

$$
\begin{aligned}
\Lambda &= H - \lambda_0 g_0 - \lambda_1 g_1 - \cdots - \lambda_M g_M \\
g_0 &= \sum_{X=0}^{\infty} p(x) - 1 \\
g_1 &= \sum_{x=0}^{\infty} xp(x) - \langle x \rangle \\
&\vdots \\
g_M &= \sum_{x=0}^{\infty} x^M p(x) - \langle x^M \rangle
\end{aligned}
\tag{10}
$$

Taking Eq. 10, the maximum is found by differentiating by $p(x)$ and setting the result to zero:

$$\frac{\partial \Lambda}{\partial p(x)} = -\ln p(x) - 1 - \lambda_0 - \lambda_1 x - \cdots - \lambda_M x^M = 0 \tag{11}$$

or, trivially:

$$p_H(x) = \exp\left(-1 - \lambda_0 - \lambda_1 x - \cdots - \lambda_M x^M\right) \tag{12}$$

An analytical expression for the maximum entropy distribution is determined with the same number of parameters as the number of known lower-order moments.

With Eq. 9 it should be obvious how Eq. 2 is satisfied. Using the lower-order moments, $\underline{\mu}$, the maximum entropy distribution is found by determining the Lagrange parameters, $\underline{\lambda}$. The maximum-entropy moments, denoted with a subscript H, can be determined from Eq. 9. Take, for example, the determination of $\langle x^m \rangle_H$ in a single component system:

$$\langle x^m \rangle_H = \sum_{x=0}^{\infty} x^m p_H(x) \tag{13}$$

Again, this method is extended to multi-component systems. Zero-information closure uses maximum-entropy moments as approximations for higher-order moments in simulation:

$$\underline{\mu}' = \underline{\mu}'_H \tag{14}$$

Note that the order of closure, M, used throughout the section is applied without drawing any conclusions about accuracy. At present the order of closure is chosen in a trial-and-error fashion. In the future, error analysis may be used to determine the optimal closure order using ZI-Closure.

The method used within this study for the determination of the Lagrange parameters, $\underline{\lambda}$, given a set of known lower-order moments, $\underline{\mu}$, is a simple Newton–Raphson optimization scheme, although more sophisticated algorithms may be explored.

Table 1
Schlögl model network description

Model	Schlögl
Reactions	$2X + A \xrightarrow{k_1} 3X$ $3X \xrightarrow{k_2} 2X + A$ $B \xrightarrow{k_3} X$ $X \xrightarrow{k_4} B$
Degrees of freedom	X
Initial condition	$X_0 = 25$

2.5 ZI-Closure Example

To conclude this section, results for zero-information closure will be presented along with an example of a novel analytical method now available for network analysis. The Schlögl model was chosen for two main reasons. First, it may exhibit two main peaks in its probability distribution (bimodality). It can thus require up to 12 probability distribution moments to accurately reproduce the underlying distribution, and demonstrates some of the necessary versatility of the ZI-closure scheme. Second, the network characteristics are very sensitive to the chosen kinetic parameters, and it is thus ideal for demonstrating some of the novel stochastic analytical methods available using moment closure. *See* Table 1 for the network description.

As a dual peaked distribution, the Schlögl model requires up to 12 lower-order moments in order to reproduce its complex function, which is impossible with analytical closure schemes. Figure 1 demonstrates how accuracy improves as the closure-order is increased. Several maximum entropy optimizations are executed using an increasing closure order (from second to tenth-order closure).

With moment closure it is now possible to obtain steady-states quickly using appropriate closure schemes. In particular, by setting the left side of Eq. 2 to zero, a steady-state distribution is immediately available. Figure 2 shows an example of the profile obtained by solving for steady-states across a range of key values. The parameter range (k_4 from 2 to 5 s^{-1}) was chosen such that the left and right sides of the graph have a single peaked distribution, whereas the blue shaded region in the middle is where the steady-state distribution has two peaks. This is explicitly shown in Fig. 3 as well, where three example parameter values show how the Schlögl model flows from normally distributed to bimodal and back again.

These examples show two important features of ZI-Closure. First, there are 300 steady-state values obtained using ZI-Closure in Fig. 2. While this number of simulations would be impractical for SSA simulations (indeed, hundreds of hours of simulation time

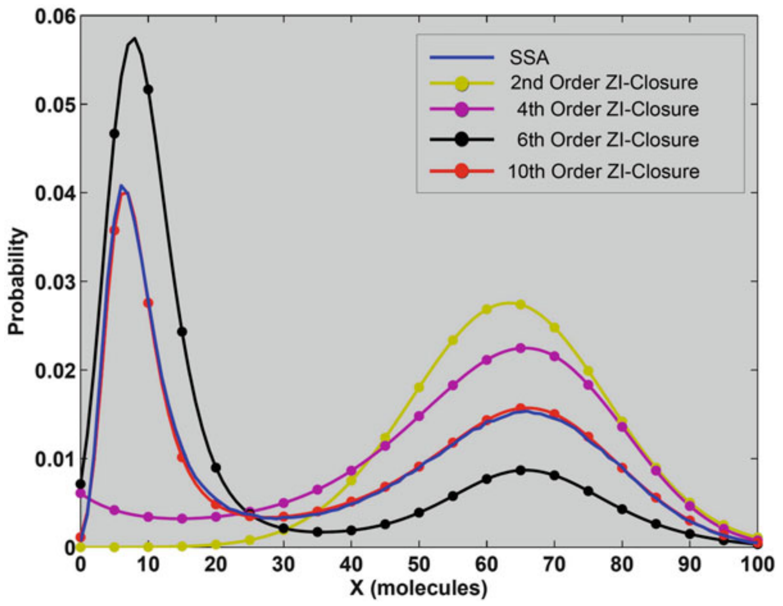


Fig. 1 Maximum entropy demonstration. The parameters are $k_1 = 0.15$ (molec.$^{-1}$ s^{-1}), $k_2 = 0.0015$ (molec.$^{-2}$ s^{-1}), $k_3 = 20$ molec.-s^{-1}, and $k_4 = 3.5$ s^{-1}. The stochastic simulation algorithm results are an exact reproduction of the chemical master equation solution (*blue, solid line*). As more moments are added to the maximum entropy optimization program (from 2 to 10 moments), the optimization results become more representative of the exact solution distribution

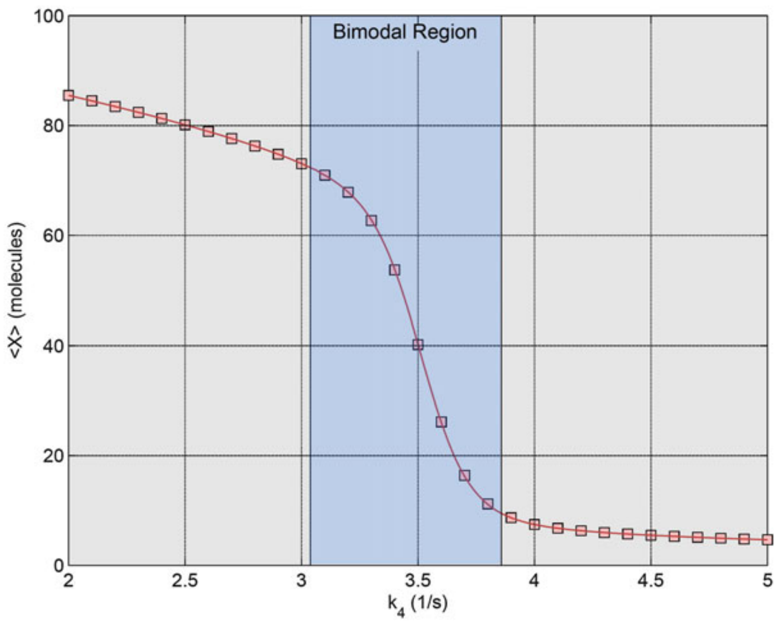


Fig. 2 Schlögl sensitivity analysis. The parameters are $k_1 = 0.15$ molec.$^{-1}$ s^{-1}, $k_2 = 0.0015$ molec.$^{-2}$ s^{-1}, and $k_3 = 20$ molec.-s^{-1}. The range for k_4 was chosen such that at low values (*left side*) the distribution has a single peak, for middle range values (*blue region*) the distribution has two peaks, and at high values (*right side*) it returns to a single peaked distribution

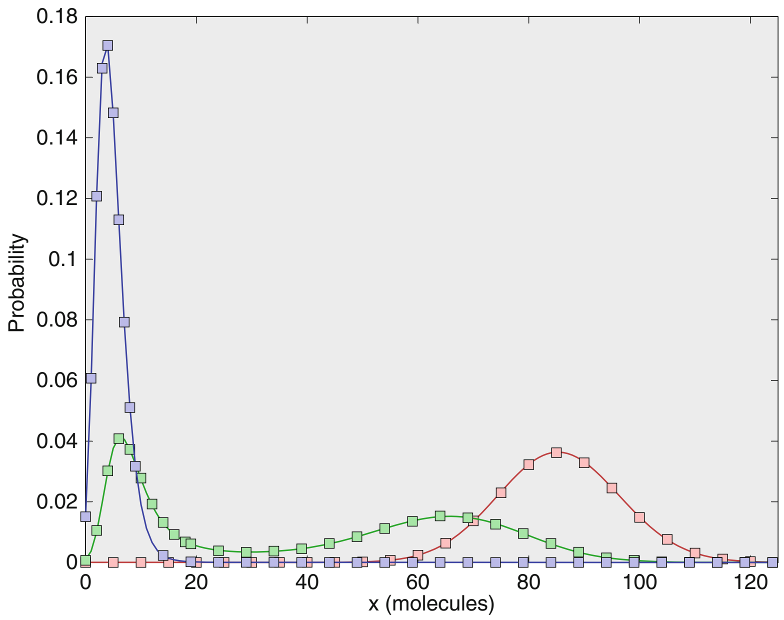


Fig. 3 Schlögl distribution. The parameters are $k_1 = 0.15$ molec.$^{-1}$ s^{-1}, $k_2 = 0.0015$ molec.$^{-2}$ s^{-1}, and $k_3 = 20$ molec.-s^{-1}. Three k_4 values were chosen: $k_4 = 2$ s^{-1} (*red*), $k_4 = 3.5$ s^{-1} (*green*), and $k_4 = 5$ s^{-1} (*blue*). The distribution goes from a single peak to two peaks and back. The SSA results are represented as *squares*, while the ZI-Closure results (12th order closure) are *lines*, demonstrating the versatility of the method

were used to obtain the 31 SSA points for comparison), it takes a matter of minutes with ZI-closure scheme. With ZI-closure, steady-state optimization can be almost immediate for some small systems. Second, the line itself is, for all practical purposes, a sensitivity analysis for parameter k_4 for the Schlögl system. Sensitivity analysis is an important tool in chemical reaction design because it can identify the parameters that most influence a specific behavior in a system. In this case it is clear that in the bimodal region the distribution mean is very sensitive to the value of k_4. This type of analysis could have otherwise taken hours of computation time with a direct SSA simulation.

The Schlögl model is a particularly interesting network to investigate because the underlying distribution can take on a range of functions, from a simple normal distribution to a complex bimodal distribution that takes a dozen moments to describe. ZI-Closure can obtain steady-state distributions immediately, and subsequently certain analytical techniques can be performed. A sensitivity analysis for k_4 shows that the steady-state optimization can be obtained quickly relative to the SSA, and also that the identification of important network parameters can be determined immediately. This is just the start of what we think is possible using moment closure techniques. The vast toolbox of deterministic analytical techniques, like eigenvalue and nonlinear analysis, may also become available for stochastic systems by applying ZI-closure to small chemical reaction networks.

3 Conclusions: Future Directions

In order to make stochastic modeling more accessible, we have developed the Synthetic Biology Software Suite (SynBioSS), a software suite that automates the steps for building models of and conducting numerical simulations for synthetic biological systems [41–43]. There are three components in SynBioSS: *Designer*, *Wiki*, and *Simulator*.

With SynBioSS Designer, gene network models are created automatically after the user enters molecular components and their relationships. Every reaction in the model has a corresponding kinetic rate that describes the rate of association of its reactant molecules and the formation or destruction of any covalent bonds or stable non-covalent interactions. SynBioSS Wiki has been specifically created to store and recall just this sort of kinetic data. Simulating gene regulatory networks is feasible with the third component of SynBioSS, the Desktop Simulator. SynBioSS DS can be downloaded as an installation executable for Windows. At the heart of SynBioSS DS has been Hy3S.

We are planning to implement the ZI-closure scheme within SynBioSS. We anticipate that an augmented SynBioSS will aid scientists in modeling gene regulatory networks in a way fit for analysis and design. In particular, the calculation of steady states with ZI-closure affords a parameter sensitivity analysis and stability analysis for stochastic reaction networks. Modeling ranges of kinetic and thermodynamic parameters may rationalize the design of complex, nonlinear networks of reactions. This in turn may assist in identifying components and interactions for synthetic biological systems.

Admittedly, the field of computational synthetic biology is still nascent. A long distance separates theory from practical implementation, but with advances in modeling techniques, from Gillespie's SSA, to hybrid simulation algorithms, to CME closure schemes, this distance is shortened.

Acknowledgements

This work was supported by a grant from the National Institutes of Health (American Recovery and Reinvestment Act grant GM086865) and a grant from the National Science Foundation (CBET-0644792) with computational support from the

Minnesota Supercomputing Institute (MSI). Support from the University of Minnesota Digital Technology Center and the University of Minnesota Biotechnology Institute is also acknowledged.

References

1. Elowitz MB, Leibler S (2000) A synthetic oscillatory network of transcriptional regulators. Nature 403:335–338
2. Gardner TS, Cantor CR, Collins JJ (2000) Construction of a genetic toggle switch in *Escherichia coli*. Nature 403:339–342
3. Andrianantoandro E, Basu S, Karig DK, Weiss R (2006) Synthetic biology: new engineering rules for an emerging discipline. Mol Syst Biol 2:2006.0028
4. Volzing K, Borrero J, Sadowsky MJ, Kaznessis YN (2013) Antimicrobial peptides targeting gram-negative pathogens, produced and delivered by lactic acid bacteria. ACS Synth Biol 2 (11):643–650, PubMed PMID: 23808914
5. Ramalingam K, Maynard J, Kaznessis YN (2009) Forward engineering of synthetic biological AND gates. Biochem Eng J 47:38–47
6. Alon U (2003) Biological networks: the tinkerer as an engineer. Science 301:1866–1867
7. Endy D (2005) Foundations for engineering biology. Nature 438:449–453
8. Volzing K, Biliouris K, Kaznessis YN (2011) proTeOn and proTeOff, new protein devices that inducibly activate bacterial gene expression. ACS Chem Biol 6(10):1107–1116
9. Kaern M, Blake WJ, Collins JJ (2003) The engineering of gene regulatory networks. Annu Rev Biomed Eng 5:179–206
10. Keasling J (2005) The promise of synthetic biology. Bridge Natl Acad Eng 35:18–21
11. Salis H, Kaznessis YK (2005) Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions. J Chem Phys 122:1–13
12. Haseltine EL, Rawlings JB (2002) Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics. J Chem Phys 117:6959–6969
13. Salis H, Kaznessis YN (2005) Numerical simulation of stochastic gene circuits. Comp Chem Eng 29:577–588
14. Cao Y, Li H, Petzold L (2004) Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. J Chem Phys 121:4059–4067
15. Chatterjee A, Mayawala K, Edwards JS, Vlachos DG (2005) Time accelerated monte carlo simulations of biological networks using the binomial {tau}-leap method. Bioinformatics 21:2136–2137
16. Tian T, Burrage K (2004) Binomial leap methods for simulating stochastic chemical kinetics. J Chem Phys 121:10356–10364
17. W E, Liu D, Vanden-Eijnden E (2005) Nested stochastic simulation algorithm for chemical kinetic systems with disparate rates. J Chem Phys 123:194107
18. Munsky B, Khammash M (2006) The finite state projection algorithm for the solution of the chemical master equation. J Chem Phys 124:044104
19. McQuarrie DA (1967) Stochastic approach to chemical kinetics. J Appl Prob 4:413–478
20. Moyal JE (1949) Stochastic processes and statistical physics. J Roy Stat Soc Ser B 11: 150–210
21. Oppenheim I, Shuler KE (1965) Master equations and Markov processes. Phys Rev 138: 1007–1011
22. Oppenheim I, Shuler KE, Weiss GH (1967) Stochastic theory of multistate relaxation processes. Adv Mol Relax Process 1:13–68
23. Oppenheim I, Shuler KE, Weiss GH (1977) Stochastic processes in chemical physics: the master equation. The MIT Press, Cambridge, MA
24. Gillespie DT (1976) A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. J Comput Phys 22:403–434
25. Gillespie DT (1977) Exact stochastic simulation of coupled chemical reactions. J Phys Chem 81:2340–2361
26. Gibson MA, Bruck J (2000) Efficient exact stochastic simulation of chemical systems with many species and many channels. J Phys Chem 104:1876–1889
27. Cao Y, Gillespie DT, Petzold LR (2005) Avoiding negative populations in explicit Poisson tau-leaping. J Chem Phys 123:054104
28. Chatterjee A, Vlachos DG (2006) Temporal acceleration of spatially distributed kinetic monte Carlo simulations. J Comput Phys 211:596–615
29. Salis H, Kaznessis YN (2005) An equation-free probabilistic steady state approximation: dynamic application to the stochastic

simulation of biochemical reaction networks. J Chem Phys 123:214106

30. Sotiropoulos V, Kaznessis YN (2008) An adaptive time step scheme for a system of SDEs with multiple multiplicative noise. Chemical Langevin equation, a proof of concept. J Chem Phys 128:014103
31. Kaznessis Y (2006) Multi-scale models for gene network engineering. Chem Eng Sci 61: 940–953
32. Kaznessis Y (2007) Models for synthetic biology. BMC Syst Biol 1:47
33. Harris LA, Clancy PA (2006) A "partitioned leaping" approach for multiscale modeling of chemical reaction dynamics. J Chem Phys 125: 144107
34. Tuttle L, Salis H, Tomshine J, Kaznessis YN (2005) Model-driven design principles of gene networks: the oscillator. Biophys J 89: 3873–3883
35. Tomshine J, Kaznessis YN (2006) Optimization of a stochastically simulated gene network model via simulated annealing. Biophys J 91:3196–3205
36. Gillespie CS (2009) Moment closure approximations for mass-action models. IET Syst Biol 3:52–58
37. Sotiropoulos V, Kaznessis YN (2011) Analytical derivation of moment equations in stochastic chemical kinetics. Chem Eng Sci 66: 268–277
38. Smadbeck P, Kaznessis YN (2012) Efficient moment matrix generation for arbitrary chemical networks. Chem Eng Sci 84:612–618
39. Smadbeck P, Kaznessis YN (2013) A closure scheme for chemical master equations. Proc Natl Acad Sci U S A 110(35):14261–14265
40. Schlögl F (1972) Chemical reaction models for non-equilibrium phase transition. Z Phys 253:147–161
41. Salis H, Sotiropoulos V, Kaznessis YN (2006) Multiscale Hy3S: hybrid stochastic simulations for supercomputers. BMC Bioinform 7(93): 2006
42. Hill A, Tomshine J, Wedding E, Sotiropoulos V, Kaznessis YK (2008) SynBioSS: the synthetic biology modeling suite. Bioinformatics 24:2551–2553
43. Weeding E, Houle J, Kaznessis YN (2010) SynBioSS designer: a web-based tool for the automated generation of kinetic models for synthetic biological constructs. Brief Bioinform 11(4):394–402

Chapter 10

Feedback Loops in Biological Networks

Elisa Franco and Kate E. Galloway

Abstract

We introduce fundamental concepts for the design of dynamics and feedback in molecular networks modeled with ordinary differential equations. We use several examples, focusing in particular on the mitogen-activated protein kinase (MAPK) pathway, to illustrate the concept that feedback loops are fundamental in determining the overall dynamic behavior of a system. Often, these loops have a structural function and unequivocally define the system behavior. We conclude with numerical simulations highlighting the potential for bistability and oscillations of the MAPK pathway re-engineered through synthetic promoters and RNA transducers to include positive and negative feedback loops.

Key words Feedback, Bistability, Oscillations, MAPK pathway, Synthetic biology

1 Introduction

Cells sense their environment and make decisions through coordinated molecular events. The dynamic interactions among nucleic acids, enzymes, and small molecules define such molecular events and specify their possible outcomes. For example, a set of reactions among a set of enzymes and genes may trigger transient, sustained, or periodic responses in other enzymes, depending on external stimuli [1]. Feedback among molecular components plays a crucial role in defining such complex behaviors, and synthetic feedback loops are routinely designed to redirect cellular responses and fate [2].

Mathematical models capturing the behavior of a molecular system are useful to support and guide experiments [3]. Feedback loops may result in counterintuitive behaviors in a system, thus a combination of numerical and theoretical analysis of a validated model can yield important insights, for example helping to identify the key species and parameters. Models can often be simplified to focus on such key reactions, and it may be possible to achieve valid conclusions on the behavior of the system without having to resort to extensive numerical simulations [4].

Mario Andrea Marchisio (ed.), *Computational Methods in Synthetic Biology*, Methods in Molecular Biology, vol. 1244,
DOI 10.1007/978-1-4939-1878-2_10, © Springer Science+Business Media New York 2015

In this chapter, we focus on classical methods from dynamical systems and control theory that can be used on ordinary differential equation (ODE) models of molecular networks. We begin by briefly introducing ODE models through the mitogen-activated protein kinase (MAPK) pathway, for which a hierarchy of models of different complexity is available in the literature [4–6]. ODE models for molecular networks always include nonlinear terms: we introduce the concept of linearization, through which one can systematically explore the behavior of a system in a neighborhood of its stationary points. We illustrate this simple method with several examples, in particular the MAPK pathway.

In Subheading 4.1 we highlight the concept that feedback loops can unequivocally determine the possible dynamic responses of a system. Some of the first and best known mathematical conjectures in this area were formulated by R. Thomas [7], and focus on the presence of positive or negative feedback loops in the linearized model of a system (loops in the Jacobian graph): a negative feedback loop is a necessary condition for stable periodic behavior, while a positive loop is a necessary condition for multistationarity (*see* [8] for a very thorough survey). These conjectures were proved in [9] and [10], with several further extensions and refinements [11–14]. While Thomas' conjectures are only necessary, they have been helpful in guiding the design of numerous synthetic molecular circuits [15–24]. We conclude with numerical simulations exploring the potential for bistability and oscillations of the MAPK pathway in yeast, re-engineered to include artificial positive and negative feedback through synthetic promoters and RNA gates [25].

2 Dynamic Models for Molecular Systems

Deterministic ODEs are commonly adopted in conventional engineering fields: ODEs are easily derived directly from the laws of physics, thermodynamics, and electromagnetism, and are a good description of macroscopic systems where stochastic effects are negligible. Molecular systems operating at high copy numbers have been successfully modeled using ODEs; for gene networks, alternative descriptions include stochastic equations or boolean models [3]. The MAPK pathway is a well-known signal transduction network which has been successfully modeled using ODEs: we will use it as our example system throughout this chapter.

One can identify two main approaches to the derivation of ODE models for biochemical systems. The first is a mechanistic approach, whereby the modeler tries to identify all possible chemical reactions that contribute to the process behavior; this approach is particularly fruitful in well-characterized systems (for example, understood model pathways or in vitro networks), but the resulting models may be extremely complex and require heavy numerical treatment. The famous Huang–Ferrell model of the MAPK pathway [5] is one of the best examples of this approach. A list of ten reactions is used to model the three-stage, double phosphorylation pathway, and build 18 ODEs with 30 parameters using the mass action kinetics formalism. To illustrate this process, we consider solely the activation stage of the cascade, where MAPKKK, which we denote as m_{3K} for brevity, is activated (m^*_{3K}) and inactivated, respectively, by two "input" enzymes u_1 and u_2. These reactions are:

$$m_{3K} + u_1 \underset{r_1}{\overset{f_1}{\rightleftharpoons}} m_{3K} \cdot u_1 \xrightarrow{k_1} m^*_{3K} + u_1,$$
$$m^*_{3K} + u_2 \underset{r_2}{\overset{f_2}{\rightleftharpoons}} m^*_{3K} \cdot u_2 \xrightarrow{k_2} m_{3K} + u_2.$$

The corresponding ODEs associated with these isolated reactions for M_{3K} activation/inactivation are:

$$\begin{aligned}
\frac{d\,m_{3K}}{dt} &= -f_1 m_{3K}\, u_1 + r_1 m_{3K} \cdot u_1 + k_2 m^*_{3K} \cdot u_2,\\
\frac{d\,m_{3K} \cdot u_1}{dt} &= +f_1 m_{3K}\, u_1 - (r_1 + k_1) m_{3K} \cdot u_1,\\
\frac{d\,m^*_{3K}}{dt} &= -f_2 m^*_{3K}\, u_2 + r_2 m^*_{3K} \cdot u_2 + k_1 m_{3K} \cdot u_1,\\
\frac{d\,m^*_{3K} \cdot u_2}{dt} &= +f_2 m^*_{3K}\, u_1 - (r_2 + k_2) m^*_{3K} \cdot u_1.
\end{aligned}$$

However, when considered in the context of the entire pathway, M^*_{3K} binds and phosphorylates M_{2K}: thus, the ODEs describing the dynamics of M^*_{3K} include additional second order terms. This example highlights the rapidly growing size and complexity of detailed models built using mass action kinetics. Nevertheless, it must be noted that the mass action formalism allows to derive ODEs systematically once reactions are specified, and many free software tools are available to automatically perform this operation [26, 27].

The second approach is phenomenological and driven by sensible approximations that describe qualitatively the observed dynamics; this approach generally yields models more amenable to analytical treatment, which however may not capture faithfully the system's dynamics and ignore several sources of uncertainty. Using a combination of mathematical analysis and numerical simulations supported by experimental data, the Huang–Ferrell model can be collapsed into a simpler, gray-box model where several intermediate reactions are captured by cooperative Hill functions. For instance, the dynamics of a kinase species x being doubly phosphorylated by its input u (where the input is the upstream kinase), yielding active kinases x_p and $x_{p\ p}$, can be written as [28]:

$$\begin{aligned}
\frac{dx}{dt} &= -uk_1\frac{x}{K_1+x} + V_2\frac{x_p}{K_2+x_p},\\
\frac{dx_p}{dt} &= uk_1\frac{x}{K_1+x} - V_2\frac{x_p}{K_2+x_p} - uk_3\frac{x_p}{K_3+x_p} + V_4\frac{x_{pp}}{K_4+x_{pp}},\\
\frac{dx_{pp}}{dt} &= uk_3\frac{x_p}{K_3+x_p} - V_4\frac{x_{pp}}{K_4+x_{pp}},
\end{aligned}$$

where V_i are the maximal enzyme rates, k_i are the catalytic rate constants, and K_i are the Michaelis constants [28]. The readers familiar with Michaelis–Menten enzyme kinetics will immediately recognize that the functional terms in the equations above come from a simple assumption of timescale separation between the binding/unbinding dynamics of an enzyme to its substrate, and the catalytic step of the reaction. By solving numerically the equations above for plausible reaction parameters we find that, as a function of a constant input concentration u, the doubly phosphorylated kinase x_{pp} at the end of the cascade exhibits a switch-like response. If matching steady-state behavior is the objective of the model, one could further collapse the equations above into a first order system that relates the input u with the output of the cascade $m = x_{pp}$:

$$\frac{dm}{dt} = \frac{\alpha u^n}{K_M^n + u^n} - m,$$

where now m indicates the concentration of doubly phosphorylated kinase, and parameters α, K, and n are chosen to capture to the observed input/output relationship.

Based on this simplified model for the double phosphorylation process of each kinase, one can assemble a naive phenomenological model for the entire cascade:

$$\begin{aligned}
\frac{dm_3}{dt} &= \alpha_3\frac{u^{n_3}}{K_{M3}^{n_3}+u^{n_3}} - m_3\\
\frac{dm_2}{dt} &= \alpha_2\frac{m_3^{n_2}}{K_{M2}^{n_2}+m_3^{n_2}} - m_2\\
\frac{dm_1}{dt} &= \alpha_1\frac{m_2^{n_1}}{K_{M1}^{n_1}+m_2^{n_1}} - m_1.
\end{aligned} \tag{1}$$

Later we will use this simplified model to illustrate control and dynamical systems theory methods to analyze its behavior. A more accurate, yet simple, model of the pathway is proposed in [29], including double phosphorylation steps for each kinase.

3 Analysis of Dynamic Behaviors

We can write the ODE model of a generic molecular process as:

$$\begin{aligned} \frac{dx}{dt} &= f(x, u), \\ x(0) &= x_0, \end{aligned} \tag{2}$$

where x is a vector in $\mathbb{R}^n$ whose components are the variables of interest in the model. In a system of molecules, these components are concentrations. Vector x describes the behavior in time of the system, and it is also called the state vector. Vector u in $\mathbb{R}^m$ represents external inputs to the system, for example concentrations of inducers or activating enzyme species. Function $f(x, u)$ captures the interactions among the chosen dynamic variables and the inputs. Finally, the problem includes a specification of initial conditions (or initial state) in the vector x_0.

Most models of biomolecular phenomena are nonlinear: thus, it is difficult (with few exceptions) to derive analytical predictions of their dynamics. The most general way to handle nonlinear systems is to analyze their dynamics in a neighborhood of their equilibrium points.

3.1 Linearization

Linearization analysis consists in approximating the behavior of a nonlinear system in a neighborhood of its equilibrium points using its linearized dynamics; a brief introduction to this technique is provided in this chapter, and the reader should refer to [30, 31] for more details.

The equilibrium points of the general dynamical system (Eq. 2) for a given value of external inputs $\overline{u}$ are defined as the states $\overline{x}$ such that $f(\overline{x}, \overline{u}) = 0$. In other words, if the system's state is precisely $\overline{x}$, all future states will remain equal to $\overline{x}$.

As a simple illustrative example, consider the differential equation:

$$\frac{dx}{dt} = ux - x^2 \tag{3}$$

If we set $\dot{x} = 0$, we find the condition $x(u - x) = 0$, which is satisfied for $\overline{x} = 0$, $\overline{x} = u$. Once the system's equilibrium has been found, we can write a Taylor series approximation for the system's dynamics near each equilibrium, stopping at the first order:

$$\begin{aligned} \frac{dx}{dt} = f(x, \overline{u}) &\approx \overbrace{f(\overline{x}, \overline{u})}^{=0} + \left.\frac{\partial f(x, u)}{\partial x}\right|_{x=\overline{x}, u=\overline{u}} (x - \overline{x}) \\ &\quad + \left.\frac{\partial f(x, u)}{\partial u}\right|_{x=\overline{x}, u=\overline{u}} (u - \overline{u}) \\ &\approx J_x(x - \overline{x}) + J_u(u - \overline{u}), \end{aligned}$$

where J_x and J_u are constant scalars or matrices that capture the differential behavior of the system near the equilibrium. This procedure is the first step of linearization. Now with a change of variable, defining $\xi = (x - \overline{x})$ and $\omega = u - \overline{u}$ we can rewrite the system as:

$$\frac{d\xi}{dt} = J_x \xi + J_u \omega,$$

which is a linear dynamical system describing the near equilibrium dynamics of the original nonlinear system.

Going back to the illustrative example at Eq. 3, where $f(x, u) = ux - x^2$ we find that $J_x = \overline{u} - 2\overline{x}$, and $J_u = \overline{x}$. Therefore, the approximated system's dynamics near each equilibrium point are:

$$\overline{x} = 0 \quad \Rightarrow \quad \frac{d\xi}{dt} = \overline{u}\xi, \quad \overline{x} = \overline{u} \quad \Rightarrow \quad \frac{d\xi}{dt} = -\overline{u}(\xi - \omega),$$

where ξ and ω are values of the state and the input near the equilibrium.

For models defined by several states and differential equations, linearization yields a linear system described by two matrices:

$$J_x = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}_{|x=\overline{x}, u=\overline{u}}, \qquad J_u = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} & \cdots & \frac{\partial f_1}{\partial u_m} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} & \cdots & \frac{\partial f_2}{\partial u_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1} & \frac{\partial f_n}{\partial u_2} & \cdots & \frac{\partial f_n}{\partial u_m} \end{bmatrix}_{|x=\overline{x}, u=\overline{u}}.$$

Matrix J_x is known as the system's Jacobian matrix. If the system of ODEs has n equations (states), the Jacobian is always an $n \times n$ matrix; J_u is an $n \times m$ matrix, where m is the number of external inputs.

Once the system has been linearized, we can investigate its local behavior with standard linear analysis methods. In particular, by finding the eigenvalues of the Jacobian we can immediately establish if the equilibrium is stable or unstable. Eigenvalues λ and eigenvectors v of a matrix A are defined by the following relationship:

$$Av = \lambda v.$$

If A is viewed as a linear map, eigenvectors represent special directions in the domain of A which remain unaltered in the codomain, except for scalar transformations. The eigenvalues of a matrix A are the roots λ of the polynomial equation:

$$\det(A - \lambda Id) = 0,$$

where $I\ d$ is the identity matrix of appropriate dimension. It is well known [31, 32] that the fundamental solution of a matrix ODE system $\dot{x} = Ax + Bu$ is determined by the matrix exponential

$\Phi = e^{At}$. (The natural response of the system, when $u = 0$, is $x(t) = e^{At}x_0$). In most practical cases, a real or complex matrix A is similar to a diagonal matrix Δ whose elements on the diagonal are the eigenvalues of A: $A = P\,\Delta\,P^{-1}$, where P is a matrix of eigenvectors associated with the eigenvalues of A. This means that we can rewrite the fundamental solution matrix $\Phi = e^{At} = Pe^{\Delta t}P^{-1}$ (the matrix exponential of a diagonal matrix is simply a diagonal matrix whose elements are the corresponding exponentials). Thus, the behavior of a linear system is given by linear combinations of exponential functions, whose convergent or divergent behavior exclusively depends on the sign of the eigenvalues. By determining the eigenvalues, and most importantly their sign, we can classify the system as stable, when all eigenvalues have a negative sign; when at most one zero eigenvalue is present, the system is classified as marginally stable; when at least one eigenvalue is positive, the system is unstable.

Finding the eigenvalues of J_x at each equilibrium allows us to build an approximate map of how the system behaves. Returning to our simple scalar example at Eq. 3:

$$\overline{x} = 0 \quad \Rightarrow \quad J_x = \overline{u}, \quad \overline{x} = \overline{u} \quad \Rightarrow \quad J_x = -\overline{u}$$

Therefore, for nonzero $\overline{u}$, the system always has one stable and one unstable equilibrium.

3.1.1 Linearization Example: The MAPK Cascade

We can carry out a linearization analysis of the MAPK cascade model (Eq. 1), choosing $\alpha_i = 1$ and $K_{Mi} = 1$ for $i = 1, 2, 3$, and $n_3 = 1$, $n_2 = n_1 = 2$:

$$\begin{aligned}\frac{dm_3}{dt} &= \frac{u}{1+u} - m_3\\ \frac{dm_2}{dt} &= \frac{m_3^2}{1+m_3^2} - m_2\\ \frac{dm_1}{dt} &= \frac{m_2^2}{1+m_2^2} - m_1.\end{aligned}$$

First, we find the equilibria by setting each derivative to zero. It is very easy to find that there is a single equilibrium where $\overline{m_3} = u/(1+u)$, $\overline{m_2} = \overline{m_3}^2/(1+\overline{m_3}^2)$, and $\overline{m_1} = \overline{m_2}^2/(1+\overline{m_2}^2)$. The Jacobian of the system is:

$$J_x = \begin{bmatrix} -1 & 0 & 0 \\ \alpha & -1 & 0 \\ 0 & \beta & -1 \end{bmatrix}, \tag{4}$$

where $\alpha = \frac{2\overline{m_3}}{(1+\overline{m_3}^2)^2}$ and $\beta = \frac{2\overline{m_2}}{(1+\overline{m_2}^2)^2}$. The eigenvalues can be read directly on the diagonal of J_x, because it is a lower triangular matrix. We find $\lambda_1 = \lambda_2 = \lambda_3 = -1$. Therefore, this system is stable near its single equilibrium point. Given the structure of the Jacobian and of the ODEs, the equilibrium is stable regardless of the choice of parameters made. Therefore, this simplified model of the cascade suggests that its stable dynamic behavior is robust with respect to uncertainty in the parameters. We will later see that if the cascade includes additional interactions among the kinases, which generate feedback loops, we will not be able to reach the same conclusion.

3.1.2 Phase Portraits

Phase portraits are extremely useful graphical representations, in particular for models of low dimensions. The solution trajectories are parameterized over time, and plotted contrasting different components [31]. These graphs can be quickly traced qualitatively, and numerous numerical routines are available for quantitative plots (see, for example, MATLAB's `pplane` function).

For illustrative purposes, one typically considers second order linear systems, such as:

$$\begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix} = A \begin{bmatrix} x \\ y \end{bmatrix}.$$

The system's phase portrait simply consists in a plot where $x(t)$ is graphed versus $y(t)$ on a plane. This plot can be quickly sketched by identifying the eigenvalues and eigenvectors of matrix A. The real part of the eigenvalues determines whether trajectories converge toward the origin as an equilibrium point (negative real part, equilibrium is stable) or diverge (positive real part, equilibrium is unstable); real eigenvectors define invariant subspaces on which the behavior of the trajectory is uniquely determined by the associated eigenvalue. Figure 1 shows typical examples of two-dimensional phase portraits, such as sinks (a), sources (b), and hyperbolic points (c). When eigenvalues are complex conjugates, trajectories spiral in or out of the origin depending on the sign of the real part (Fig. 1d,e); if the real part is zero, the system is classified as a center, i.e. a system whose trajectories oscillate without damping (Fig. 1f).

The only equilibrium of the MAPK pathway model considered in Subheading 3.1.1 is a sink (Fig. 1a), because all the eigenvalues of the Jacobian are real and have negative real part.

3.2 Bifurcations

If one or more parameters of the system vary, the number of equilibria and their local stability properties may change. In a biological network binding rates may vary as a function of environmental stimuli and result in dramatically different dynamic responses: for instance, there is evidence that MAPK pathway response can exhibit a variety of responses depending on the input hormones [33], which affect binding affinities of its components. The pathway is known to exhibit a multistationary (multiple equilibria either stable or unstable) or oscillatory responses [34]. The variation of one or more

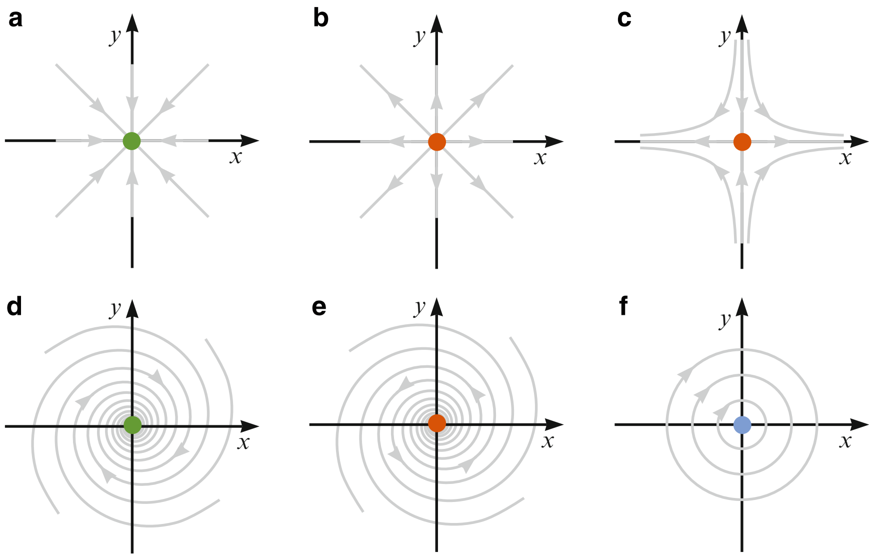


Fig. 1 Two-dimensional phase portraits. (**a**) Sink (all eigenvalues have negative real part). (**b**) Source (all eigenvalues have positive real part). (**c**) Hyperbolic point (one eigenvalue has positive real part, one eigenvalue has negative real part; eigenvectors coincide with the axes). (**d**) Stable spiral sink (eigenvalues are complex conjugate and have negative real part). (**e**) Unstable spiral sink (eigenvalues are complex conjugate and have positive real part). (**f**) Center (eigenvalues are pure imaginary, the system is marginally stable)

parameters followed by a change in dynamics is generally termed a bifurcation phenomenon. Classical examples are the saddle-node and the Hopf bifurcation [31]. Subheading 4.1 provides examples of bifurcations in a biological network with different types of feedback.

4 Feedback in Synthetic Biological Networks

Gene networks rely on feedback to regulate expression of proteins, reduce noise, and guarantee desired dynamic behaviors [35–38]. The target behavior of engineered networks depends as critically on the use of feedback: in this section we provide several examples of networks where the design of positive or negative feedback allows to achieve dramatically different behaviors. We begin with a general two-gene model which has been used to describe a variety of simple synthetic and natural networks; we show that in some cases the feedback topology is the key player in determining the dynamic outcomes of the system [7, 39]. Then, we highlight the effects of synthetic feedback loops on a model for the MAPK signaling pathway in yeast, which has recently been engineered to re-route mating behaviors [25].

4.1 Feedback Loops Reshape the Dynamic Behavior of a System

We consider a standard model for transcription and translation of two genes, where proteins reciprocally modulate their expression forming a feedback loop. Similar models are commonly encountered in the literature (see, for instance, [19, 22]). For illustrative purposes, we use a nondimensional model (*see* **Note 1**):

$$\dot{r}_1 = \gamma_1 + H_1(p_2) - r_1, \qquad \dot{p}_1 = \beta_1 r_1 - p_1, \tag{5a}$$

$$\dot{r}_2 = \gamma_2 + H_2(p_1) - r_2, \qquad \dot{p}_2 = \beta_2 r_2 - p_2, \tag{5b}$$

where, for $i = 1, 2$, r_i are RNA species concentrations; p_i are protein concentrations; $H_i(\cdot)$ are Hill functions, and all Greek letters denote reaction rates that are positive scalars.

Depending on the regulatory action and feedback created by the protein transcription factors, and thus depending on the type of Hill function, the network presents a different number of equilibria and different possible dynamic behaviors. For example, suppose $H_1(p_2) = \alpha_1 \frac{p_2^n}{1+p_2^n}$ and $H_2(p_1) = \alpha_2 \frac{p_1^n}{1+p_1^n}$: this is a two-gene positive feedback loop, which is often encountered in developmental networks [40, 41]. The Jacobian of the system is:

$$J = \begin{bmatrix} -1 & 0 & 0 & \frac{\partial H_1}{\partial p_2} \\ \beta_1 & -1 & 0 & 0 \\ 0 & \frac{\partial H_2}{\partial p_1} & -1 & 0 \\ 0 & 0 & \beta_2 & -1 \end{bmatrix}, \tag{6}$$

where $\partial H_i/\partial p_j = \alpha_i \frac{n p_j^{n-1}}{(1+p_j^n)^2}$, $(i,j) \in \{(1,2),(2,1)\}$. Note that the Jacobian entries, evaluated at a positive equilibrium, are sign definite, i.e. they do not change sign for arbitrary choices of the (positive) parameters α_i, β_i, and n.

The Jacobian sign pattern is thus a "structural" property of this system, and it can be associated with a graph: nodes correspond to the concentrations of biochemical species and are interconnected by positive (+1) or negative (−1) arcs according to the corresponding Jacobian entries, as shown in Fig. 2a. Thus, the positive or negative sign of the loops generated does not depend on the specific choice of the parameters.

We can derive expressions for the equilibria of the system, which are given by the intersections of the two equilibrium conditions (Fig. 2a, top row):

$$p_1 = \beta_1\left(\gamma_1 + \alpha_1 \frac{p_2^n}{1+p_2^n}\right), \quad p_2 = \beta_2\left(\gamma_2 + \alpha_2 \frac{p_1^n}{1+p_1^n}\right).$$

For $n = 1$ there is an intersection with p_1 and p_2 positive. For $n > 1$, the system may admit multiple, typically three, positive equilibria. For an assigned value of n, we consider one equilibrium and we evaluate its stability properties by finding the eigenvalues of the Jacobian, which are the roots of its characteristic polynomial

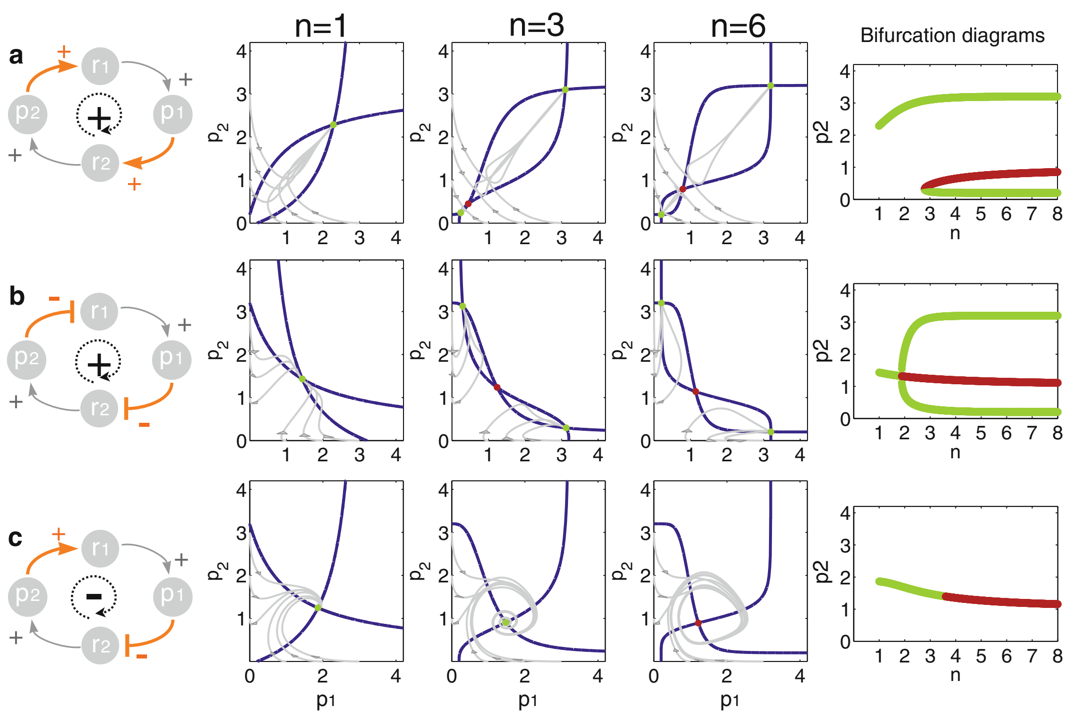


Fig. 2 Feedback loops in two-gene systems. (**a**) Two-gene system with double positive feedback loop (positive cycle). *Pointed arrowheads* indicate positive Jacobian interconnection entries, while *hammer-arrowheads* indicate negative interconnections. (**b**) Two-gene system with double negative feedback loop, resulting in an overall positive cycle. (**c**) Two-gene feedback interconnection with positive and negative regulation, resulting in an overall negative cycle. In all simulations (nullclines in *blue*, sample trajectories in *gray*) the nondimensional parameters are chosen as $\gamma_1 = \gamma_2 = 0.2$, $\alpha_1 = \alpha_2 = 3$, $\beta_1 = \beta_2 = 1$ and n is varied. The *right column* shows the corresponding value of p_2 equilibria for varying n, and their different pattern of transition to instability (*green dots* are stable, *red dots* are unstable equilibria)

$$(s+1)^4 - K = 0, \quad \text{where} \quad K = \beta_1\beta_2 \frac{\partial H_1}{\partial p_2}\frac{\partial H_2}{\partial p_1} > 0. \qquad (7)$$

Note that K explicitly depends on parameters n, α_i, and β_i and on the equilibria for p_1 and p_2 (which are in turn a function of all the parameters of the ODEs). The roots of Eq. 7 are:

$$s = -1 + q, \ -1 - q, \ -1 + iq, \ -1 - iq, \quad \text{where} \quad q = K^{\frac{1}{4}}.$$

If $K > 1$, there is only one root having positive real part, and it is real. If $K < 1$, all of the roots have negative real part. Thus, the system can only admit real exponential instability, i.e. instability arising due to a real eigenvalue changing sign from negative to positive. Figure 2a, top row, shows equilibrium conditions and example trajectories in the $p_1 - p_2$ plane of the phase space for different values of n (stable equilibria are represented as green circles, unstable equilibria as red circles). The last column in Fig. 2 shows the evolution of the number and stability properties of equilibria for p_2 as n varies (note that other parameters could have picked to study the presence of bifurcations).

If $H_1 = \alpha_1 \frac{1}{1+p_2^n}$, $H_2 = \alpha_2 \frac{1}{1+p_1^n}$, network (Eq. 5) specifies a two-gene double negative feedback loop, depicted in Fig. 2b, left. This circuit is also known as toggle switch, an example of which is the famous synthetic biological circuit by Gardner [15]; a natural example of a toggle switch is the Cdc2-Wee1 network considered, for instance, in [4]. We can repeat the same analysis performed for the two-gene double positive feedback loop, and get similar results in terms of admissible transitions to instability, which can be only real exponential, regardless of the considered equilibrium (Fig. 2b).

We now compare the previous two examples to the case when Hill functions have opposite regulatory roles, i.e. $H_1 = \alpha_1 \frac{p_2^n}{1+p_2^n}$ an $H_2 = \alpha_2 \frac{1}{1+p_1^n}$: the network can behave as a two-gene oscillator [22]. First, we observe that the Jacobian is still a sign definite matrix. However, the "interconnection" terms $\partial H_1/\partial p_2$ and $\partial H_2/\partial p_1$, the derivatives of the Hill functions, now have opposite signs, due to the different slopes of such functions, and thus generate an overall *negative* feedback loop (Fig. 2c). The equilibrium conditions are now

$$p_1 = \beta_1\left(\gamma_1 + \alpha_1 \frac{p_2^n}{1+p_2^n}\right), \quad p_2 = \beta_2\left(\gamma_2 + \alpha_2 \frac{1}{1+p_1^n}\right),$$

and admit a single intersection regardless of the value of α_i, β_i, and n (Fig. 2c, central panels show the equilibrium conditions for increasing values of n). The characteristic polynomial is

$$(s+1)^4 - K = 0, \quad \text{where} \quad K = \beta_1\beta_2 \frac{\partial H_1}{\partial p_2}\frac{\partial H_2}{\partial p_1} < 0. \tag{8}$$

Since now $K < 0$, all of the roots of Eq. 8 are complex:

$$s = (-1+q)+iq,\ (-1-q)-iq,\ (-1-q)+iq,\ (-1+q)-iq,$$
$$\text{where} \quad q = \frac{(-K)^{\frac{1}{4}}}{\sqrt{2}}.$$

As a consequence, only oscillatory unstable dynamics can arise, rather than real exponential. Precisely, unstable oscillations do arise when $K < -4$. As we can see by studying the original nonlinear system, for any given value of n there is only one equilibrium, whose stability properties can change, again, depending on the values of α_i and β_i.

To summarize, the analysis of this simple two-gene system has shown that, without a precise knowledge of the functions H_1 and H_2, we can reach very strong conclusions regarding the possible dynamic behaviors of the system. These conclusions are consistent with Thomas' conjectures [7], and do not depend on specific functions or parameter choices. Rather, they depend on the presence of positive or negative feedback interconnection among components, thus on the presence of a positive or a negative cycle in the Jacobian associated with the system. In particular, this example clearly highlights that there is a relationship between Jacobian cycles and admissible transitions to instability. A qualitatively similar study was carried out and validated by building synthetic bacterial circuits in [16]; analysis relied on the S-systems formalism [42].

4.2 Synthetic Feedback in the MAPK Pathway

The dynamic profile of gene expression coordinates spatio-temporal processes in organisms. At the single-cell level, dynamics of signaling components can dictate the dynamics of gene expression and control cellular entry into divergent cell fates. The MAPK pathway in PC-12 cells provides a classic example of signaling dynamics regulating cellular fate. In PC-12 cells, unique extracellular cues alter the MAPK network topology by inducing positive or negative feedback loops leading to differential temporal profiles of MAPK activation. Each temporal profile maps to a distinct and divergent cellular behavior [33]. Synthetic switching of these topologies alters the dynamic profile and routes cells to the alternative fate, suggesting that control of network topology and thus signaling dynamics controls cellular fates (e.g., differentiation, division, and apoptosis). Given the importance of cellular fate in fields such as stem cell biology and cancer biology, synthetic circuits that can control dynamic signaling and thus cell fate may provide useful research tools as well as potential therapies.

Synthetic reshaping of dynamic signaling profiles in a MAPK pathway has allowed the construction of pulse generators, accelerators, delays, ultrasensitive responses, and bistable switches [43, 44]. Construction of positive feedback loops that induce bistability in a MAPK pathway was shown to be dependent on feedback strength [44]. Additionally, components within the MAPK pathway can be tuned to allow for the existence of bistability [45].

The yeast pheromone-responsive pathway is a canonical MAPK pathway with a three-tiered MAPK cascade. Due to the genetic tractability of yeast relative to mammalian systems, the pheromone-responsive pathway, also called the mating pathway, has been extensively analyzed experimentally and modeled computationally. Signaling in the mating pathway is initiated by pheromone alpha-factor (α), binding to a transmembrane receptor which initiates G-protein signaling and a phosphorylation cascade from Ste4 to the canonical scaffold-bound three-tiered MAPK cascade. As the output of the cascade, the phosphorylated MAPK Fus3 translocates the nucleus, activating transcription factors and transcription at mating-responsive genes including the $Fus1$ locus. The phosphatase Msg5 antagonizes (inhibits) signaling by dephosphorylating Fus3. Activation of mating genes induces cell-cycle arrest, polarized

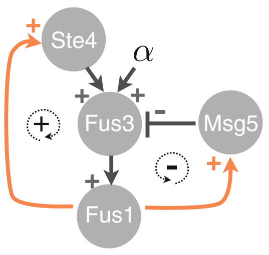


Fig. 3 Scheme of the engineered MAPK pathway in Eqs. 9–12. This scheme can also be seen as a graph representing the sign-definite Jacobian. *Orange arrows* indicate the synthetic feedback loops

cell growth, and fusion of haploid cells to form diploids with opposite mating-type cells.

To synthetically rewire the topology of the mating pathway, positive and negative feedback loops were constructed around the native pathway in [25]. To construct feedback loops (Fig. 3), a pathway-responsive promoter was cloned from the *Fus*1 locus into plasmids. Ste4 overexpression was shown to initiate pathway activation, positively regulating pathway activity. Thus, a positive feedback loop was constructed by placing Ste4 under the regulation of the *Fus*1 promoter. Conversely, overexpression of Msg5 attenuated signaling, negatively regulating pathway activation. Pairing Msg5 with the *Fus*1 promoter constructed a negative feedback loop. Pairing positive and negative feedback constructs with RNA-based transducers of varying activity generated constructs with a range of feedback strengths. Experimentally, the strength of positive feedback was shown to dictate the pathway sensitivity to activation. Similarly, the strength of negative feedback correlated with the degree of pathway attenuation.

To mathematically capture insights into the synthetically wired system, a phenomenogical model of the MAPK pathway with synthetic feedback was also constructed [25]:

$$\frac{dSte4}{dt} = \beta_{Ste4} - \delta_{Ste4} Ste4 + \boxed{k_{pf} \frac{Fus1^{n_{pf}}}{K_{M,Fus1,pf}^{n_{pf}} + Fus1^{n_{pf}}}}, \tag{9}$$

$$\frac{dFus3}{dt} = \beta_{Fus3} - \delta_{Fus3} Fus3 + k_{\alpha} \frac{\alpha^n}{K_{M,\alpha}^n + \alpha^n} + k_{Ste4} \frac{Ste4^m}{K_{M,Ste4}^m + Ste4^m} - k_{Msg5} Fus3 \frac{Msg5^q}{K_{M,Msg5}^q + Msg5^q}, \tag{10}$$

$$\frac{dFus1}{dt} = \beta_{Fus1} - \delta_{Fus1} Fus1 + k_{Fus3} \frac{Fus3^p}{K_{M,Fus3}^p + Fus3^p}, \tag{11}$$

$$\frac{dMsg5}{dt} = \beta_{Msg5} - \delta_{Msg5} Msg5 + \boxed{k_{nf}\frac{Fus1^{n_{nf}}}{K_{M,Fus1,nf}^{n_{nf}} + Fus1^{n_{nf}}}}. \tag{12}$$

Terms corresponding to the engineered positive and negative feedback loops are highlighted inside boxes. It is an easy exercise to compute the system's Jacobian matrix; this matrix is sign definite, meaning that the sign of each entry does not depend on the parameters chosen. Because of sign definiteness, the scheme in Fig. 3 can be also used as a "graph" representation of the Jacobian matrix similar to those obtained in Fig. 2.

In the rest of this section we consider the cases where the system is added exclusively one feedback loop, positive in the first case (activation of Ste4 by Fus1), negative in the second (activation of Msg5 by Fus1). With numerical simulations we will highlight the potential for bistability of the system with positive feedback, and of oscillations in the system with negative feedback. Unless otherwise noted, we use the same parameters used in [25], which were fitted to experimental data (*see* Fig. 6).

4.2.1 A Synthetic Positive Feedback Loop Can Yield Bistability

If we add exclusively a positive feedback loop to the system, Msg5 can be seen as an input (possibly constant or slowly varying) to the main pathway (a scheme is in Fig. 4a). In the following, we indicate the Msg5 input as u. We also assume that the inducer α is absent. Thus, our equations reduce to:

$$\frac{dSte4}{dt} = \beta_{Ste4} - \delta_{Ste4} Ste4 + \boxed{k_{pf}\frac{Fus1^{n_{pf}}}{K_{M,Fus1,pf}^{n_{pf}} + Fus1^{n_{pf}}}}, \tag{13}$$

$$\frac{dFus3}{dt} = \beta_{Fus3} - \delta_{Fus3} Fus3 + k_{Ste4}\frac{Ste4^m}{K_{M,Ste4}^m + Ste4^m} - k_u Fus3 \frac{u^q}{K_{M,u}^q + u^q}, \tag{14}$$

$$\frac{dFus1}{dt} = \beta_{Fus1} - \delta_{Fus1} Fus1 + k_{Fus3}\frac{Fus3^p}{K_{M,Fus3}^p + Fus3^p}, \tag{15}$$

where again we highlight with a box the term introducing positive feedback in the network. In the absence of α factor, the network output self-activates to a high value due to the presence of feedback. This behavior is showed in the numerical simulations in Fig. 4: increasing values of rate k_{pf} results in stronger self-activation of the pathway.

To explore the potential for bistability, as done in the previous section we can find equilibrium conditions for $Ste4$ and $Fus1$:

$$\overline{Ste4} = \frac{1}{\delta_{Ste4}}\left\{\beta_{Ste4} k_{pf}\frac{\overline{Fus1}^{n_{pf}}}{K_{M,Fus1,pf}^{n_{pf}} + \overline{Fus1}^{n_{pf}}}\right\} \tag{16}$$

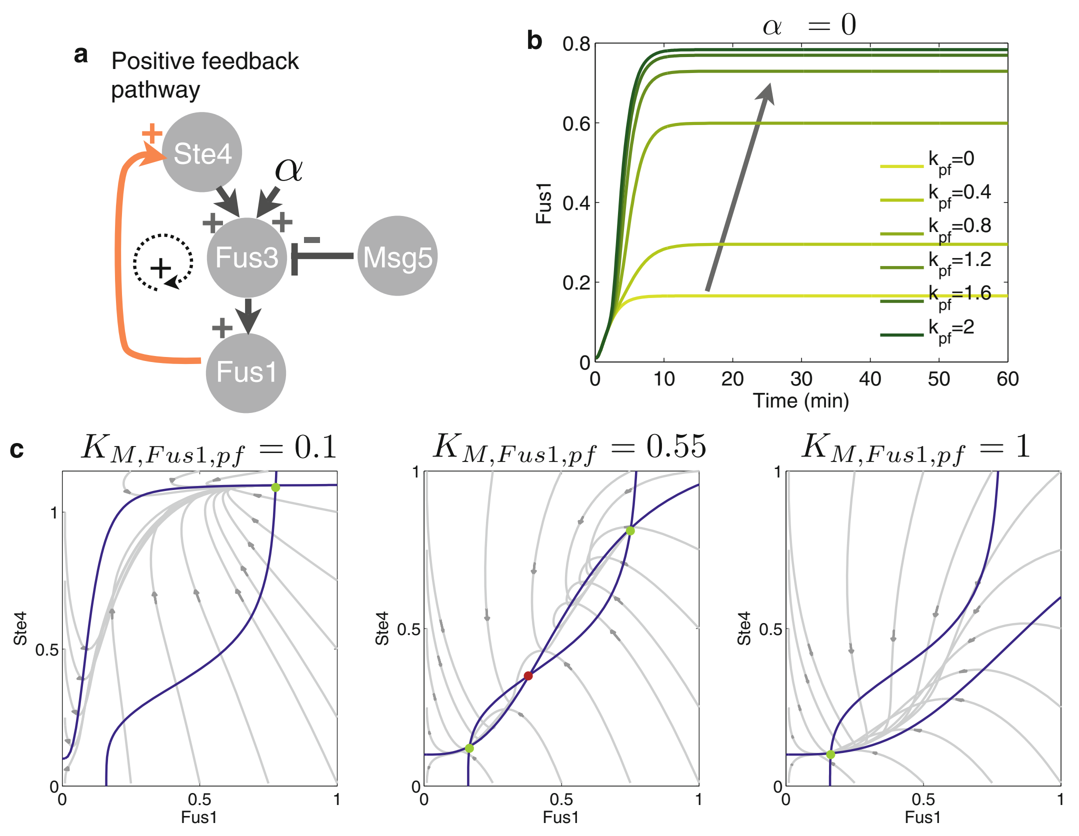


Fig. 4 Positive feedback pathway analysis. (**a**) Scheme of the network with positive feedback. (**b**) Concentration of Fus1 as a function of time, for different values of k_{pf}. (**c**) Equilibrium conditions (*blue*), their intersections (*green*, stable points; *red*, unstable points), and sample trajectories (*gray*) in the plane Fus1-Ste4. Bistability can be achieved when $K_{M,\ Fus1,\ pf}$ has values around 0. 5–0. 7

$$\overline{Fus3} = \frac{1}{\delta_{Fus3} + k_u \frac{u^q}{K^q_{M,u}+u^q}} \left\{ \beta_{Fus3} + k_{Ste4} \frac{\overline{Ste4}^m}{K^m_{M,Ste4} + \overline{Ste4}^m} \right\} \quad (17)$$

$$\overline{Fus1} = \frac{1}{\delta_{Fus1}} \left\{ \beta_{Fus1} + k_{Fus3} \frac{\overline{Fus3}^p}{K^p_{M,Fus3} + \overline{Fus3}^p} \right\}. \quad (18)$$

These equilibrium conditions depend on several parameters, each affecting the number and stability properties of the admissible equilibria. We focus our attention on the engineered reactions creating the positive feedback loop. We find that parameter $K_{M,\ Fus1,\ pf}$ is particularly important to achieve bistability; this parameter represents the half-max activation value of Ste4 by Fus1. The corresponding Hill coefficient is equal to 3, making the half-max value act as an activation threshold for Ste4. Equilibrium conditions Eqs. 16 and 18 are plotted in Fig. 4 for different values of $K_{M,\ Fus1,\ pf}$ ($k_{pf} = 2$). For the chosen parameter set, bistability is achieved only within a range of values of $K_{M,\ Fus1,\ pf}$.

4.2.2 A Synthetic Negative Feedback Loop Has the Potential to Yield Oscillations

We now numerically simulate the pathway in the presence of an engineered negative feedback loop only. In this case, Ste4 can be considered an external input, which we now call w. We assume that α factor is present. The ODEs are:

$$\frac{dFus3}{dt} = \beta_{Fus3} - \delta_{Fus3}Fus3 + k_\alpha \frac{\alpha^n}{K^n_{M,\alpha} + \alpha^n} + k_w \frac{w^m}{K^m_{M,w} + w^m} - k_{Msg5}Fus3 \frac{Msg5^q}{K^q_{M,Msg5} + Msg5^q}, \tag{19}$$

$$\frac{dFus1}{dt} = \beta_{Fus1} - \delta_{Fus1}Fus1 + k_{Fus3}\frac{Fus3^p}{K^p_{M,Fus3} + Fus3^p}, \tag{20}$$

$$\frac{dMsg5}{dt} = \beta_{Msg5} - \delta_{Msg5}Msg5 + \boxed{k_{nf}\frac{Fus1^{n_{nf}}}{K^{n_{nf}}_{M,Fus1,nf} + Fus1^{n_{nf}}}}. \tag{21}$$

The box highlights the negative feedback term; Fig. 5a shows the topology of this pathway, which corresponds to the sign pattern

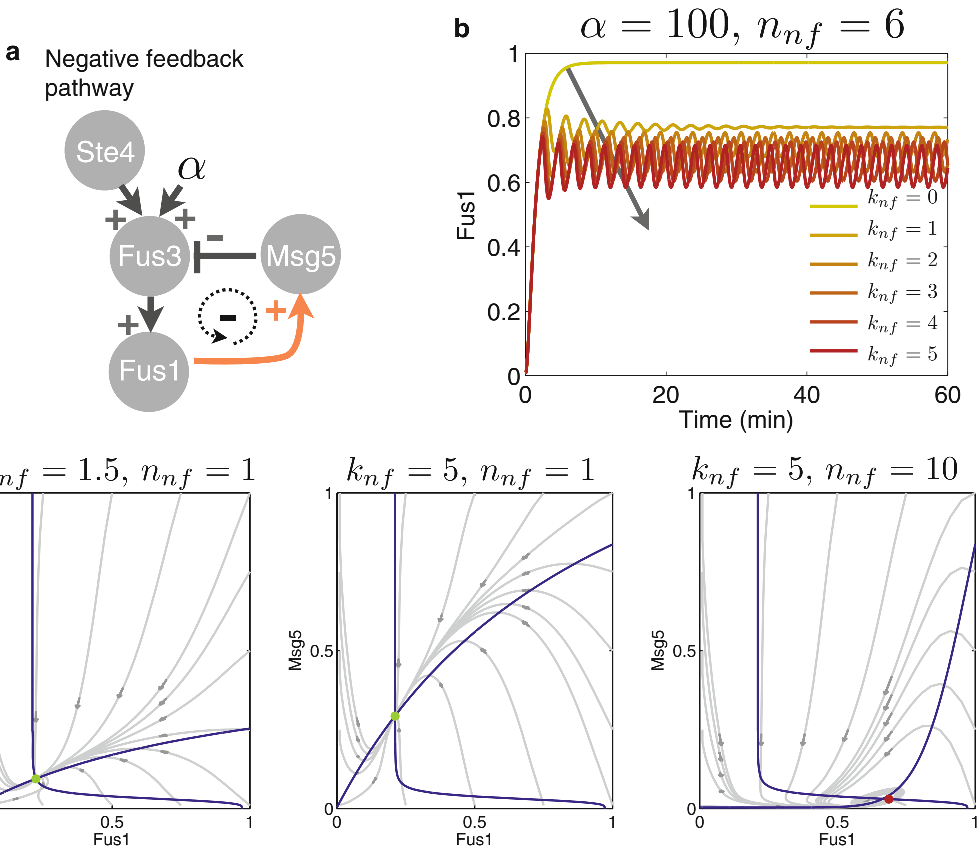


Fig. 5 Negative feedback pathway analysis. (**a**) Scheme of the network with engineered negative feedback. (**b**) Concentration of Fus3 as a function of time, for different values of k_{nf}. (**c**) Equilibrium conditions (*blue*), their intersections (*green*, stable points; *red*, unstable points), and sample trajectories (*gray*) in the plane Fus1-Msg5. Local oscillations can be achieved for high values of k_{nf} and n_{nf}. (Color figure online)

of the Jacobian matrix. Equilibrium conditions can be derived as done for the positive feedback pathway at Eqs. 16–18. To explore the potential for oscillations in the presence of negative feedback, we focus again on the parameters of the engineered reaction controlling Msg5 as a function of the output Fus1. Equilibrium conditions always intersect at one individual point, as shown in Fig. 5c. We find that for increasing values of both the rate k_{nf} and the Hill coefficient n_{nf}, the single equilibrium becomes unstable, with complex conjugate eigenvalues which correspond to local oscillations. The behavior in time of the output Fus1 is shown in Fig. 5b for a range of values of k_{nf}, and high $n_{nf} = 6$. For the chosen parameter set, our numerical analysis reveals that an extremely high value of k_{nf} and n_{nf} (experimentally not achievable) has the potential to yield oscillations, however their amplitude is limited and their frequency is very high. This means that experimentally it would be difficult to achieve oscillations in this particular synthetic pathway. More systematic exploration of the system's parameter space may reveal the existence of operating conditions that can yield more realistic oscillations.

5 Conclusions

In this chapter we have provided a general overview of the role of feedback in molecular networks. We have introduced simple yet powerful methods commonly used in dynamical systems and control theory to identify the behavior of a nonlinear dynamical system around its equilibria. Feedback loops dramatically affect the possible dynamic outcomes of a system: we showed that in some cases such outcomes may be determined exclusively by the type of feedback (positive or negative) present in the network, regardless of the parameters. We address the reader to [39] for further analysis on this topic. In some cases, parameters responsible for a bifurcation can be easily identified and tuned to achieve the desired behavior. These ideas have been largely exploited in the design of synthetic gene networks in the last decade [15, 16, 19, 20].

Throughout the chapter, we also used the MAPK pathway as an example of a system that can be successfully modeled with ODEs [5, 6, 25, 29] and lends itself well to the linearization analysis we presented. We focused on a recently engineered MAPK pathway in yeast, where positive and negative feedback loops were engineered using inducible promoters and RNA transducers [25]. Through numerical simulations, we showed that positive feedback can yield bistability and negative feedback can yield oscillations. We leave it as an exercise to the reader to verify the exclusive potential for bistability or oscillations in the two engineered versions of the network [39], following the steps outlined in Subheading 4.1 (Fig. 6).

Species	Description	β (production)	LB	UB
Fus1	Promoter activity, pathway output	0.001	1.00E-03	2
Fus3	MAPK, central hub in pathway	0.1	1.00E-03	1
Msg5	Native Msg5, negative regulator	0.01	1.00E-03	1
Ste4	Native Ste4, positive regulator	0.2	1.00E-04	0.5

Species	Description	δ (degadation)	LB	UB
Fus1	Promoter activity, pathway output	1	1.00E-03	3
Fus3	MAPK, central hub in pathway	0.7	1.00E-03	1
Msg5	Native Msg5, negative regulator	3	1.00E-03	5
Ste4	Native Ste4, positive regulator	2	1.00E-03	5

Rates	Description	Value	LB	UB
k_{α}	Rate constant for α-factor activation of Fus3	2	5.00E-01	4
k_{Fus3}	Rate constant for Fus3 activation of Fus1	1.3	1.00E-03	2
k_{Ste4}	Rate constant for Ste4 activation of Fus3	1	1.00E-03	5
k_{Msg5}	Rate constant for Msg5 desphosphorylation of	10	1.00E-03	15
$k_{c,Ste4}$	Rate constant for production of Ste4 from boo	2	1.00E-03	5
$k_{FB,Ste4}$	Rate constant for production of Ste4 from pos	2	1.00E-03	5
$k_{c,Msg5}$	Rate constant for production of Msg5 from res	2.5	1.00E-03	5
$k_{FB,Msg5}$	Rate constant for production of Msg5 from ne	1.5	1.00E-03	5

Hill coeff	Description	Value	LB	UB
n	Hill coefficient for α-factor effect on Fus3	2	1	5
n_{nf}	Hill coefficient for negative feedback loop, Fu	1	1	5
n_{pf}	Hill coefficient for positive feedback loop, Fus	3	1	5
m	Hill coefficient for Ste4 activation of Fus3	4	1	5
q	Hill coefficient for Msg5 deactivation of Fus3	4	1	5
p	Hill coefficient for Fus3 activation of Fus1	1	1	5

K_M	Description	Value	LB	UB
$K_{M,\alpha}$	Half-maximal concentration of α-factor (phero	14	10	40
$K_{M,Ste4}$	Half-maximal concentration of Ste4	0.5	1E-03	5
$K_{M,Msg5}$	Half-maximal concentration of Msg5	0.05	1E-03	5
$K_{M,Fus3}$	Half-maximal concentration of Fus3	1	1E-03	2
$K_{M,Fus1,nf}$	Half-maximal concentration of Fus1 for negativ	1	1E-03	5
$K_{M,Fus1,pf}$	Half-maximal concentration of Fus1 for positiv	0.1	1E-03	5

Fig. 6 Parameters for the MAPK pathway model with *upper* (UB) and *lower* (LB) bounds for fitting

6 Notes

1. We will carry out the nondimensionalization procedure for the toggle switch network, leaving the derivation for the other cases to the reader. We follow nondimensionalization steps similar to those proposed in [19] and [22, 24]. Consider the (dimensional) model:

$$\tau \dot{R}_1 = c_1 + a_1 \frac{1}{K_{M1}^n + P_2^n} - R_1, \qquad \dot{P}_1 = k_p R_1 - k_d P_1, \quad (22a)$$

$$\tau \dot{R}_2 = c_2 + a_2 \frac{1}{K_{M2}^n + P_1^n} - R_2, \qquad \dot{P}_2 = k_p R_2 - k_d P_2. \quad (22b)$$

Here c_i is the "leak" transcription of RNA. For simplicity, we assume that the translation and degradation rates for the proteins are the same. Constant τ is the mRNA half-life in the system. Constants K_{Mi} represent the number of proteins necessary to half-maximally repress R_i. Finally, assume the translation efficiency of each RNA species is given by $\bar{p}_i$, which corresponds to the average number of proteins produced by a single RNA molecule.

We define the nondimensional variables: $r_i = R_i/\bar{p}_i$, $p_i = P_i/K_{Mj}$, $(i, j) \in \{(1, 2), (2, 1)\}$. We rescale time as $\tilde{t} = t/\tau$, and also define the nondimensional parameters:

$$\gamma_i = \frac{c_i}{\bar{p}_i}, \quad \alpha_i = \frac{a_i}{\bar{p}_i K_{Mi}^n}, \quad \beta_i = \frac{k_p \bar{p}_i}{k_d K_{Mj}}, \quad T = \frac{1}{\tau k_d}.$$

The resulting nondimensional equations are:

$$\dot{r}_1 = \gamma_1 + \alpha_1 \frac{1}{1 + p_2^n} - r_1, \qquad T\dot{p}_1 = \beta_1 r_1 - p_1, \quad (23a)$$

$$\dot{r}_2 = \gamma_2 + \alpha_2 \frac{1}{1 + p_1^n} - r_2, \qquad T\dot{p}_2 = \beta_2 r_2 - p_2, \quad (23b)$$

Finally, if we assume $T \approx 1$, we get a system in the same form as Eq. 5.

References

1. Tyson JJ, Chen KC, Novak B (2003) Sniffers, buzzers, toggles and blinkers: dynamics of regulatory and signaling pathways in the cell. Curr Opin Cell Biol 15(2):221–231
2. Hasty J, McMillen D, Collins JJ (2002) Engineered gene circuits. Nature 420:224–230
3. De Jong H (2002) Modeling and simulation of genetic regulatory systems: a literature review. J Comput Biol 9:67–103
4. Angeli D, Sontag E (2003) Monotone control systems. IEEE Trans Autom Control 48 (10):1684–1698
5. Huang C-YF, Ferrell JE (1996) Ultrasensitivity in the mitogen-activated protein kinase cascade. Proc Natl Acad Sci U S A 93:10078–10083
6. Asthagiri AR, Lauffenburger DA (2001) A computational study of feedback effects on signal dynamics in a mitogen-activated protein

kinase (MAPK) pathway model. Biotechnol Prog 17:227–239

7. Thomas R (1981) On the relation between the logical structure of systems and their ability to generate multiple steady states or sustained oscillations. In: Dora J, Demongeot J, Lacolle B (eds) Numerical methods in the study of critical phenomena. Springer series in synergetics, vol 9. Springer, Berlin/Heidelberg, pp 180–193
8. Domijan M, Pécou E (2011) The interaction graph structure of mass-action reaction networks. J Math Biol 51(8):1–28
9. Gouze J-L (1998) Positive and negative circuits in dynamical systems. J Biol Syst 6:11–15
10. Snoussi E (1998) Necessary conditions for multistationarity and stable periodicity. J Biol Syst 6:3–9
11. Banaji M, Craciun G (2009) Graph-theoretic approaches to injectivity and multiple equilibria in systems of interacting elements. Commun Math Sci 7(4):867–900
12. Kaufman M, Soule C, Thomas R (2007) A new necessary condition on interaction graphs for multistationarity. J Theor Biol 248 (4):675–685
13. Richard A, Comet J-P (2011) Stable periodicity and negative circuits in differential systems. J Math Biol 63(3):593–600
14. Soulé C (2004) Graphic requirements for multistationarity. ComPlexUs 1(3):123–133
15. Gardner TS, Cantor CR, Collins JJ (2000) Construction of a genetic toggle switch in *Escherichia coli*. Nature 403:339–342
16. Atkinson MR, Savageau M, Myers J, Ninfa A (2003) Development of genetic circuitry exhibiting toggle switch or oscillatory behavior in *Escherichia coli*. Cell 113:597–607
17. Kim J, White KS, Winfree E (2006) Construction of an in vitro bistable circuit from synthetic transcriptional switches. Mol Syst Biol 68
18. Padirac A, Fujii T, Rondelez Y (2012) Bottom-up construction of in vitro switchable memories. Proc Natl Acad Sci 109(47): E3212–E3220
19. Elowitz MB, Leibler S (2000) A synthetic oscillatory network of transcriptional regulators. Nature 403:335–338
20. Stricker J, Cookson S, Bennett MR, Mather WH, Tsimring LS, Hasty J (2008) A fast, robust and tunable synthetic gene oscillator. Nature 456:516–519
21. Tigges M, Marquez-Lago TT, Stelling J, Fussenegger M (2009) A tunable synthetic mammalian oscillator. Nature 457:309–312
22. Kim J, Winfree E (2011) Synthetic *in vitro* transcriptional oscillators. Mol Syst Biol 7:465
23. Montagne K, Plasson R, Sakai Y, Fujii T, Rondelez Y (2011) Programming an in vitro DNA oscillator using a molecular networking strategy. Mol Syst Biol 7
24. Franco E, Friedrichs E, Kim J, Jungmann R, Murray R, Winfree E, Simmel FC (2011) Timing molecular motion and production with a synthetic transcriptional clock. Proc. Natl Acad Sci 108(40):E784–E793
25. Galloway KE, Franco E, Smolke CD (2013) Dynamically reshaping signaling networks to program cell fate via genetic controllers. Science 341(6152):1235005
26. Tomita M, Hashimoto K, Takahashi K, Shimizu TS, Matsuzaki Y, Miyoshi F, Saito K, Tanida S, Yugi K, Venter JC, et al. (1999) E-CELL: software environment for whole-cell simulation. Bioinformatics 15(1):72–84
27. Hoops S, Sahle S, Gauges R, Lee C, Pahle J, Simus N, Singhal M, Xu L, Mendes P, Kummer U (2006) COPASIÑa complex pathway simulator. Bioinformatics 22(24):3067–3074
28. Kholodenko BN (2000) Negative feedback and ultrasensitivity can bring about oscillations in the mitogen-activated protein kinase cascades. Eur J Biochem 267(6):1583–1588
29. Angeli D, Ferrell JE, Sontag ED (2004) Detection of multistability, bifurcations, and hysteresis in a large class of biological positive-feedback systems. Proc Natl Acad Sci U S A 101(7):1822–1827
30. Khalil HK (2002) Nonlinear systems. Pearson Higher Education, Harlow
31. Perko L, (1991) Differential equations and dynamical systems. Springer, New York
32. Å strom KJ, Murray RM (2009) Feedback systems: an introduction for scientists and engineers. Princeton University Press, Princeton
33. Santos SDM, Verveer PJ, Bastiaens PIH (2007) Growth factor-induced MAPK network topology shapes Erk response determining PC-12 cell fate. Nat Cell Biol 9:324–330
34. Qiao L, Nachbar RB, Kevrekidis IG, Shvartsman SY (2007) Bistability and oscillations in the Huang-Ferrell model of MAPK signaling. PLoS Comput Biol 3(9):e184
35. Becskei A, Serrano L (2000) Engineering stability in gene networks by autoregulation. Nature 405(6786):590–593
36. Austin D, Allen M, McCollum J, Dar R, Wilgus J., Sayler G, Samatova N, Cox C, Simpson M (2006) Gene network shaping of inherent noise spectra. Nature 439(7076):608–611
37. Nevozhay D, Adams RM, Murphy KF, Josić K, Balázsi G (2009) Negative autoregulation linearizes the dose–response and suppresses the

heterogeneity of gene expression. Proc Natl Acad Sci 106(13):5123–5128

38. Rosenfeld N, Elowitz MB, Alon U (2002) Negative autoregulation speeds the response times of transcription networks. J Mol Biol 323:785–793
39. Blanchini F, Franco E, Giordano G (2013) A structural classification of candidate oscillators and multistationary systems. bioRxiv doi:10.1101/000562
40. Alon U (2006) An introduction to systems biology: design principles of biological circuits. Chapman & Hall/CRC, Boca Raton
41. Davidson EH, Rast JP, Oliveri P, Ransick A, Calestani C, et al. (2002) A genomic regulatory network for development. Science 295 (5560):1669–1678
42. Savageau MA, Voit EO (1987) Recasting nonlinear differential equations as s-systems: a canonical nonlinear form. Math Biosci 87:83–115
43. Bashor CJ, Helman NC, Yan S, Lim WA (2008) Using engineered scaffold interactions to reshape MAP kinase pathway signaling dynamics. Science 319(5869):1539–1543
44. Ingolia NT, Murray AW (2007) Positive-feedback loops as a flexible biological module. Curr Biol 17:668–677
45. O'Shaughnessy EC, Palani S, Collins JJ, Sarkar CA (2011) Tunable signal processing in synthetic MAP kinase cascade. Cell 144:119–131

Part III

Circuit Analysis and Simulations

Chapter 11

Efficient Analysis Methods in Synthetic Biology

Curtis Madsen, Chris Myers, Nicholas Roehner, Chris Winstead, and Zhen Zhang

Abstract

This chapter describes new analysis and verification techniques for synthetic genetic circuits. In particular, it applies *stochastic model checking* techniques to models of genetic circuits in order to ensure that they behave correctly and are as robust as possible for a variety of different inputs and parameter settings. In addition to stochastic model checking, this chapter proposes new variants to the *incremental stochastic simulation algorithm* (iSSA) that are capable of presenting a researcher with a simulation trace of the typical behavior of the system. Before the development of this algorithm, discerning this information was extremely error-prone as it involved performing many simulations and attempting to wade through the massive amounts of data. This algorithm greatly aids researchers in designing genetic circuits as it efficiently shows the researcher the most likely behavior of the circuit. Both the iSSA and stochastic model checking can be used in concert to give a researcher the likelihood that the system exhibits its most typical behavior, as well as, non-typical behaviors. This methodology is applied to several genetic circuits leading to new understanding of the effects of various parameters on the behavior of these circuits.

Key words Design space exploration, Synthetic genetic circuits, Stochastic model checking, Stochastic simulation, Synthetic biology

1 Introduction

Recently, researchers have been interested in not only understanding how biological systems work but also designing new ones. In order to accomplish this goal, researchers have combined ideas from systems biology and genetic engineering with the engineering principles of standards, abstraction, and automated construction to create the field of *synthetic biology* [2, 8]. Some applications of this field include enabling designed biological systems to consume toxic waste [6], hunt and kill tumor cells [1], and produce drugs [28] and biofuels [3]. To realize these goals, researchers are actively working on better ways to model and analyze *synthetic genetic circuits*, networks of genes that influence each other's expression via the production and binding of transcription factor proteins. When designing and analyzing genetic circuits, researchers are often interested in building circuits that exhibit a particular behavior. Usually determining a circuit's behavior involves simulating a model to produce some time series data and analyzing this data to discern whether or not the circuit behaves appropriately. This method becomes less attractive as circuits grow in complexity because it ends up becoming very time-consuming to generate a sufficient amount of runs for analysis. In addition, trying to select representative runs out of a large data set is tedious and error-prone, which motivates methods of automating this analysis. These problems have led to the need for design space exploration techniques that allow synthetic biologists to easily explore the effect of varying parameters in their systems and efficiently consider alternative designs of their systems.

This chapter presents a new approach to design space exploration which leverages a combination of stochastic simulation and model checking. In particular, this chapter describes an *incremental stochastic simulation algorithm* (iSSA) that is a variant of *Gillespie's stochastic simulation algorithm* (SSA) [13] which can aid researchers in determining a system's typical behavior. To determine the likelihood of this typical behavior, this chapter describes a method to analyze genetic circuits using *stochastic model checking* which applies *Markov chain analysis* to obtain the desired results. An example work flow of using these methods in concert to analyze a genetic circuit is shown in Fig. 1. Here, a genetic circuit is both simulated using the iSSA and abstracted using logical abstraction. Next, a property representing the typical behavior of the system is determined from the iSSA results and sent to the stochastic model checker along with the resulting *continuous-time Markov chain* (CTMC) from logical abstraction. Finally, the output of this work flow is the probability that the circuit exhibits its typical behavior.

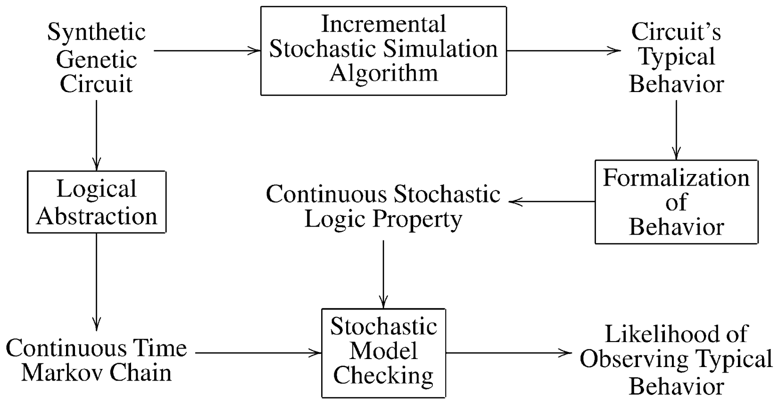

Synthetic Genetic Circuit
Incremental Stochastic Simulation Algorithm
Circuit's Typical Behavior
Logical Abstraction
Continuous Stochastic Logic Property
Formalization of Behavior
Continuous Time Markov Chain
Stochastic Model Checking
Likelihood of Observing Typical Behavior

Fig. 1 An example work flow of using iSSA, logical abstraction, and stochastic model checking to determine the likelihood of a system exhibiting its typical behavior

2 Genetic Toggle Switch

All living organisms are composed of one or more cells, and each of these cells carries out specialized functions in order to keep the organism alive. There are networks within the cell called *genetic circuits* which help regulate the amount of proteins that are synthesized from the cell's genes. This regulation can be used to signal when the cell should divide, when the cell should take in nutrients from the environment, when the cell should change to a defensive state, and when the cell should die, among other behaviors. Understanding these circuits can give synthetic biologists greater insight into why cells behave as they do and can in turn lead to better engineered synthetic genetic circuits. One of the first synthetic genetic circuits constructed is the genetic toggle switch described in [10] and shown in Fig. 2. This circuit has two stable states: either LacI is high and TetR is low (the OFF state) or LacI is low and TetR is high (the ON state). To switch from one stable state to the other, the values of IPTG and aTc can be altered. In 2000, Gardner et al. successfully designed and constructed this circuit and inserted it into *Escherichia coli* bacteria where they were able to observe this switch-like behavior. In Fig. 2, LacI represses production of TetR and *green fluorescent protein* (GFP) by binding to the P_{trc-2} promoter, while TetR represses production of LacI by binding to the $P_{LtetO-1}$ promoter. GFP is a reporter protein that causes the cell to glow, thereby indicating whether the toggle is in the ON or OFF state. The other molecules in the diagram, IPTG and aTc, are *chemical inducers*. IPTG inhibits LacI's ability to act as a repressor by binding with it to form a complex C1. Similarly, aTc inhibits TetR's ability to act as a repressor by binding with it to form a complex C2. This chapter uses this genetic circuit as a running example.

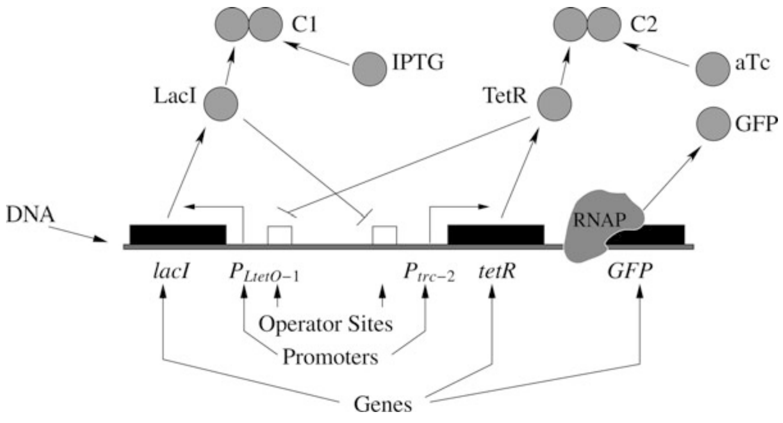


Fig. 2 The genetic toggle switch circuit where LacI and TetR repress each other (denoted by the ⊣ *arrows*). In this circuit, LacI can be sequestered by IPTG, TetR can be sequestered by aTc, and GFP is the reporter protein causing the cell to glow indicating whether the toggle is in the ON or OFF state

3 Classical Chemical Kinetics

A genetic circuit can be converted into a *chemical reaction network* composed of *species* and *reactions* that can be analyzed using *classical chemical kinetics* (CCK). CCK are well-known methods which assume that the network reacts continuously in a well-stirred medium and that there is a large amount of molecules in the system. In CCK, the rate of each reaction is considered to be the speed at which it alters the amount of the species that participate in it per unit time. To compute these rates, CCK converts the network into a set of *ordinary differential equations* (ODEs) using the *law of mass action*. The law of mass action states that the rate of each chemical reaction can be computed by multiplying the reaction's *rate constant* by the product of the *concentrations* (i.e., the amount of molecules divided by the compartment volume) of its reactant species. An ODE model can be analyzed using some well-known techniques [27]. These techniques typically involve linearizing the system by evaluating the rate of change of each of the species' concentrations using the initial concentration of each species. The system is then evolved by selecting a small time step and computing a new concentration for each species at the end of the time step. This process is then repeated by computing new rates with the new concentration values and a new time step. Since it is often the case that the ODE rate equations are dependent on the concentration of many species, the results of ODE analysis may be erroneous if the time step is too large. However, if the time step is too small, then the analysis takes an extremely long time as many unnecessary steps are taken to evolve the system. To address this problem, there have been improvements that try to adapt the time step selection so that it is able to make progress in evolving the system but is never so large that the ODE rates for each species change too dramatically.

4 Stochastic Chemical Kinetics

As stated previously, CCK assumes that the system behaves continuously and deterministically. When the amounts of each species are very large, this assumption is reasonable as changes in the population size of each species are relatively small and can be viewed as continuous changes. In addition, small fluctuations of species counts in a system where the population sizes are large can be safely disregarded as they do not affect the system significantly. Therefore, the dynamics of the system can also be viewed as a deterministic process. In genetic circuits, the species counts are typically very small and each reaction occurs sporadically. Since these circuits violate the assumptions of CCK, the true behavior of the system is not able to be captured when analyzing it using CCK.

Thus, researchers have turned to *stochastic chemical kinetics* (SCK) for analyzing genetic circuits. SCK assumes that a chemical reaction network behaves as a discrete-stochastic process. It achieves this assumption by treating the network as a well-stirred system with a discrete number of molecules. SCK can then simulate the time evolution of the system by stochastically firing reactions to change the amounts of each species in the system.

SCK keeps track of species' amounts instead of species' concentrations. Indeed, the system state in SCK is simply the population of each species in the system, denoted $\vec{x} = (x_1, \ldots, x_N)$ where N is the number of species in the system. Changes in the system state occur through the firing of reactions. When a reaction r_j fires, it applies a *state change vector*, $\vec{v}_j = (v_{1j}, \ldots, v_{Nj})$, to the system state where each $v_{i\,j}$ is the amount of change reaction r_j causes to species s_i's population. The new system state is then defined by $\vec{x} + \vec{v}_j$. When simulating a genetic circuit using SCK, all that matters are the current molecule counts as they are the only values that can change and affect the *propensity functions* $a_j(\vec{x})$ for each reaction r_j in the system. Each $a_j(\vec{x})$ is the probability that, given $\vec{x}$, reaction r_j occurs within the next infinitesimal time interval that is so small that at most one reaction event can occur. Therefore, each transition of the system only depends on the current state of the system and does not depend on the history of the system. This fact means that the circuit can be treated as a *temporally homogeneous jump Markov process* [14].

The Markov process described by SCK can be defined as the probability, $P(\vec{x}, t + dt|\vec{x}_{\text{init}}, t_{\text{init}})$, that the system state is $\vec{x}$ at time $t + dt$ given that it was $\vec{x}_{\text{init}}$ at time t_{init}. In order to compute this probability, all the possible states that are only one step away from the system being in state $\vec{x}$ are considered. The probability of moving from each of these states to state $\vec{x}$ (i.e., each reaction propensity) is multiplied by the probability that the system reached state $\vec{x} - \vec{v}_j$ at time t and these values are summed together. Additionally, the probability that the system is already in state $\vec{x}$ and no reactions occur is added to this probability to give the final Markov process probability. Formally, this probability is defined as:

$$\begin{aligned} P(\vec{x}, t + dt|\vec{x}_{\text{init}}, t_{\text{init}}) =& P(\vec{x}, t|\vec{x}_{\text{init}}, t_{\text{init}}) \left[1 - \sum_{j=1}^{M} a_j(\vec{x})dt \right] \\ &+ \sum_{j=1}^{M} [P(\vec{x} - \vec{v}_j, t|\vec{x}_{\text{init}}, t_{\text{init}}) a_j(\vec{x} - \vec{v}_j)dt] \end{aligned} \tag{1}$$

where M is the number of reactions in the system. Taking the limit as dt goes to 0 gives the *chemical master equation* (CME—*see* Chapter 9 as well):

$$\frac{\partial P(\vec{x}, t|\vec{x}_{\text{init}}, t_{\text{init}})}{\partial t} = \sum_{j=1}^{M} [P(\vec{x} - \vec{v}_j, t - dt|\vec{x}_{\text{init}}, t_{\text{init}})a_j(\vec{x} - \vec{v}_j) \\ - P(\vec{x}, t|\vec{x}_{\text{init}}, t_{\text{init}})a_j(\vec{x})] \tag{2}$$

which defines the time evolution of the state probabilities for the genetic circuit. Solving the CME is typically infeasible for most genetic circuits due to the fact that Eq. 2 is a set of coupled ODEs for each system state and most realistic systems have state spaces that are infinite.

The behavior defined by the CME can be determined using the SSA, an algorithm that generates a time course simulation trajectory for the CME [12, 13]. The main advantage of the SSA over other stochastic simulation approaches is that it steps over time steps where no reactions occur. Algorithm 1 presents a version of the SSA known as the *direct method*. This algorithm begins by initializing the simulation time to t_{init} and the current state vector which holds counts for each molecule to $\vec{x}_{\text{init}}$ (line 1). The algorithm then enters a loop (lines 2–11) where the state vector is updated and time is advanced until time reaches the time limit T. During each iteration of this loop, the propensity function of each reaction is computed and summed (lines 2–4), two random numbers are drawn (line 5), the time of the next reaction and the next reaction are determined using these random numbers and the propensity functions (lines 6–7), and the state vector and time are updated and recorded (lines 8–9). Since the development of the SSA, there have been many variations and improvements to the algorithm. Among these are the first-reaction method [12], the next-reaction method [11], the SSA-CR method [29], tau-leaping [15], ssSSA [5], and wSSA [18].

5 Reaction-Based Abstractions

Even with all of the improvements to the SSA, analysis of genetic circuits using SCK can still be very computationally intensive. One way to alleviate this problem is to try to reduce the model to a smaller, less complex model that still preserves the behavior of the original model. This reduction can be done by applying abstractions to the model before it is analyzed. *Reaction-based abstractions* attempt to reduce the number of species and reactions in the model [17]. These abstractions typically improve simulation time as they attempt to eliminate time scale separation in the model. Namely, they try to remove fast reactions that slow down the simulation because they fire often preventing the simulation from making significant progress.

There are many reaction-based abstractions [17]. One example abstraction is *operator site reduction* which is used to remove fast reversible reactions in genetic circuit models that represent the binding of RNAP, *activators*, and *repressors* to promoter operator sites. This abstraction begins by first determining the species that represent the operator sites of the network (i.e., the promoters).

Algorithm 1: SSA(Initial state $\langle \vec{x}_{\text{init}}, t_{\text{init}} \rangle$; Reactions $R_{i \in 1 \ldots m}$; Time limit T)

```
1  Initialize the current simulation time $t = t_{\text{init}}$ and the current state vector $\vec{x} = \vec{x}_{\text{init}}$.
2  forall Reactions $R_{i \in 1 \ldots m}$ do
3      Evaluate the propensity functions $a_i(\vec{x})$ at state $\vec{x}$.
4  Evaluate the sum of all the propensity functions:
       $a_0(\vec{x}) = \sum_{i=1}^{m} a_i(\vec{x})$
5  Draw two unit uniform random numbers, $r_1$, $r_2$.
6  Determine the time, $\tau$, until the next reaction:
       $\tau = \frac{1}{a_0(\vec{x})} \ln\left(\frac{1}{r_1}\right)$
7  Determine the next reaction, $R_\mu$, where $\mu$ is the smallest integer satisfying
       $\sum_{i=1}^{\mu} a_i(\vec{x}) > r_2 a_0(\vec{x})$
8  Determine the new state after firing $R_\mu$: $t = t + \tau$ and $\vec{x} = \vec{x} + \vec{v}_\mu$ where $\vec{v}_\mu$ is the update vector after firing $R_\mu$.
9  Record $(\vec{x}, t)$.
10 if $t < T$ then
11     Go to step 2.
```

It then finds how many different binding configurations that the operator site can be in and computes a *quasi-steady-state approximation* value for each of these bindings. This approximation is essentially the values of the species counts that form each binding multiplied by an *equilibrium constant* for the reaction (the forward rate constant over the reverse rate constant). The sum of these values and 1 become the denominator of the new reaction rates for the reduced reactions. For each binding reaction that leads to production of another species, the reduced reaction rate is the product of the rate of the original production reaction, the total species count of the operator, and the quasi-steady-state approximation value for the binding over the summation described above. The function rate(p), shown in Eq. 3, can be derived which returns the rate of production initiated from promoter p.

$$\text{rate}(p) = \begin{cases} \dfrac{n_p k_o p K_o \text{RNAP}}{1+K_o\text{RNAP}+\sum_{s_r \in \text{Rep}(p)} (K_r s_r)^{n_c}} & \text{if Act}(p) = 0 \\ \dfrac{n_p k_b p K_o \text{RNAP} + \sum_{s_a \in \text{Act}(p)} n_p k_a p K_{oa} \text{RNAP} (K_a s_a)^{n_c}}{1+K_o\text{RNAP}+\sum_{s_r \in \text{Rep}(p)} (K_r s_r)^{n_c} + \sum_{s_a \in \text{Act}(p)} K_{oa}\text{RNAP}(K_a s_a)^{n_c}} & \text{otherwise} \end{cases} \tag{3}$$

In this equation, n_p is the number of proteins produced per transcript, k_o is the open-complex production rate, k_b is the basal production rate, k_a is the activated production rate, K_o is the RNAP binding equilibrium constant, K_{oa} is the activated RNAP binding equilibrium constant, K_r is the repressor binding equilibrium constant, K_a is the activator binding equilibrium constant, and n_c is the degree of cooperativity. Note that Act(p) returns the activating species, s_a, for promoter p, and Rep(p) returns the repressing species, s_r, for promoter p.

If there are complex formation reactions between species and chemical inducers such as that between LacI and IPTG, *complex formation* and *sequestering* abstractions can be applied to the model. These abstractions are related as they both deal with complex formation reactions. The complex formation abstraction uses a steady state approximation to remove complexes from the original model. This abstraction replaces the value of a complex, c_i, in a rate function with the expression $K_c s_i s_j$, where s_i and s_j are species that bind to form c_i and K_c is the complex formation equilibrium constant. For instance, applying this abstraction to complex C1 in the genetic toggle switch would replace each instance of it with $K_c|\text{LacI}||\text{IPTG}|$. The sequestering abstraction, on the other hand, uses the quasi-steady-state approximation in addition to the law of mass conservation and replaces the value of a species, s_i, in a rate function with the expression $\frac{s_{i_\text{total}}}{1+K_c s_j}$, where s_{i_total} is the variable for the total amount of the species (free and in complex) and s_j is the variable for the other species that binds with s_i to form the complex. This rate shows that as the amount of s_j increases, the effective amount of s_i decreases. In the genetic toggle switch, everywhere TetR appears in a rate equation, it is replaced by $\frac{|\text{TetR}|}{1+K_c|\text{aTc}|}$ when using this abstraction.

Applying the operator site reduction and sequestering abstractions to the genetic toggle switch reaction network shown in Fig. 3a, the model is reduced from 14 species and 13 reactions to 5 species and 5 reactions. This resulting reaction graph is shown in Fig. 3b.

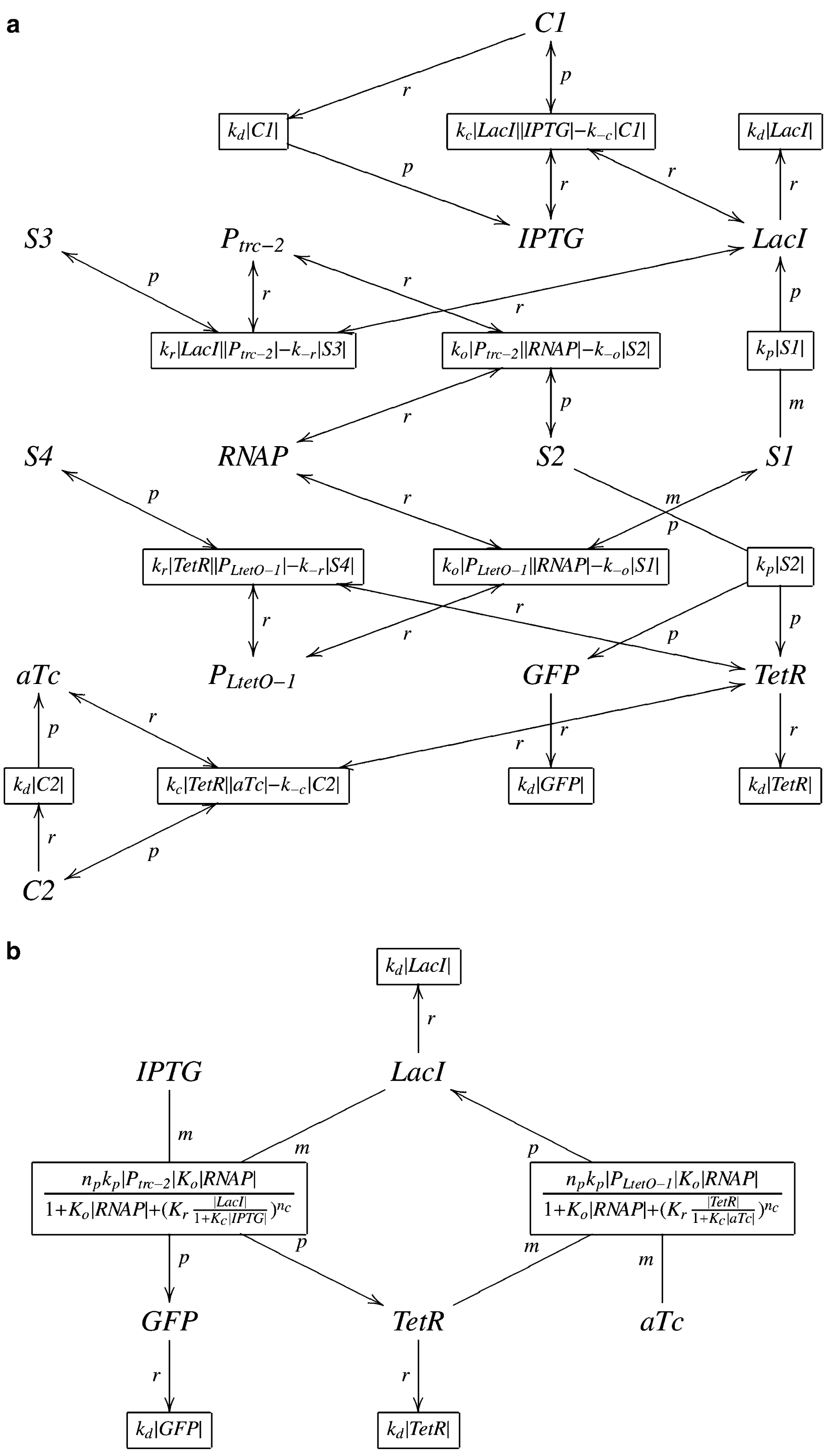

a
C1
$k_d|C1|$
$k_c|LacI||IPTG|-k_{-c}|C1|$
$k_d|LacI|$
S3
P_{trc-2}
IPTG
LacI
$k_r|LacI||P_{trc-2}|-k_{-r}|S3|$
$k_o|P_{trc-2}||RNAP|-k_{-o}|S2|$
$k_p|S1|$
S4
RNAP
S2
S1
$k_r|TetR||P_{LtetO-1}|-k_{-r}|S4|$
$k_o|P_{LtetO-1}||RNAP|-k_{-o}|S1|$
$k_p|S2|$
aTc
$P_{LtetO-1}$
GFP
TetR
$k_d|C2|$
$k_c|TetR||aTc|-k_{-c}|C2|$
$k_d|GFP|$
$k_d|TetR|$
C2
b
$k_d|LacI|$
IPTG
LacI
$\frac{n_p k_p|P_{trc-2}|K_o|RNAP|}{1+K_o|RNAP|+(K_r \frac{|LacI|}{1+K_c|IPTG|})^{nc}}$
$\frac{n_p k_p|P_{LtetO-1}|K_o|RNAP|}{1+K_o|RNAP|+(K_r \frac{|TetR|}{1+K_c|aTc|})^{nc}}$
GFP
TetR
aTc
$k_d|GFP|$
$k_d|TetR|$
r
p
m

Fig. 3 Reaction graph for the genetic toggle switch (**a**) before and (**b**) after reaction-based abstraction. Reaction rate equations are shown in the *boxes* representing each reaction. The edges between species and reactions are labeled *r* if the species is a reactant in the reaction, *p* if the species is a product of the reaction, and *m* if the species is a modifier in the reaction

6 Incremental Stochastic Simulation Algorithm

Typically, whenever a researcher wants to determine the behavior of a genetic regulatory circuit, he or she analyzes it using either ODE or SSA simulation. For example, simulation results for the genetic toggle switch are expected to select either the ON or the OFF state when the circuit is initially set to a state where all the signals are low. Figure 4 shows the individual SSA simulation runs that capture this expected behavior. In contrast, Fig. 5 shows an ODE simulation of the toggle switch where the circuit neither switches ON or OFF but

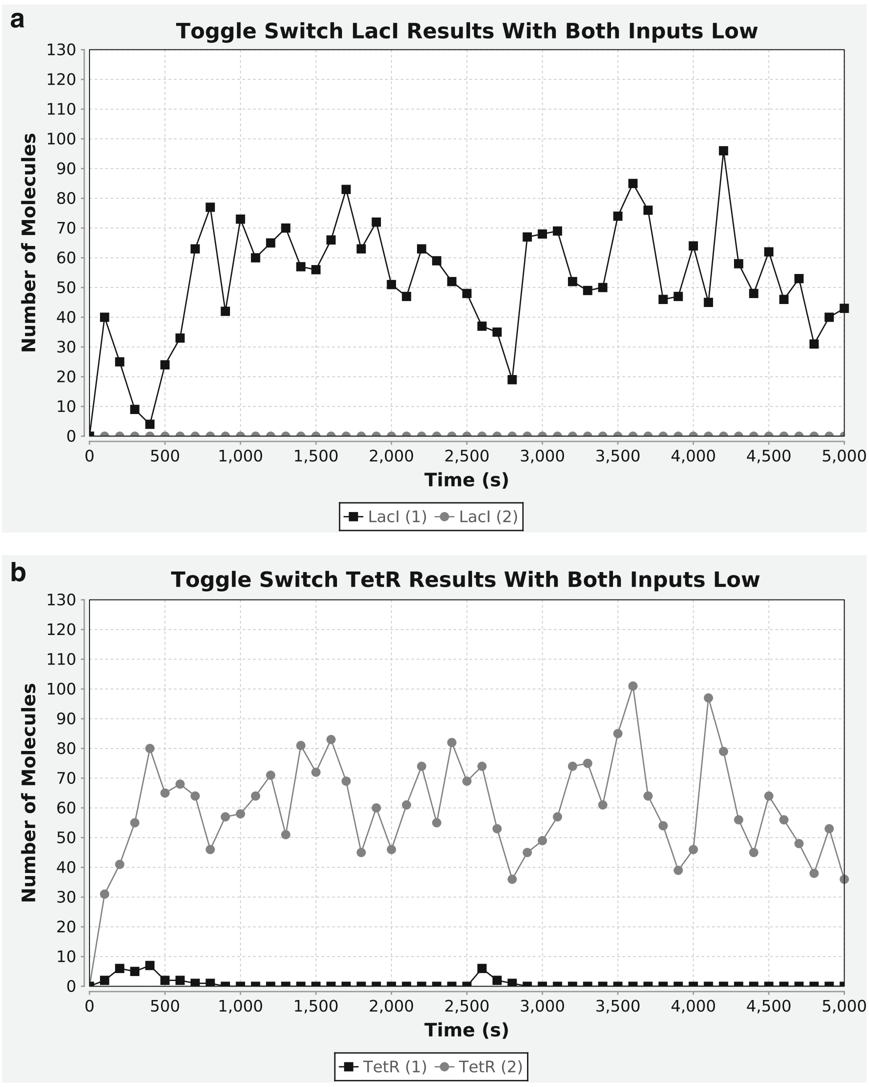


Fig. 4 Individual SSA results for the genetic toggle switch initialized to a state where aTc, IPTG, LacI, TetR, and GFP are low. (**a**) Plot showing two individual SSA runs for the LacI species. (**b**) Plot showing two individual SSA runs for the TetR species

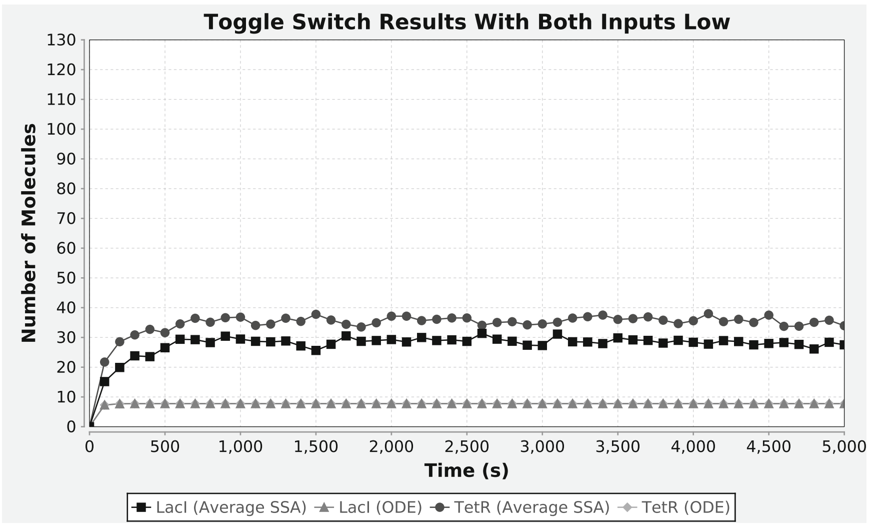


Fig. 5 ODE results and the mean $\overline{x}(t)$ of 100 SSA runs for the genetic toggle switch initialized to a state where aTc, IPTG, LacI, TetR, and GFP are low

instead stabilizes at an intermediate state. Similarly, the plot of the average of 100 SSA simulations shows the toggle switch going to a false intermediate state. This plot also shows that the average of 100 runs suffers from the smoothing effect and fails to capture the noisy fluctuations observed in the original circuit.

The goal of the iSSA, on the other hand, is to produce time series data that represents typical behavior of a chemical reaction network. The main idea behind the iSSA is to perform stochastic simulation runs in small time increments [20, 33]. At the end of each time increment, statistics are computed over all of the simulation runs. These statistics are then used in the next time increment to constrain and select a new starting state for each run.

The iSSA algorithm, presented in Algorithm 2, is essentially a wrapper around Gillespie's SSA that continually starts and stops several simulation runs at the beginning and end of each time increment. The iSSA takes as parameters a maximum number of simulation runs (`maxRuns`), a simulation time limit (`timeLimit`), a desired number of "typical" paths to follow (`paths`), an initial state-vector and system time $\langle \vec{x}_{\text{init}}, t_{\text{init}} \rangle$, and a collection of reactions $R_{l \in 1 \ldots m}$. The iSSA begins by initializing the record table where ending states are stored ($\mathbf{X}$) to the initial state and system time (line 1). At the start of each kth increment, the run number, i, is reset to 1 (line 2). Next, a starting state is selected using the `select` function along with the starting time for each run in the increment (`start`) (line 3). In addition to computing the starting state for the increment, the ending time for the increment, `limit`, is computed using the `findLimit` function (line 4). At this point, the iSSA executes a step of Gillespie's SSA where it selects a random time and reaction event and computes a new state (lines 5–12). The newly computed simulation time is compared against `limit` to determine if the algorithm needs to compute another step of the SSA (line 13). If this limit has not been reached, then the algorithm recomputes `limit` and another SSA step is performed (line 14). Otherwise, the algorithm records the current time and state for the current run by calling the `record` function (line 15). If the maximum number of runs has not been reached, a new simulation run is started from the original state chosen by the `select` function (lines 16–18). Once all the runs have been completed for the time increment, the algorithm determines if it has reached the end of the simulation time or if it needs to compute another time increment (lines 19–21). Finally, the iSSA returns several sequences of states equal to the `paths` variable chosen by the `select` function that represents that many "typical" simulation traces of the model.

Previously published variants of iSSA are susceptible to being skewed by outlier results, are highly dependent on the time increment returned by `findLimit`, and are incapable of producing results for more than one "typical" path [20, 33]. To combat these problems, the iSSA variant presented here utilizes a clustering algorithm to follow multiple "typical" paths for a circuit, uses an adaptive time step, and selects the median states from the previous increment's clusters as the starting states in the next increment. In order to accomplish these goals, the `record` function simply records the ending state for each run in a state table. The `findLimit` function is defined to be doubly adaptive as in Algorithm 3. The general idea of this function is to continually update the remaining time in the time increment based on changes in the slowest reaction's propensity and how much progress has been made in the increment so far. In this algorithm, the amount of progress made towards the current limit is computed by subtracting the current time, t, from the starting time, `start`, and by multiplying this value by the smallest propensity from the previous time increment, $prev\_a_{min}$ (line 1). Then, the remaining amount of desired events is computed by finding the difference between the desired number of events, `#events`, and the previously computed progress, `progress` (line 2). Next, the current smallest propensity value is stored into the $prev\_a_{min}$ variable for future computations (line 3), and the new time limit is returned by dividing the remaining progress, `remaining`, by the current smallest propensity value and adding this difference to the starting time (line 4). It should be

Algorithm 2: `iSSA`(Maximum number of runs `maxRuns`; Time limit `timeLimit`; Number of Paths `paths`; Initial state $\langle \vec{x}_{\text{init}}, t_{\text{init}} \rangle$; Reactions $R_{l\in 1\ldots m}$)

```
1  Initialize: $k = 1$ and $\mathbf{X}^{(0)} = \langle \vec{x}_{\text{init}}, t_{\text{init}} \rangle$.
2  Set $i = 1$.
3  Set $\langle \vec{x}, t \rangle$ = select($\mathbf{X}^{(k-1)}$, paths) and start = $t$.
4  Set limit = findLimit(start, $\vec{x}$, $t$).
5  Execute a Gillespie SSA step:
6    forall Reactions $R_{l\in 1\ldots m}$ do
7        Evaluate the propensity functions $a_l(\vec{x})$ at state $\vec{x}$.
8    Evaluate the sum of all the propensity functions:
             $$a_0(\vec{x}) = \sum_{l=1}^{m} a_l(\vec{x})$$
9    Draw two unit uniform random numbers, $r_1$, $r_2$.
10   Determine the time, $\tau$, until the next reaction:
             $$\tau = \frac{1}{a_0(\vec{x})} \ln\left(\frac{1}{r_1}\right)$$
11   Determine the next reaction, $R_\mu$, where $\mu$ is the smallest integer satisfying
             $$\sum_{l=1}^{\mu} a_l(\vec{x}) > r_2 a_0(\vec{x})$$
12   Determine the new state: $t = t + \tau$ and $\vec{x} = \vec{x} + \vec{v}_\mu$.
13 if $t$ < limit then
14     Go to step 4.
15 record($\mathbf{X}^{(k)}$, $\vec{x}$, $t$, $i$).
16 if $i$ < maxRuns then
17     Set $i = i + 1$.
18     Go to step 3.
19 if $t$ < timeLimit then
20     Set $k = k + 1$.
21     Go to step 2.
```

noted in this algorithm that when t equals `start`, `progress` equals zero and the algorithm returns a limit that tries to capture the desired number of the slowest event. By continually updating the time increment in this manner, this method is able to adjust to drastic changes in reaction propensities and capture approximately `#events` of the slowest reaction event.

The `select` function uses the k-means clustering algorithm on the data from the previous increment found in the record table [22]. This clustering creates `paths` clusters the first time the function is called for an increment, and this result is cached so that it can be drawn from in subsequent calls to `select`. The `select` function proportionally returns an ending state that has the smallest Euclidean distance from one of the medians of each cluster based on how many ending states are in each cluster. The iSSA defined in this way is capable of returning multiple paths by stitching the starting points of each increment with the selected ending point that has the highest proportion of runs that ended in its cluster.

Algorithm 3: findLimit(Start time start; Current state $\vec{x}$; Current time t)

```
1 Set progress = $(t - \texttt{start}) \times prev\_a_{min}$.
2 Set remaining = #events − progress.
3 Store $prev\_a_{min} = a_{min}(\vec{x})$.
4 return start + $\frac{\texttt{remaining}}{a_{min}(\vec{x})}$.
```

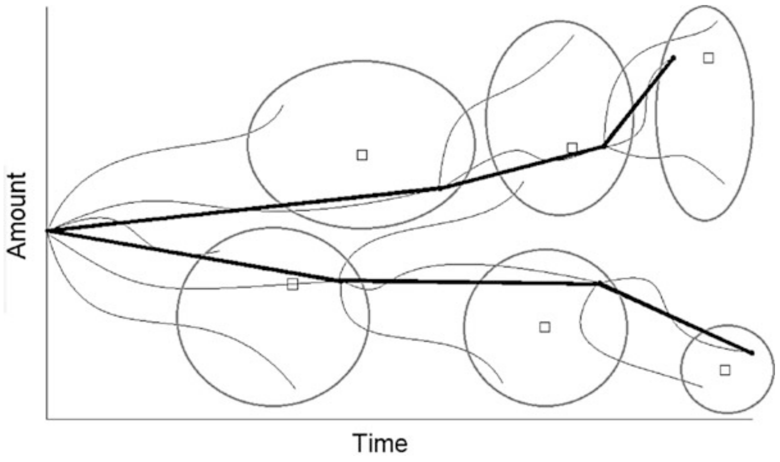
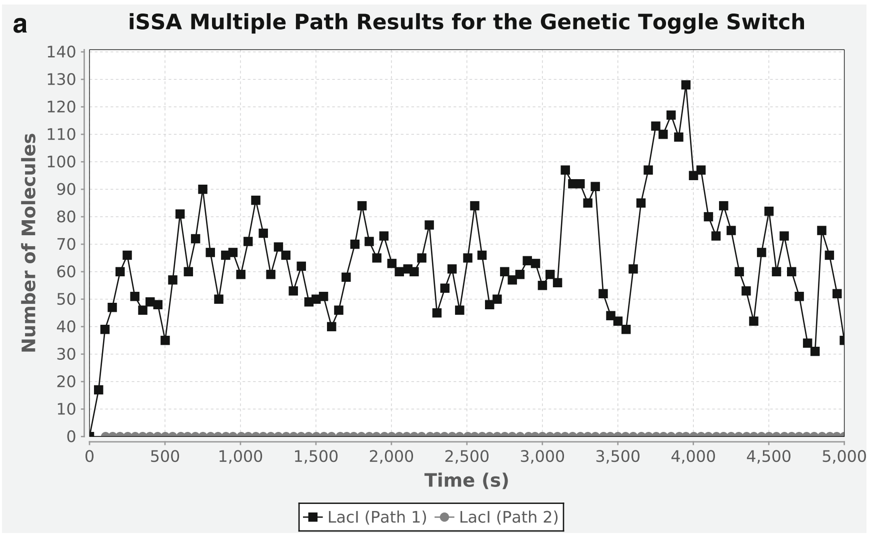


Fig. 6 Illustration showing how the iSSA performs simulations. At the end of each time increment, the ending states are clustered (clusters are shown by *gray circles*) and a state from each cluster is selected as a starting state for the next time increment. The algorithm then starts a number of runs from each representative state equal to the number of runs that ended in that representative state's cluster. The algorithm is using an adaptive time increment and selecting the median state from each cluster

Figure 6 illustrates how the iSSA is able to utilize the *k*-means clustering algorithm and an adaptive time step to select two representative median states in each time increment. The gray circles represent the clusters and the thick black lines represent the selected paths. An example of using the iSSA method to simulate the genetic toggle switch is shown in Fig. 7 in which the toggle switch starts in a state where both inputs are low and both LacI and TetR are low. The circuit has two "typical" behaviors in this case. It can either switch to the ON state or the OFF state. The iSSA captures both of these cases showing that it is capable of tracking multiple paths.



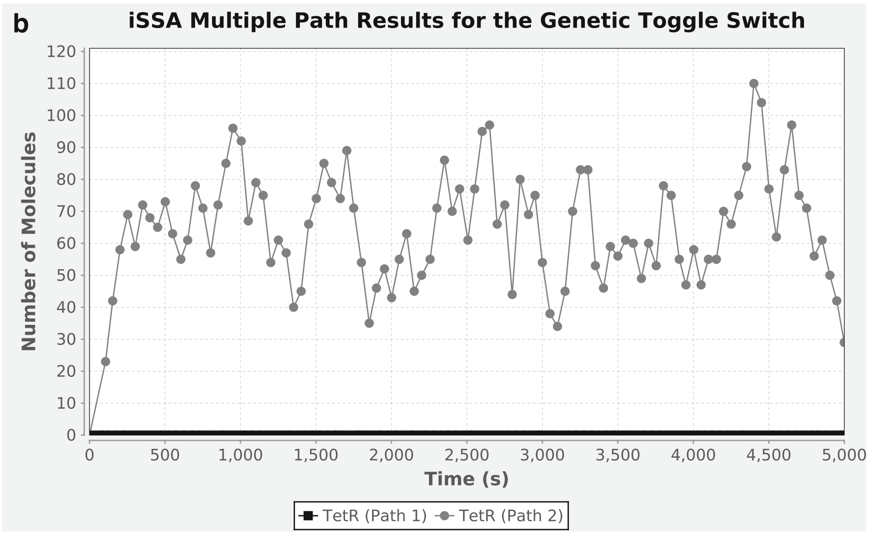


Fig. 7 iSSA simulation results for the genetic toggle switch. (**a**) Plot showing two paths for the LacI species. (**b**) Plot showing two paths for the TetR species. In this simulation, the toggle switch starts out in a state with both inputs low as well as LacI and TetR being low. This initial condition leads to the circuit selecting to switch to the ON or the OFF state. As seen in the plots, the iSSA is able to capture both of these cases

7 Stochastic Model Checking

When designing and analyzing genetic circuits, researchers are often interested in the probability that the system reaches a given state or satisfies a specific condition within a certain amount of time. Usually, this process involves simulating the system to produce some time series data and analyzing this data to discern the state probabilities. However, as the complexity of models of genetic circuits grow, the amount of simulation data needed to determine these probabilities within a certain confidence interval becomes prohibitively expensive to compute. To address this problem, researchers can use stochastic model checking techniques employing Markov chain analysis methods to find the state space of the system directly and compute the probability of being in each state at a given time. However, due to genetic circuits having infinite state spaces, this goal is accomplished by logically abstracting a genetic circuit into a finite-state CTMC. This CTMC can then be analyzed using Markov chain analysis to determine the likelihood that the circuit satisfies a given property. This methodology is used to determine the likelihood of certain behaviors in a genetic circuit. When compared to stochastic simulation-based analysis of the same circuit, the results agree with the reported probabilities but obtain a substantial speedup over these approaches.

7.1 Translation of a Genetic Circuit into a CTMC

The critical step in preparing a genetic circuit for stochastic model checking is the conversion of the circuit into a CTMC. This conversion process begins by finding the state space of the genetic circuit. First, a sparse matrix where each entry, $p_{i,j}$, represents the rate of moving from state i to state j is constructed. Next, a state is created with an encoding of the initial values of the species in the model. The conversion then performs a depth first search by changing one species encoding at a time to a higher or lower encoding if they exist in a predetermined level set L. Each valid change found this way is pushed onto a stack. The algorithm then pops an encoding off the stack and checks to see if a transition rate for moving from the current state to the new state exists in the matrix. If it does, the algorithm stops exploring this path and pops the next change off the stack to explore further. Otherwise, the transition rate is calculated and is added to the matrix. Figure 8 shows a graphical representation of the state space for the genetic toggle switch with thresholds selected at 0, 30, and 60 for both LacI and TetR yielding nine states labeled S0 through S8. State S0, the state where LacI is at its highest level of 60 and TetR is at its lowest level of 0, is the initial state.

Once the entire state space of the genetic circuit is found, the transition rates between the states are computed and inserted into the sparse matrix. These rates are determined using the reaction rates from the genetic circuit after the reaction-based abstractions described in Section 5 have been applied to the model. These rates are computed using the formulas in Eqs. 4 and 5:

$$\text{production}(s, l, l') = \frac{\sum_{p \in \text{Pro}(s)} n_p \cdot \text{rate}(p)}{(l'[s] - l[s])} \tag{4}$$

$$\text{degradation}(s, l, l') = \frac{k_d l[s]}{(l[s] - l'[s])} \tag{5}$$

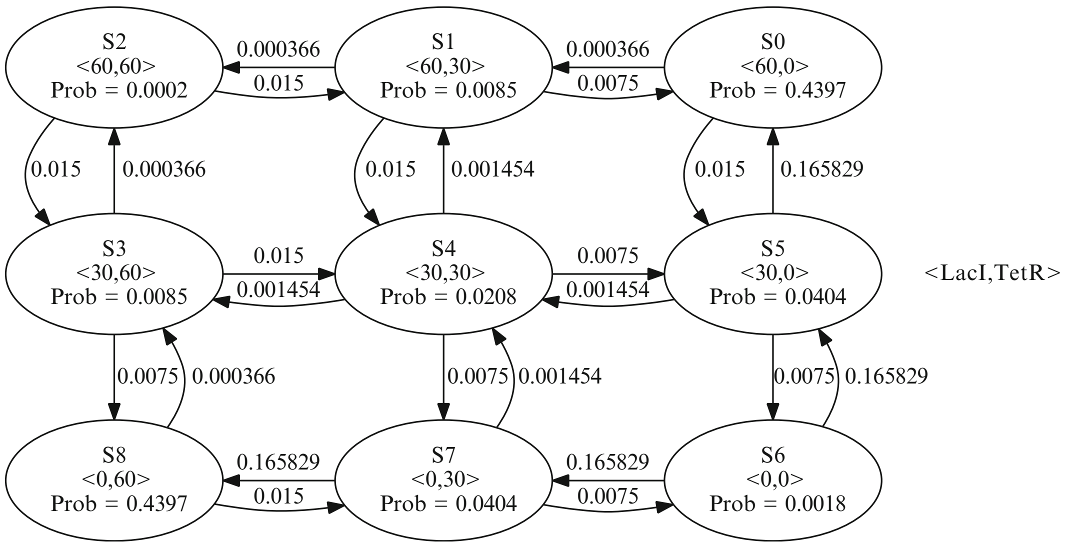


Fig. 8 The CTMC annotated with probabilities after applying steady state Markov chain analysis

In these equations, s is the species value that is changed by the transition rate equation, l is the current state array, l' is the next state array, and Pro(s) returns the set of promoters that initiate transcription of genes that lead to the production of species s. When the state transition increases the level of species s from $l[s]$ to $l'[s]$, then the production formula is used, and when the state transition decreases the level of s from $l[s]$ to $l'[s]$, then the degradation formula is used. The rate for production is computed by determining the rate of production for each promoter p which produces species s using the rate(p) function defined in Eq. 3. To obtain a numerical value from this equation, the species variables in the rate(p) equation are replaced by indexing into the l state array to get their current values. This rate is then multiplied by n_p, the number of proteins produced per transcript, to convert this rate into the rate for a single protein production. The rate of degradation is computed as $k_d l[s]$ where k_d is the degradation rate parameter and $l[s]$ is the starting level for species s before degradation. In both cases, these rates must be normalized by the difference in the level before and after the state change. This normalization is necessary because the rates are for the production or degradation of a single molecule of s while the state change only occurs after $l'[s] - l[s]$ molecules are produced or $l[s] - l'[s]$ molecules are degraded. The conversion process coupled with the corresponding rate functions have been carefully constructed such that the CTMC generated gives a reasonable approximation of the behavior of the genetic circuit. This translation procedure allows the user to efficiently trade-off between accuracy and analysis time by adjusting the number of thresholds.

7.2 Specifying a Property

CTMC properties can be specified using *continuous stochastic logic* (CSL) [4, 21]. The grammar for CSL properties used by the stochastic model checking algorithm presented in this chapter is given as follows:

$$\begin{aligned} Prop &::= U(\mathrm{T},\Psi,\Psi)|F(\mathrm{T},\Psi)|G(\mathrm{T},\Psi)|St(\Psi) \\ \Psi &::= true|\Psi\wedge\Psi|\neg\Psi|\phi\geq\phi|\phi>\phi|\phi=\phi \\ \phi &::= v_i|c_i|\phi+\phi|\phi-\phi|\phi{*}\phi|\phi/\phi|Prop \\ \mathrm{T} &::= true|\mathrm{T}\wedge\mathrm{T}|\neg\mathrm{T}|t\geq c_i|t>c_i|t=c_i \end{aligned}$$

where v_i is a variable, c_i is a constant, and t stands for time in the system.

Table 1 defines the symbols used in the CSL grammar. As described in this table, Ψ represents a state formula that must be true in a given state and can be composed of other state formulae combined using logical connectives or comparisons between numerical expressions, ϕ. The formula $U(T,\Psi_1,\Psi_2)$ represents the probability that an execution of the system satisfies the until formula $\Psi_1 U^{\mathrm{T}}\Psi_2$ which means that Ψ_1 must remain true until Ψ_2 becomes true within the time frame that the time bound expression, T, evaluates to true. The eventually operator, F, is essentially used as a shorthand for describing an until property where the left-hand side of the formula is true. For example, the eventually formula $F(\mathrm{T},\Psi)$ would simply require that Ψ becomes true before T evaluates to false. The globally true formula $G(\mathrm{T},\Psi)$ requires that Ψ remains true during the time that T evaluates to true. This formula builds off of the eventually operator by requiring that $\neg\Psi$ does not eventually become true while T evaluates to true and returns one minus the resulting probability. The formula $St(\Psi)$ represents the probability that once the system reaches its steady state, it is in a state where Ψ is satisfied. It should be noted that *Prop* is a symbol in ϕ's grammar rule which allows for CSL properties to

Table 1
CSL grammar symbols

Symbol	Description
Prop	CSL property that, when checked, produces the probability that a state formula is true in the steady state or within a given time bound expression.
Ψ	State formula that can be the logical conjunction of other state formulae or a comparison of numerical expressions.
ϕ	Numerical expression that can be a variable, a constant, a CSL property, or an arithmetic operator applied to other numerical expressions.
T	Time bound expression that can be the logical conjunction of other time bound expressions or a comparison of the time variable to a constant.

be nested within other CSL properties. As a shorthand, Ψ and T can also contain false, $\vee$, $<$, and $\leq$ which are easily derived.

7.3 Stochastic Model Checking

After the CTMC is constructed and a CSL property has been specified, a stochastic model checker can check each state to determine the likelihood of the CSL property. There are two types of stochastic model checking used to compute the likelihood that a property is true: statistical and numerical-based techniques [21, 34]. Statistical techniques involve simulating a system a large number of times and terminating whenever a property is shown to be true or false. When all of the simulations are complete, statistics are calculated on how many simulations satisfied the property in the time allotted versus the number of simulations that failed to do so. One downside of using statistical techniques is that the more rare an event is, the more simulations that need to be run in order to observe it, and performing these simulations may cause the time that it takes to compute a likelihood to become prohibitively expensive. Numerical methods, on the other hand, attempt to determine these likelihoods in a more direct method. They employ Markov chain analysis to compute the probability of a CSL property being satisfied. These methods are often more efficient than statistical techniques; however, they require that the state space be computable. Both statistical and numerical methods have been utilized by many tools such as the probabilistic model checker PRISM [16].

Algorithm 4 determines the probability within an error bound, ε, of a given CSL property, Φ, on a genetic circuit model, M. Additionally, this algorithm requires a set of levels, L, that includes an ordered list of threshold levels, L_s, for each species $s \in S$ in the model. Each level, $l_{s,i}$ represents a critical threshold in the amount of the species s. It is assumed that $l_{s,0}$ is always 0, and $l_{s,i-1} < l_{s,i}$ for all $i > 0$. The first step of the algorithm converts the model into a CTMC, C, using logical abstraction discussed earlier in this section and property pruning described at the end of this section (line 1). Next, the algorithm parses the CSL property and walks its expression tree looking for any nodes that represent nested properties (line 2). If any nested properties are found, the algorithm loops over each state in C (line 3) and alters M by setting its initial state to the Markov chain state, s (line 4). The algorithm then recursively calls itself for each altered model, M', using the nested property, Φ', as the new CSL property to be checked (line 5). The probability of each recursive call, p, is stored as a variable of each s where it can be referenced when determining if the state satisfies Φ (line 6). Once all of the nested properties are dealt with, the algorithm determines the amount of time necessary for the analysis, t, which is essentially the maximum value that time can take while still allowing for the time bound expression to evaluate to true in the transient property or infinity, ∞, in the case of a steady state property (line 7). Finally, the

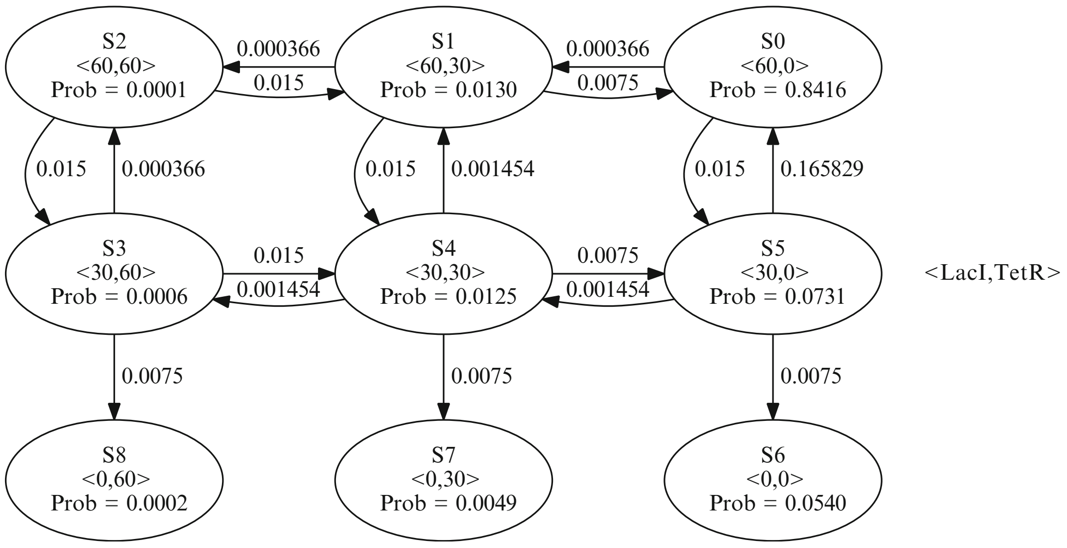


Fig. 9 The CTMC annotated with probabilities after applying transient Markov chain analysis with the CSL property, $\mathtt{F}(t \leq 100, \text{LacI} = 0)$

algorithm checks whether transient or steady state analysis should be performed and calls the appropriate analysis method (lines 8–11). When checking transient properties, the model checker determines if the encoding either does not satisfy the left-hand side of an until formula or satisfies the right-hand side of the formula since all transient properties can be written as until formulae. If either of these checks holds, the state is marked as absorbing and all transitions out of the state are pruned from the CTMC before analysis. The transient analysis method utilizes the well-known method of *uniformization* while the steady state analysis method uses the *power iteration method* [31].

An example of using the steady state analysis method to analyze the genetic toggle switch is presented in Fig. 8. In this example, the user is interested in the probability of the system being in a state where LacI is 0 in the long run which is represented by the CSL property, $\mathtt{St}$ (LacI = 0). Summing over the states that satisfy this property, states *S*6, *S*7, and *S*8, results in a probability of about 48.2%. Figure 9 shows the CTMC in Fig. 8 annotated with probabilities after applying transient Markov chain analysis to determine the probability of the CSL property $\mathtt{F}(t \leq 100, \text{LacI} = 0)$. Note that states S6, S7, and S8 are marked as absorbing states, since they satisfy the property. The sum of the probability of reaching these states represents the probability of satisfying the property, which is about 5.9%.

Algorithm 4: SMC(Model M; Levels L; CSL property Φ; Error-bound ϵ)

```
Set $C$ = computeCTMC($M, L, \Phi$)
foreach Nested property $\Phi' \in \Phi$ do
    foreach State $s \in C$ do
        Set $M' = M.setInitialState(s)$
        Set $p$ = SMC($M', L, \Phi', \epsilon$)
        $s.addVariable(\Phi', p)$
Set $t$ = determineTimeLimit($\Phi$)
if $t \neq \infty$ then
    return transientAnalysis($C, t, \Phi, \epsilon$)
else
    return steadyStateAnalysis($C, \Phi, \epsilon$)
```

8 Case Studies

The iSSA and stochastic model checking techniques are useful methods on their own for determining the validity of models of genetic circuits; however, when these methods are used in concert, they can be used to effectively perform design space exploration of genetic circuits. This section presents several examples where using the iSSA and stochastic model checking allows for better design choices when constructing genetic circuits. Among the genetic circuits analyzed are two genetic oscillators (the repressilator and the dual-feedback genetic oscillator) and three state holding gates (the genetic toggle switch, three implementations of a genetic Muller C-element, and a quorum trigger).

8.1 Repressilator

The repressilator model shown in Fig. 10 is comprised of the species CI, LacI, and TetR that are connected in a loop [7]. This circuit is a ring oscillator where each species represses the next one forming the loop. When one of the species (e.g., LacI) is produced, it represses the next species in the chain (e.g., TetR). This repression allows the species downstream of that species to start being produced (e.g., CI) which in turn leads to the repression of the first species. This cycle continues causing this circuit to oscillate. When iSSA is used as in Fig. 11, the time increments change to try and capture 25 of the slowest reaction events in each increment which leads to stable oscillatory results.

After using the iSSA to verify that the repressilator oscillates, stochastic model checking can be used to determine the probability that the circuit oscillates. Figure 12 presents the results of applying steady state analysis to the repressilator using 9 levels evenly spaced between 0 and 80 for CI, LacI, and TetR and the CSL property, $\texttt{St}((\mathrm{CI} \geq 30 \wedge \texttt{F}(t \leq \text{limit}, \mathrm{CI} < 30) \geq 0.95) \vee (\mathrm{CI} < 30 \wedge \texttt{F}(t \leq \text{limit}, \mathrm{CI} \geq 30) \geq 0.95))$, which determines the likelihood that the value of the CI species is low and goes high or is high and goes low within a predetermined amount of time. These results show that as the time limit is extended, the likelihood of the circuit oscillating increases until for 800 s, it is almost certain to

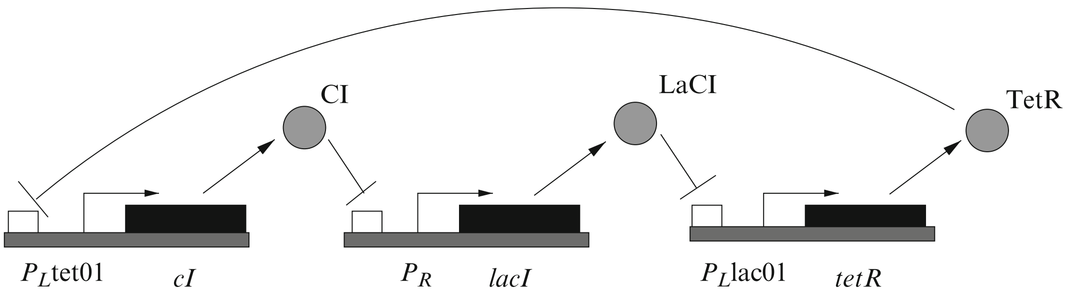


Fig. 10 Model for repressilator. In this circuit, CI represses the production of LacI, LacI represses the production of TetR, and TetR represses the production of CI. This model is expected to produce oscillatory behavior of the CI, LacI, and TetR species

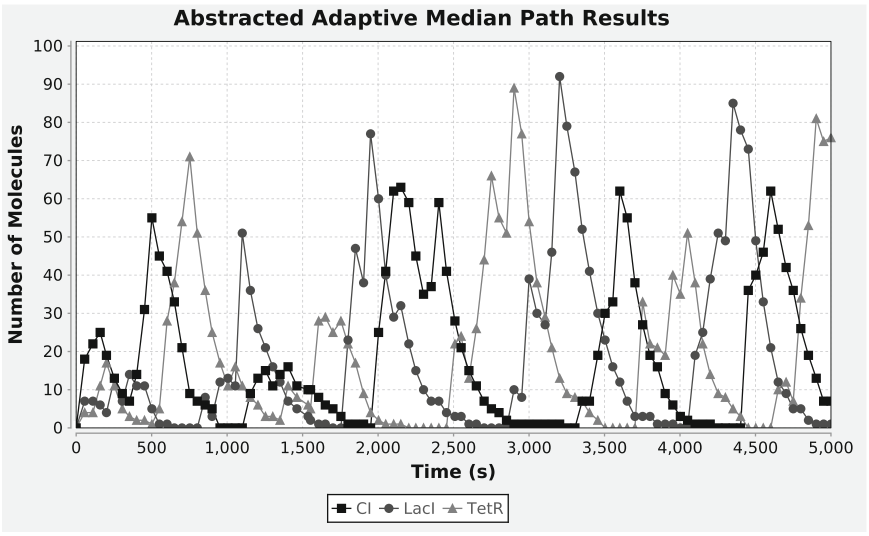


Fig. 11 iSSA results for the repressilator. This method adapts the time increment in order to try and capture 25 of the slowest reaction events in each increment which allows it to obtain stable results as expected

oscillate. In addition to the probability increasing as the time limit increases, the run-time also increases because it takes longer to perform analysis of nested properties with larger time bounds. The run-time for a time limit of 200 s is 3 min, for 400 s is 5 min and 40 s, for 600 s is 8 min and 25 s, and for 800 s is 11 min. This type of analysis can give a designer an idea of how reliable a circuit like the repressilator is as compared to other oscillator circuits so that the best circuit can be selected for a given task.

8.2 Dual-Feedback Genetic Oscillator

The dual-feedback genetic oscillator model shown in Fig. 13 is composed of two identical promoters, P_1 and P_2, which are activated by AraC and repressed by LacI [32]. LacI is produced when

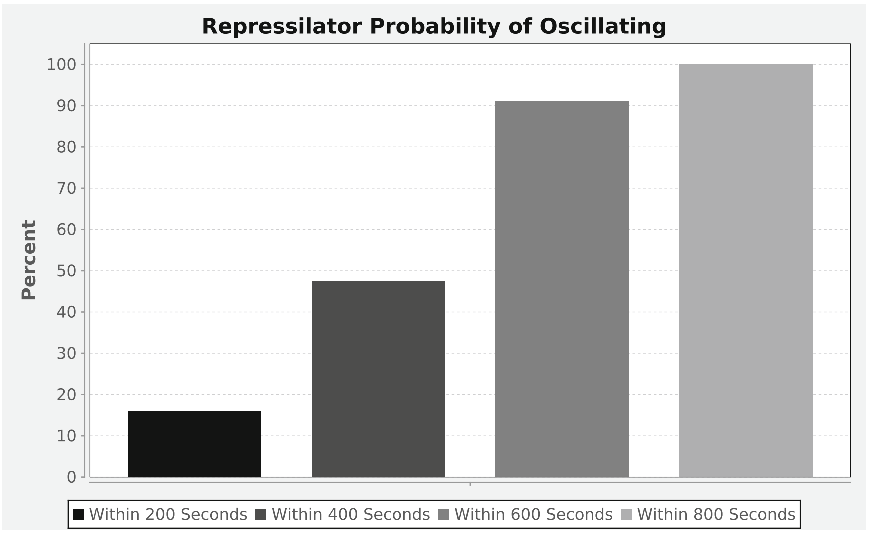


Fig. 12 Stochastic model checking results for the repressilator. This plot presents the results of checking a property on the repressilator that represents states of the circuit switching from low to high or high to low within a specified amount of time. The probabilities represent the likelihood that the circuit oscillates

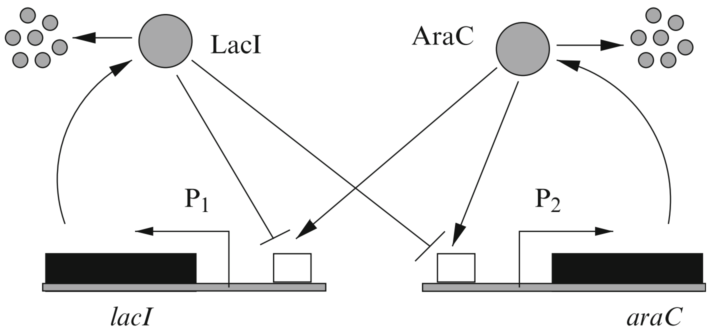


Fig. 13 Model for the dual-feedback genetic oscillator. In this model, LacI represses itself and AraC through promoters P_1 and P_2. Conversely, AraC activates itself and LacI through the same promoters. This model is expected to produce oscillatory behavior of the LacI and AraC species

transcription is initiated at promoter P_1 and the *lacI* gene is transcribed. Similarly, AraC is produced when transcription is initiated at promoter P_2 and the *araC* gene is transcribed. AraC builds up in the system causing LacI to build up. LacI is then able to repress promoters P_1 and P_2 which causes both AraC and LacI concentrations to drop leading to AraC building up again causing the circuit to oscillate. The iSSA results for this system are shown in Fig. 14 and clearly indicate the oscillatory behavior of this system.

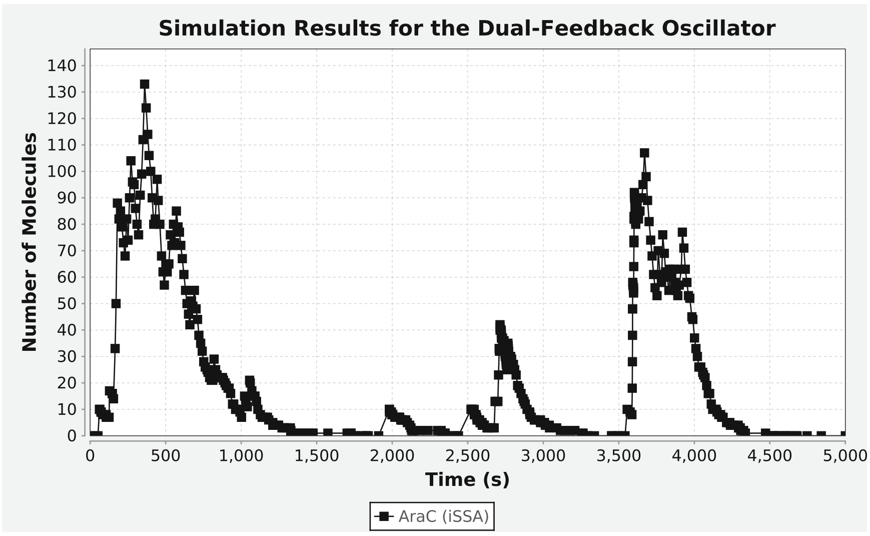


Fig. 14 iSSA simulation results for the dual-feedback genetic oscillator. This plot shows the results of applying the iSSA using 100 simulation runs and 25 slow reaction events per time increment

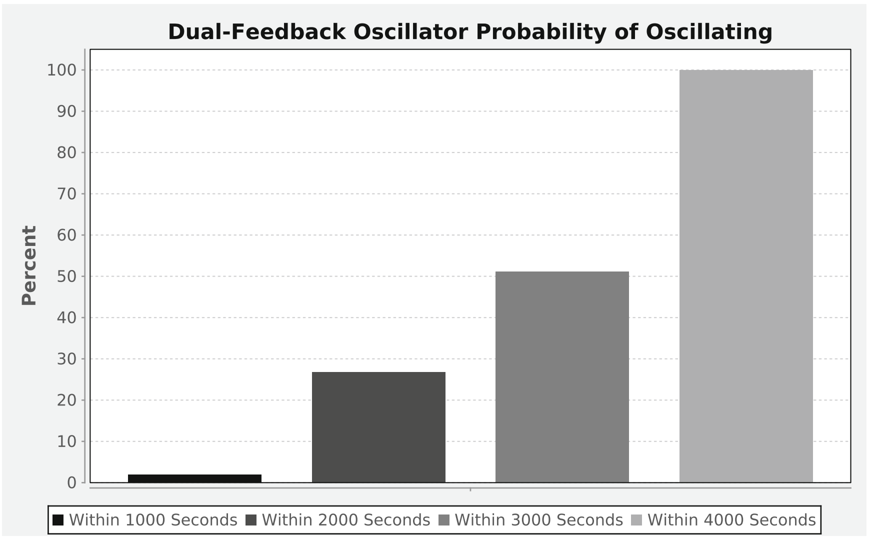


Fig. 15 Stochastic model checking results for the dual-feedback genetic oscillator. This plot presents the results of checking a property on the dual-feedback genetic oscillator that represents states of the circuit switching from low to high or high to low within a specified amount of time. The probabilities represent the likelihood that the circuit oscillates

Similar to the repressilator, the dual-feedback genetic oscillator can be analyzed using stochastic model checking to determine the probability that it oscillates within a certain time bound. Figure 15 presents the results of applying steady state analysis to this model using 8 levels evenly spaced between 0 and 120 for AraC and LacI and the CSL property, $\texttt{St}((\text{AraC} \geq 60 \wedge \texttt{F}(t \leq \text{limit}, \text{AraC} < 60) \geq 0.95) \vee (\text{AraC} < 60 \wedge \texttt{F}(t \leq \text{limit}, \text{AraC} \geq 60) \geq 0.95))$, which captures the probability that AraC is low and goes high or is high and goes low within a predetermined amount of time. Like the repressilator, these results show that as the time limit is extended, the likelihood of the circuit oscillating increases. The run-time for a time limit of 1,000 s is 40 s, for 2,000 s is 1 min and 25 s, for 3,000 s is 1 min and 55 s, and for 4,000 s is 2 min and 40 s. These results show that the dual-feedback genetic oscillator has a much longer period than the repressilator. A designer could use this information to select the appropriate genetic oscillator based on how fast the oscillations need to be for a particular application.

8.3 Genetic Toggle Switch

For our running example, the genetic toggle switch, the desired behavior is that when it starts in the OFF state, it should change to the ON state when IPTG is provided. It should remain in this state even after IPTG is removed. However, it should change to the OFF state after aTc is added, and it should remain in that OFF state even after it is later removed. Results using the iSSA to simulate the genetic toggle switch are shown in Fig. 16. In this figure, the iSSA accurately predicts that the circuit typically behaves exactly as desired.

Due to stochasticity and noise, a state holding gate like the toggle switch can fail. A useful experiment for this circuit is to determine the probability that it changes state erroneously within a cell cycle (2,100 s) which occurs if some spurious production of the

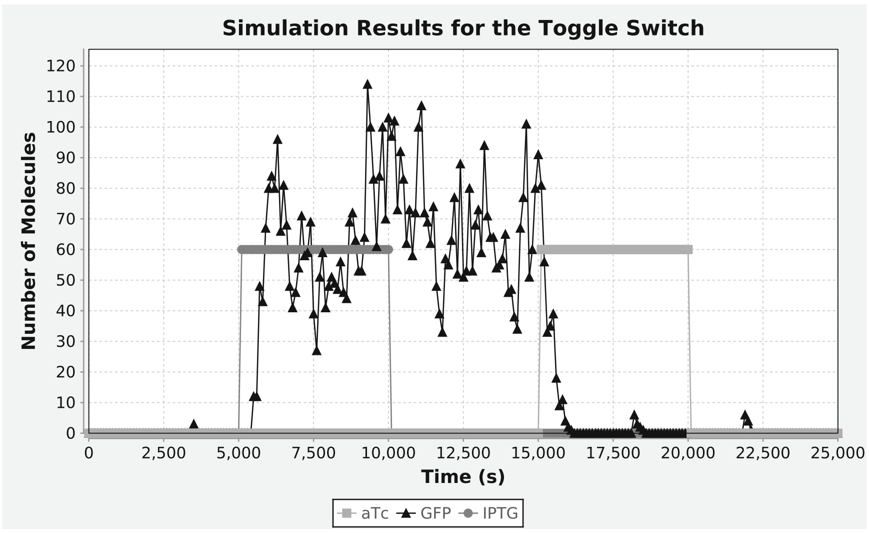


Fig. 16 iSSA simulation results for the genetic toggle switch. This plot shows results of applying the iSSA using 100 simulation runs and 25 slow reaction events per time increment. The circuit is supposed to switch ON at 5,000 s and OFF at 15,000 s due to varying inputs

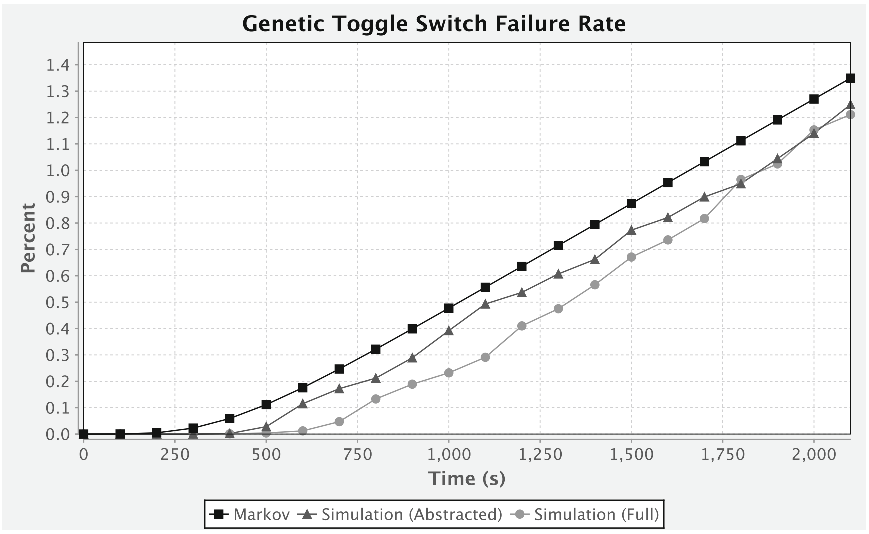


Fig. 17 Time course plot showing the probability of the genetic toggle switch changing state erroneously. This plot compares the results of performing 32,000 simulation runs both with and without reaction-based abstraction with Markov chain analysis. The CSL property used in this case is $\mathtt{F}(t \leq 2100, \text{LacI} < 20 \wedge \text{TetR} > 40)$

low signal inhibits the high signal enough to allow it to degrade away and switch state. For this experiment, the toggle switch is initialized to a starting state where LacI is set to a high state of 60 molecules and TetR is set to a low state of 0 molecules. In order to test whether or not it changes state, the CSL property, $\mathtt{F}(t \leq 2100, \text{LacI} < 20 \wedge \text{TetR} > 40)$, is checked. This property makes states absorbing in which LacI has dropped below 20 (the low state) and TetR has risen above 40 (the high state). For this analysis, 9 levels are selected for LacI uniformly distributed between 0 and 80, and 11 levels are selected for TetR uniformly distributed between 0 and 50, which produces a CTMC with 99 states. Figure 17 shows a comparison of results found using simulation both with and without reaction-based abstraction [19] and applying transient Markov chain analysis to the toggle switch. This figure shows that the transient Markov chain analysis tracks the simulation results fairly closely and ends up with a final probability of 1.35% which is quite close to the 1.2% found by simulation of the full model. However, the transient Markov chain analysis method greatly outperforms the simulation-based approaches as it takes under 1 s to obtain results while the simulation with abstraction takes about 3 min and 15 s to perform 32,000 runs and the simulation without abstraction takes about 43 min for the same number of runs.

The next analysis determines the response time of the circuit when switching from the OFF state to the ON state, and these results are presented in Fig. 18. This analysis uses the same CSL

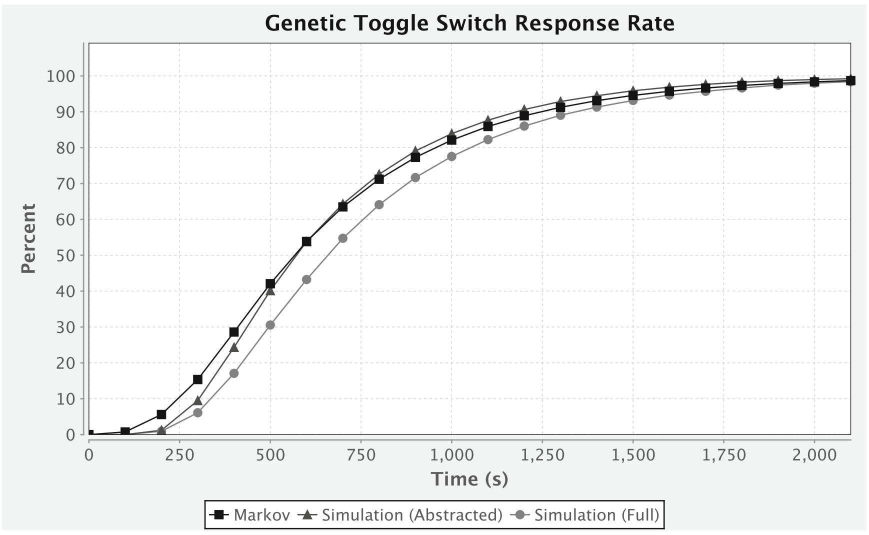


Fig. 18 Time course plot showing the probability of the genetic toggle switch changing state correctly in response to an input change. Like Fig. 17, this plot compares the results of using simulation both with and without reaction-based abstraction and analysis of the CTMC using Markov chain analysis with the same CSL property, $\mathtt{F}(t \leq 2100, \mathrm{LacI} < 20 \wedge \mathrm{TetR} > 40)$, but with a different initial value of IPTG

property but a slightly different initial condition. As before, LacI is set to 60 and TetR is set to 0, but IPTG is set to 100 representing that it has just been added to set the toggle switch to the high state. For this experiment, 14 levels for LacI are selected uniformly distributed between 0 and 130, since individual simulation results show it reaching a much higher value than in the last experiment. For TetR, only 5 levels are used uniformly selected between 0 to 60 because less resolution is required to catch its change from a low to high state. This level selection results in a CTMC of 70 states. Again, transient Markov chain analysis tracks the simulation results fairly closely ending up with a final probability of 98.7% while the simulation of the full model results in 98.9%. Also like the previous example, the transient Markov chain analysis method outperforms the simulation-based approaches as it takes about a half a second to obtain results while 32,000 simulation runs of the reaction-based abstracted model takes about 1 min and the full model takes about 3 hours and 12 min. It should be noted that the reason that the full model takes so much longer to simulate than the abstracted model is that in the presence of IPTG, the full model simulation spends an exorbitant amount of time firing binding and unbinding reactions of LacI and IPTG.

With this analysis method, the design space can be efficiently explored. For example, a genetic designer may consider the effect of parameter variation on robustness and performance. One important parameter for the genetic toggle switch is the degradation rate,

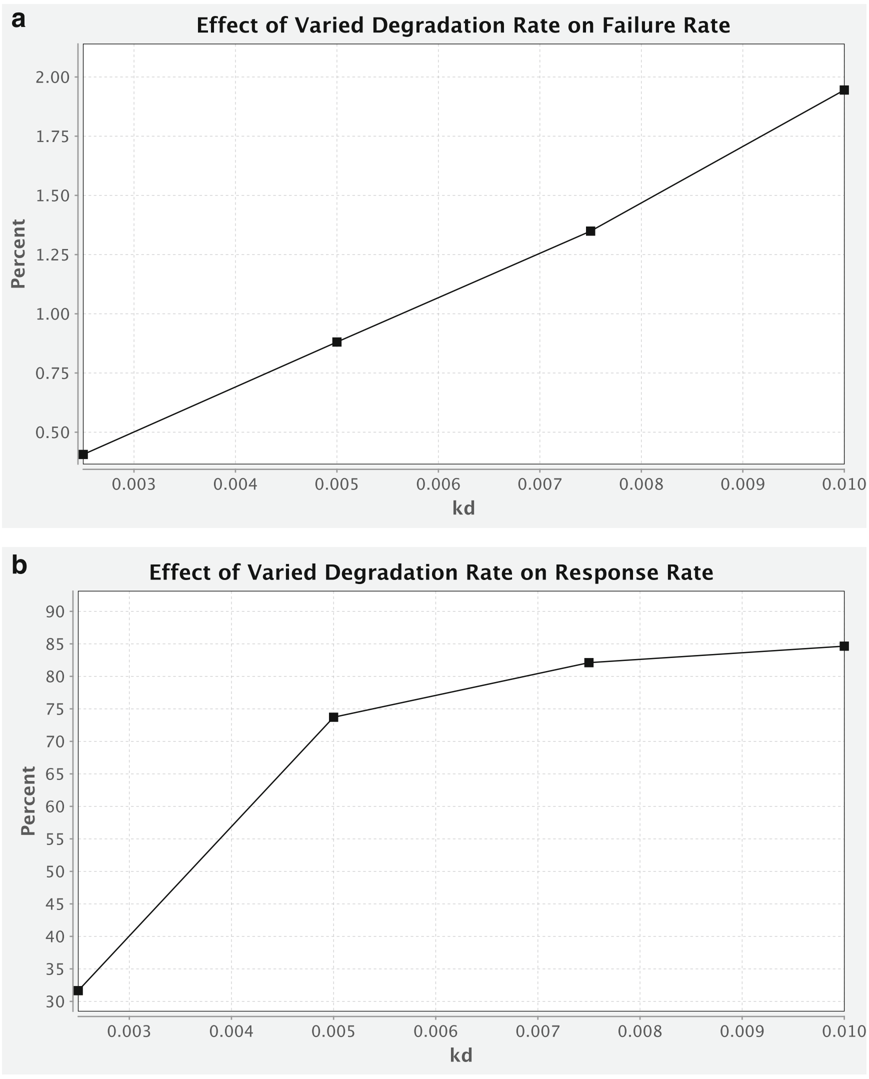


Fig. 19 Results showing the effect of varying the degradation rate, k_d, for the genetic toggle switch. (**a**) Plot depicting the probability of the genetic toggle switch changing state erroneously within 2,100 s for different values of k_d. (**b**) Plot depicting the probability of the genetic toggle switch changing state correctly within 2,100 s in response to input change for different values of k_d

k_d, and the results of varying this parameter are shown in Fig. 19. These results indicate that tuning the degradation rate has a significant effect. If it is too high, the circuit is less robust, but if it is too low, the circuit responds too slowly.

8.4 Genetic Muller C-Element

The next circuit analyzed is a genetic Muller C-element [25, 26]. C-elements are commonly used by asynchronous designers to coordinate parallel processes. To achieve this task a C-element maintains its output until both inputs agree (i.e., they both go to a high state or both go to a low state). When either of these situations occur, the output of the C-element changes to match the inputs. With mixed inputs, on the other hand, the C-element produces an output that is the same as the last time the inputs agreed. Our analysis considers three implementations of this circuit with their logic diagrams shown in Fig. 20. For each of these circuits, the inputs are IPTG and aTc and the output is GFP similar to the genetic toggle switch example. The first implementation shown in Fig. 20a has a majority gate design where the two inputs to the circuit and a feedback signal from the circuit's output are fed through three NAND gates in varying combinations. The outputs from these NAND gates are then compared with each other and the signal that has the majority of the votes is selected as the new output. The next implementation shown in Fig. 20b has a speed-independent design. The idea behind this circuit is that no matter how fast or slow the gates change, the circuit behaves correctly. The final implementation shown in Fig. 20c uses the genetic toggle switch described earlier with some additional logic. This circuit takes an inverted NAND gate signal of the two inputs as the set part of the circuit and an inverted NAND gate signal of the inverted inputs as the reset part of the circuit.

These circuits have slightly different state holding properties when compared to the genetic toggle switch. Instead of switching when one input is applied or taken away, these circuits only switch when both inputs are applied; however, they maintain whatever state they are in until both of the inputs are simultaneously ON or OFF. Figures 21, 22, and 23 present iSSA simulations of the majority gate, speed-independent, and toggle switch implementations, respectively. In each of these analyses, IPTG is added to the circuit at 5,000 s. Each circuit does not switch from OFF to ON, however, until aTc is also added at time 10,000 s. When IPTG is removed at 15,000 s, the circuits maintain state until aTc is removed at 20,000 s. These results show that the iSSA clearly captures these state changes.

Since stochastic noise is present in these circuits as it is in the genetic toggle switch, these circuits are analyzed to find their failure and response rates. Instead of analyzing these circuits against a property that only checks the amount of GFP, dual-rail properties are analyzed because they provide a better measure of the circuit changing state. Figure 24a shows a comparison of the failure rate analysis when each circuit is set in its high mixed state meaning that one input is high, one input is low, the output is high, and the internal species are set appropriately. For this analysis, the CSL property, $\mathtt{F}(t \leq 2100, \text{GFP} < 20 \ \wedge \text{E} > 40)$, is used to analyze the majority gate implementation with 16 evenly spaced levels for GFP between 0 and 150, 16 evenly spaced levels for E between 0 and 45, 6 evenly spaced levels for D between 0 and 250, and 5 evenly spaced levels for X, Y, and Z between 0 and 120. The CSL

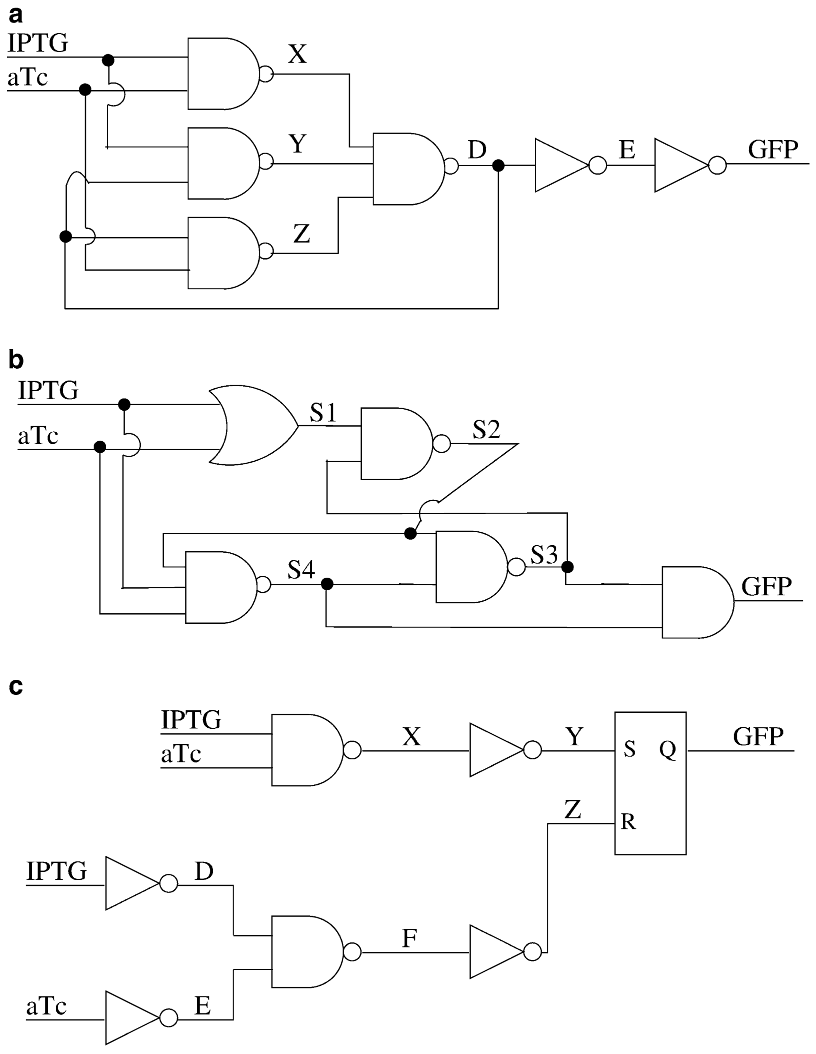


Fig. 20 Logic diagrams for the genetic Muller C-element. Each of these diagrams is made up of a collection of NAND gates, AND gates, OR gates, inverter gates, and set-reset flip-flops wired together in different configurations. AND gates are D-shaped gates and produce a high signal if both inputs are high and a low signal otherwise. NAND gates are also D-shaped gates, but they additionally have a dot at the end of them and produce the inverse of an AND gate (a low signal if both inputs are high and a high signal otherwise). OR gates are arrow-shaped gates that produce a high signal if either or both inputs are high and a low signal if both inputs are low. Inverter gates are triangle-shaped gates with dots at the end and invert a signal (high signals are changed to low signals and vice versa). Set-reset flip-flops are rectangular-shaped boxes that take a set signal (S) and a reset signal (R) and produce an output signal (Q). When S is high, Q is set to high, and when R is high, Q is set to low. (**a**) Majority gate design. (**b**) Speed-independent design. (**c**) Toggle switch design

property for the speed-independent implementation is $\text{F}(t \leq 2100, \text{S3} < 20 \wedge \text{S2} > 80)$, and the levels used are 11 evenly spaced levels for S2 between 0 and 100, 11 evenly spaced levels for S3 between 0 and 150, 6 evenly spaced levels for S1 between 0 and

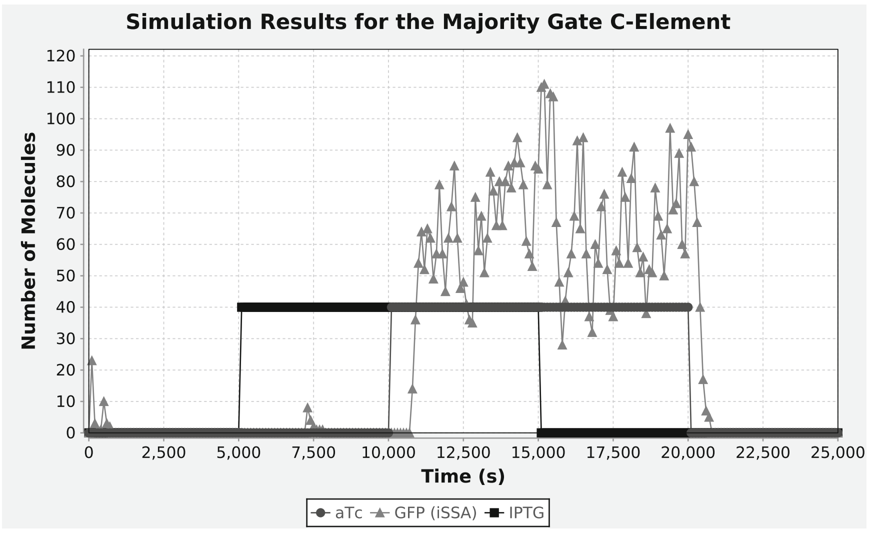


Fig. 21 Plot depicting iSSA simulation results for the majority gate genetic C-element using 100 simulation runs and 25 slow reaction events per time increment

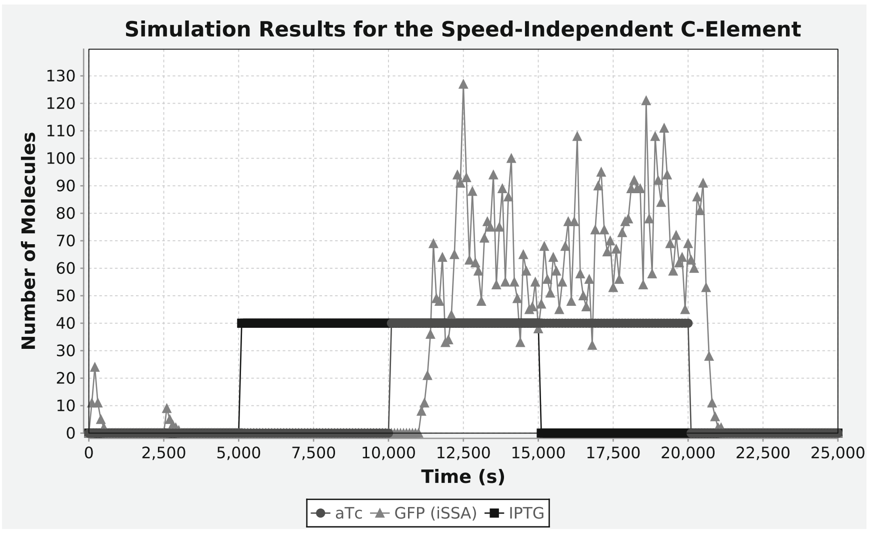


Fig. 22 Plot depicting iSSA simulation results for the speed-independent genetic C-element using 100 simulation runs and 25 slow reaction events per time increment

250, 4 evenly spaced levels for S4 and GFP between 0 and 120, and 4 evenly spaced levels for X, Y, and Z between 0 and 90. Finally, the CSL property for the toggle switch implementation is $F(t \leq 2100, \mathrm{Y} < 40 \wedge \mathrm{Z} > 80)$, and the levels used are 16 evenly spaced levels for Y between 0 and 225, 16 evenly spaced levels for Z between 0 and 90, 6 evenly spaced levels for F between 0 and 250, 5

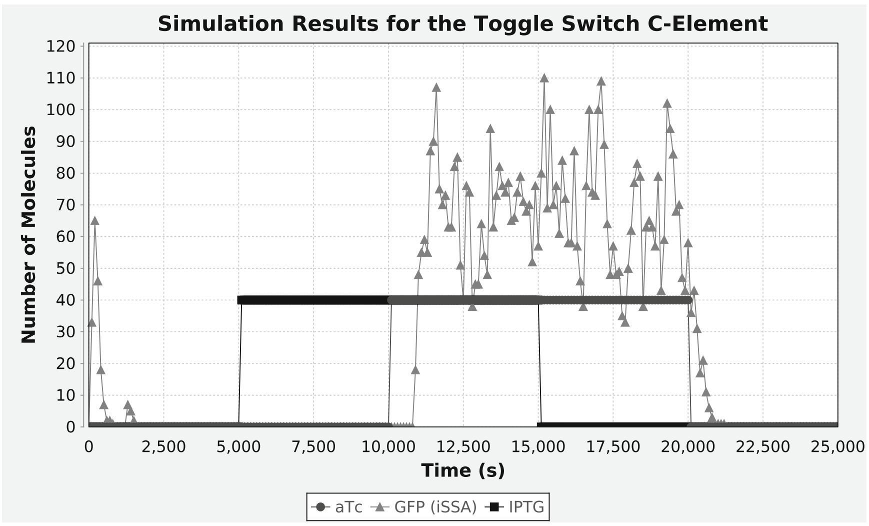
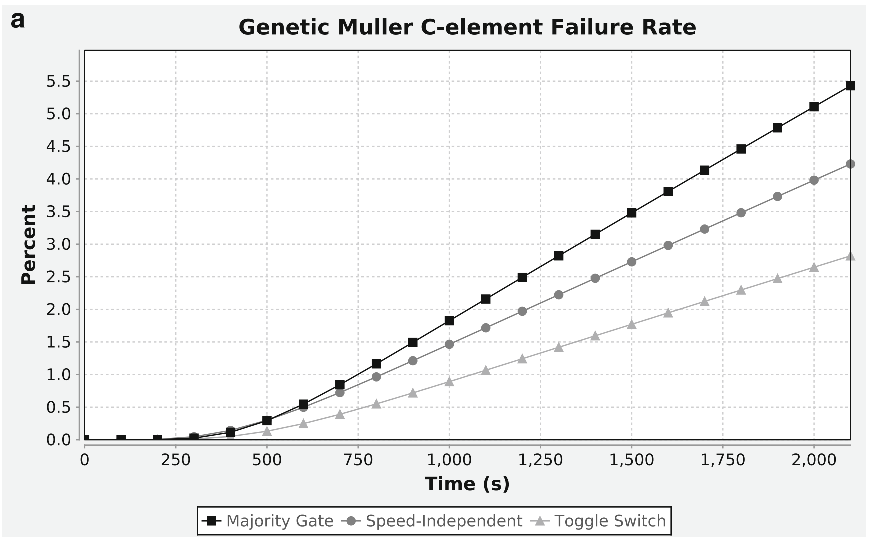


Fig. 23 Plot depicting iSSA simulation results for the toggle switch genetic C-element using 100 simulation runs and 25 slow reaction events per time increment

evenly spaced levels for GFP between 0 and 120, and 5 evenly spaced levels for D, E, and X between 0 and 80.

Run-times for these analyses are 13 min and 15 s for the majority gate, 20 min and 15 s for the speed-independent, and 21 min and 45 s for the toggle switch. This increase in run-time over the genetic toggle switch is due to the fact that each of these models contains substantially more species. In addition, most of the species in the C-element models have more levels defined for them resulting in larger CTMCs. For instance, compared to the genetic toggle switch's 70 to 99 states, the majority gate implementation has nearly 200,000 states, the speed-independent implementation has about 750,000 states, and the toggle switch implementation has nearly 1,000,000 states. From the plot in Fig. 24a, it can be discerned that the toggle switch implementation is the most likely to maintain its state with mixed inputs, followed by the speed-independent implementation, and finally the majority gate implementation.

In order to determine the response time of the C-element circuits, the same initial condition, levels, and properties for each circuit are used with the exception that both inputs are set low indicating that the circuit's output should change from high to low. The results of this analysis are shown in Fig. 24b where it can be seen that the toggle switch implementation again outperforms the speed-independent and majority gate circuits. These analyses have similar run-times to the failure rate experiment with the majority gate taking 11 min and 15 s, the speed-independent taking 20 min, and the toggle switch taking 22 min.



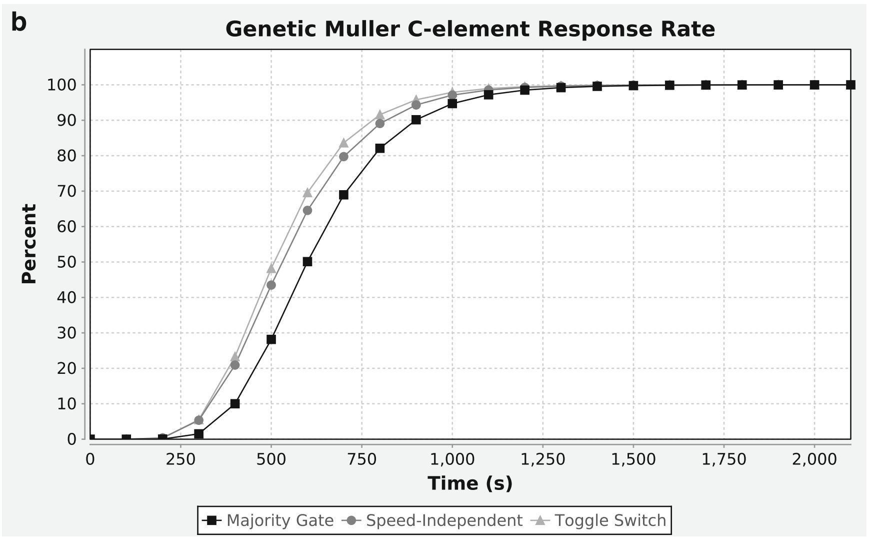


Fig. 24 Results showing the probability of the C-elements changing state for varying inputs. (**a**) Time course plot showing the probability of the C-element implementations losing state with mixed inputs. (**b**) Time course plot showing the probability of the C-element implementations changing state correctly in response to both inputs changing state. These plots compare the results of using Markov chain analysis on the majority gate implementation, the speed-independent implementation, and the toggle switch implementation

After determining that the genetic toggle switch Muller C-element is the most robust (perhaps a surprising conclusion to some), we can now perform various parameter variation experiments to determine the best parameter choices for the application. Figure 25 gives an example of varying the degradation rate of this circuit similar to the experiments performed on the genetic toggle switch in Fig. 19. These results show comparable behavior to that of the genetic toggle switch indicating that there is a trade-off

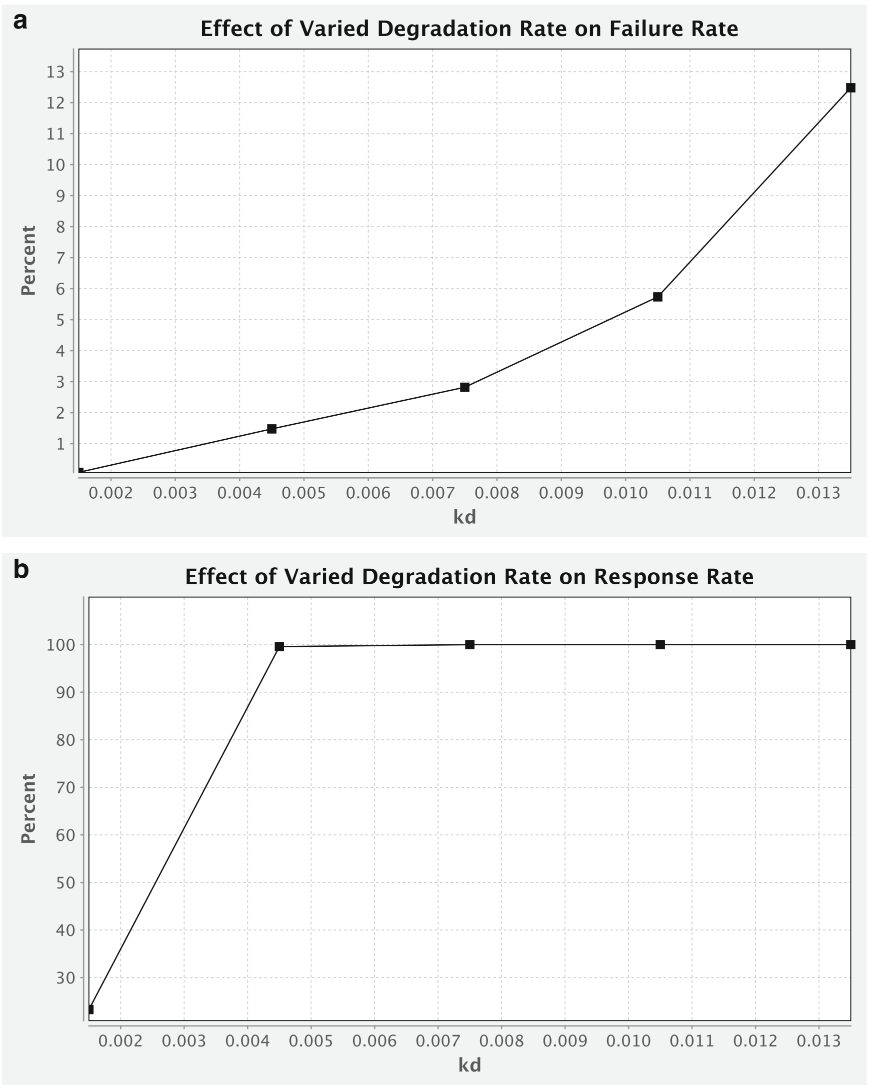


Fig. 25 Results showing the effect of varying the degradation rate, k_d, for the toggle switch implementation of the genetic Muller C-element. (**a**) Plot depicting the probability of the toggle switch implementation of the genetic Muller C-element changing state erroneously for different values of k_d. (**b**) Plot depicting the probability of the toggle switch implementation of the genetic Muller C-element changing state correctly in response to an input change for different values of k_d

between robustness and responsiveness when varying this parameter.

8.5 Quorum Trigger

The final circuit analyzed is the quorum trigger shown in Fig. 26. This circuit is designed to be placed in many cells in a population where it is supposed to allow cells to switch into an ON state in the presence of a signal in the environment. It works by allowing the production of LuxR to be activated by an environmental signal which can then bind with a hormone synthesized by LuxI known

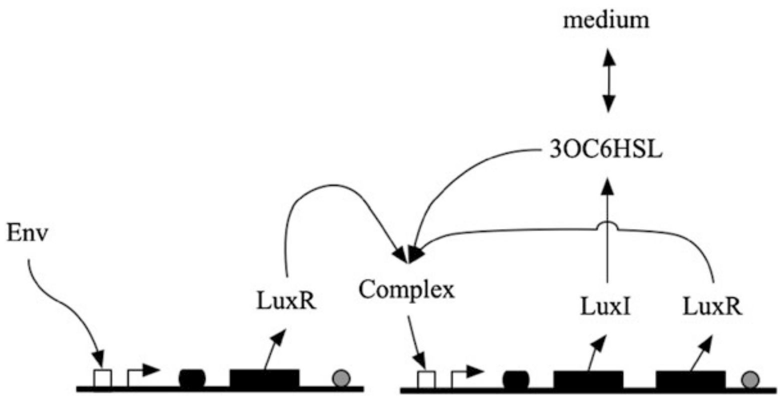


Fig. 26 The quorum trigger circuit. In this genetic circuit, the production of LuxR is activated by a signal in the environment of the circuit. In addition to this activation interaction, LuxI and LuxR production is activated by a complex which is formed when LuxR binds with 3OC6HSL, a hormone synthesized by LuxI that diffuses through a cell's membrane and into the medium the cell is in. This circuit basically works by detecting the presence of the environmental signal and switching to an ON state where it then communicates with other cells through the 3OC6HSL signal to let them know that they should also switch ON

as 3OC6HSL to form a complex. This complex then continues to activate LuxR and LuxI production leading to the cell locking into the ON state. In order to reinforce the switching in all the cells in the quorum, this circuit contains a method to communicate to other cells through the 3OC6HSL signal which can diffuse through the cellular membrane and into the medium to be taken up by other cells and affect their quorum trigger circuits.

Figures 27, 28, and 29 present the results of applying the iSSA to the quorum trigger while varying the basal rate of production. In Fig. 27, this rate is set to 0 and the iSSA predicts that the circuit does not switch ON when this is the case. When the rate is increased to 0.0001 as in Fig. 28, the iSSA shows that the circuit only switches when the environmental signal is set to a high value. However, the iSSA always predicts that the circuit switches ON when the basal rate is too high as shown in Fig. 29.

Stochastic model checking analysis of the quorum trigger circuit is able to confirm the predictions made by the iSSA and give some insight into the likelihood of the circuit switching ON for various values of k_b. Figure 30 shows that the probability that the circuit switches when the value of k_b is 0 meaning that without leakage in production, the circuit cannot switch. Figure 31, on the other hand, shows that the circuit switches ON after 10,000 s with a probability of about 8 % for a low environmental signal and with a probability of about 70 % for a high environmental signal. This difference is desirable as there should be a marked difference in how the circuit responds in the presence of the environmental signal. Confirming the iSSA analysis, having a basal rate that is 0.01 causes the circuit to switch ON with 100 % probability regardless of the

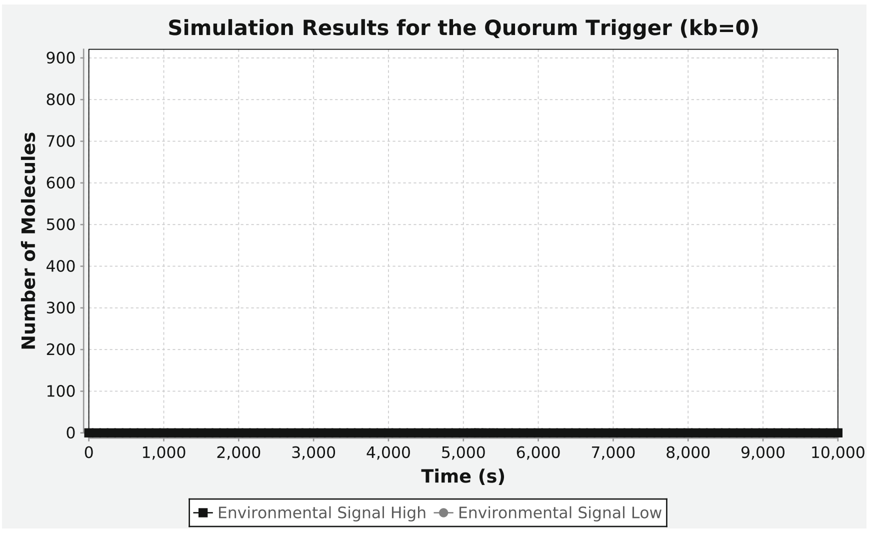


Fig. 27 iSSA simulation results for the quorum trigger when the basal rate of production, k_b, is set to 0. This plot shows that the circuit is unable to switch ON without some leakage in production

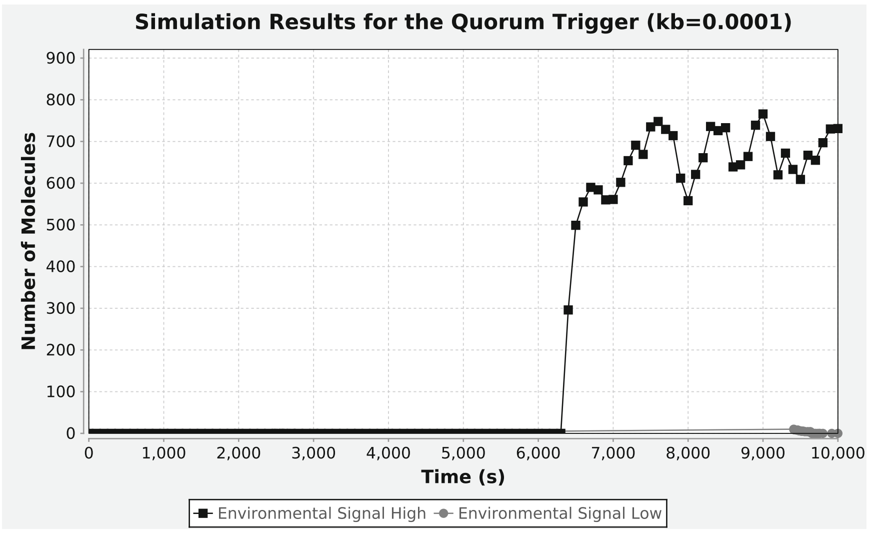


Fig. 28 iSSA simulation results for the quorum trigger when the basal rate of production, k_b, is set to 0.0001. This plot shows that the circuit only switches ON when the environmental signal is high for a low basal rate

value of the environmental signal as is seen in Fig. 32. The stochastic model checking results are obtained with run-times of 2 s when k_b is 0, 20 s when k_b is 0.0001, and 20 s when k_b is 0.01. These analyses show that a designer of this circuit would need to carefully tune the basal rate of production to be a value somewhere in the neighborhood of 0.0001 in order to ensure that the circuit does not switch ON too early but also does switch ON in a medium where the environmental signal is present.

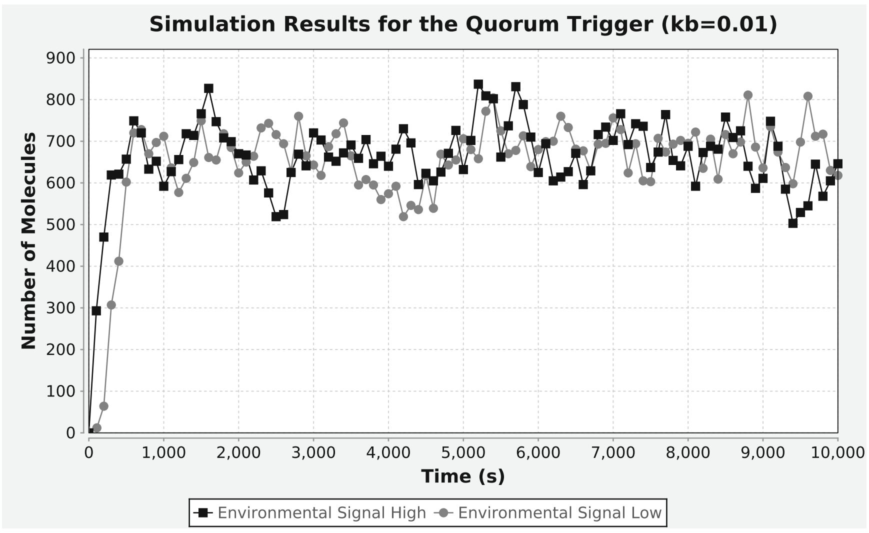


Fig. 29 iSSA simulation results for the quorum trigger when the basal rate of production, k_b, is set to 0.01. This plot shows that the circuit always switches ON due to the basal rate being too high

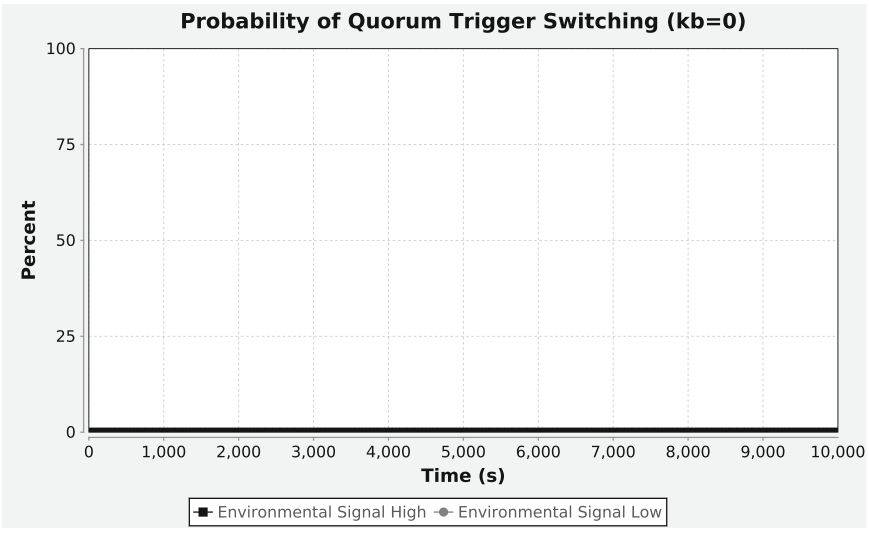


Fig. 30 Stochastic model checking results for the quorum trigger when the basal rate of production, k_b, is set to 0. These results confirm that the quorum trigger is unable to switch ON when the basal rate is 0

9 Conclusions and Future Work

Synthetic biology has the potential to allow researchers and scientists to design biological systems to solve a variety of problems. However, in order to design these systems, efficient methods to perform design space exploration through analysis and verification

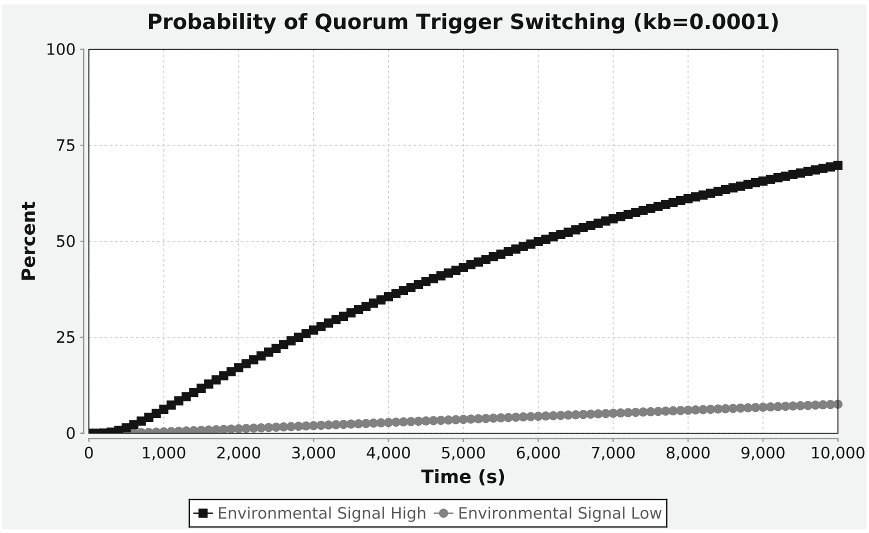


Fig. 31 Stochastic model checking results for the quorum trigger when the basal rate of production, k_b, is set to 0.0001. These results confirm that the quorum trigger is only able to switch ON when the environmental signal is high for a medium value of the basal rate

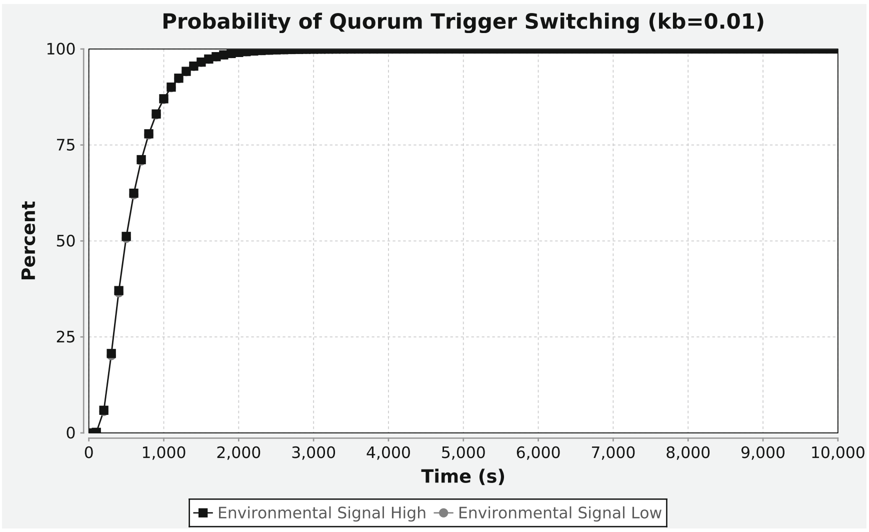


Fig. 32 Stochastic model checking results for the quorum trigger when the basal rate of production, k_b, is set to 0.01. These results confirm that the quorum trigger always switches ON if the basal rate is too high

of computational models are necessary. This chapter proposes one such methodology that can greatly aid synthetic biologists in the design process. The work described in this chapter has been integrated into the tool `iBioSim` [23, 24, 30]. This tool is freely available at http://www.async.ece.utah.edu/iBioSim/.

The iSSA delivers information about a genetic circuit that can often be hidden when using other simulation methods. By performing simulations in small time increments, it is capable of capturing important stochastic events that lead to the circuit exhibiting its "typical" behavior. Stochastic model checking translates the infinite state space of a genetic circuit into a CTMC. This CTMC can be analyzed along with a CSL property using stochastic model checking via steady state or transient Markov chain analysis. When utilized together, these methods can be used to simulate a genetic circuit with the iSSA, to generate a CSL property that captures the observed "typical" behavior or a rare behavior, and then to apply stochastic model checking to the circuit yielding the likelihood of observing typical and rare behaviors. If the desired behavior is not observed with a large enough probability, then the designer can alter elements of the circuit or parameters in the system to refine the genetic circuit. This chapter has applied the design space exploration methodology to several examples of genetic oscillators and state holding gates allowing different design and parameter choices to be efficiently analyzed and considered for each model.

Despite the utility of the proposed design space exploration methodology, there are still many ways that it can be improved. Currently, in order to select good levels, a user performs a small number of simulation runs to determine a range of values of interest. Automating the level selection process may prove to be fruitful as it is very difficult for a user to know where the levels should be placed. Additionally, the proposed methodology requires a user to analyze the resulting trace produced by the iSSA by hand in order to generate a property that captures the circuits typical behavior. A big improvement to this methodology would be the development of a process that could automatically analyze a trace and produce a CSL property that represents the behavior of the trace. Models of genetic circuits often allow for timed events where a species value can be changed at a predetermined time in simulation. Another improvement could be to extend the CTMC that is generated from the conversion process to include decision transitions for these environment changes using a *Markov decision process* (MDP) [9]. The efficiency of the stochastic model checking is also greatly impacted by the size of the model's state space. One way to deal with this problem is to apply partial order reduction to the CTMC to try to eliminate uninteresting intermediate states in the state graph. Finally, this chapter primarily focuses on stochastic analysis of synthetic genetic circuits; however, other systems that could lend themselves to this type of analysis include signal transduction pathways and metabolic networks.

Acknowledgments

The authors would like to thank Eduardo Monzon and Abiezer Tejeda for their work on improvements to the iSSA in its early stages of development.

This material is based upon work supported by the National Science Foundation under Grant Numbers 0331270, CCF-07377655, CCF-0916042, and CCF-1218095. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

1. Anderson JC, Clarke EJ, Arkin AP (2006) Environmentally controlled invasion of cancer cells by engineering bacteria. J Mol Biol 355:619–627
2. Arkin A (2008) Setting the standard in synthetic biology. Nature Biotech 26:771–774
3. Atsumi S, Liao JC (2008) Metabolic engineering for advanced biofuels production from Escherichia coli. Curr Opin Biotechnol 19(5):414–419. Tissue, cell and pathway engineering
4. Aziz A, Sanwal K, Singhal V, Brayton R (2000) Model-checking continuous-time Markov chains. ACM Trans Comput Logic 1:162–170
5. Cao Y, Gillespie DT, Petzold LR (2005) The slow-scale stochastic simulation algorithm. J Chem Phys 122:1
6. Cases I, de Lorenzo V (2005) Genetically modified organisms for the environment: stories of success and failure and what we have learned from them. Int Microbiol 8:213–222
7. Elowitz M, Leibler S (2000) A synthetic oscillatory network of transcriptional regulators. Nature 403(6767):335–338
8. Endy D (2005) Foundations for engineering biology. Nature 438:449–453
9. Feinberg E, Shwartz A (eds) (2002) Handbook of Markov decision processes - methods and applications. Kluwer International Series, Boston
10. Gardner TS, Cantor CR, Collins JJ (2000) Construction of a genetic toggle switch in *Escherichia coli*. Nature 403:339–342
11. 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
12. Gillespie DT (1976) A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. J Comput Phys 22(4):403–434
13. Gillespie DT (1977) Exact stochastic simulation of coupled chemical reactions. J Phys Chem 81(25):2340–2361
14. Gillespie DT (1992) Markov Processes: an introduction for physical scientists. Academic Press, New York
15. Gillespie DT, Petzold LR (2003) Tau leaping. J Chem Phys 119:8229–8234
16. 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 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
17. Kuwahara H (2007) Model abstraction and temporal behavior analysis of genetic regulatory networks. PhD thesis, University of Utah
18. Kuwahara H, Mura I (2008) An efficient and exact stochastic simulation method to analyze rare events in biochemical systems. J Chem Phys 129:16
19. Kuwahara H, Myers C, Barker N, Samoilov M, Arkin A (2006) Automated abstraction methodology for genetic regulatory networks. Trans Comput Syst Biol VI 4220:150–175
20. Kuwahara H, Madsen C, Mura I, Myers C, Tejada A, Winstead C (2010) Efficient stochastic simulation to analyze targeted properties of biological systems. In: Myers C (ed) Stochastic control. Sciyo, pp 505–532 http://www.intechopen.com
21. 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 (tutorial volume), vol 4486. Springer,Berlin, pp 220–270

22. MacQueen JB (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of 5-th Berkeley symposium on mathematical statistics and probability, Berkeley, vol 1, pp 281–297
23. Madsen C, Myers CJ, Patterson T, Roehner N, Stevens JT, Winstead C (2012) Design and test of genetic circuits using iBioSim. IEEE Des Test Comput 29(3):32–39
24. Myers CJ, Barker N, Jones K, Kuwahara H, Madsen C, Nguyen N-PD (2009) iBioSim: a tool for the analysis and design of genetic circuits. Bioinformatics 25(21):2848–2849
25. Nguyen N (2008) Design and analysis of genetic circuits. Master's thesis, University of Utah
26. Nguyen N, Kuwahara H, Myers C, Keener J (March 2007) The design of a genetic muller c-element. In: The 13th IEEE international symposium on asynchronous circuits and systems
27. Press WH, Flannery BP, Teukolsky SA, Vetterling WT (1992) Numerical recipes in C: the art of scientific computing, 2nd edn. Cambridge University Press, Cambridge
28. Ro D-K, Paradise EM, Ouellet M, Fisher KJ, Newman KL, Ndungu JM, Ho KA, Eachus RA, Ham TS, Kirby J, Chang MCY, Withers ST, Shiba Y, Sarpong R, Keasling JD (2006) Production of the antimalarial drug precursor artemisinic acid in engineered yeast. Nature 440:940–943
29. Slepoy A, Thompson AP, Plimpton SJ (2008) A constant-time kinetic monte carlo algorithm for simulation of large biochemical reaction networks. J Chem Phys 128(20):205101
30. Stevens JT, Myers CJ (2012) Dynamic modeling of cellular populations within iBioSim. ACS Synth Biol 2(5):223–229
31. Stewart WJ (1994) Introduction to the numerical solution of Markov chains. Princeton University Press, Princeton
32. Stricker J, Cookson S, Bennett M, Mather W, Tsimring L, Hasty J (2008) A fast, robust and tunable synthetic gene oscillator. Nature 456:516–519
33. Winstead C, Madsen C, Myers CJ (2010) iSSA: an incremental stochastic simulation algorithm for genetic circuits. In: International symposium on circuits and systems (ISCAS). IEEE, pp 553–556. Paris, France
34. Younes H, Kwiatkowska M, Norman G, Parker D (2006) Numerical vs. statistical probabilistic model checking. Int J Softw Tools Technol Transf 8:216–228

Chapter 12

Using Computational Modeling and Experimental Synthetic Perturbations to Probe Biological Circuits

Joshua R. Porter and Eric Batchelor

Abstract

This chapter describes approaches for using computational modeling of synthetic biology perturbations to analyze endogenous biological circuits, with a particular focus on signaling and metabolic pathways. We describe a bottom-up approach in which ordinary differential equations are constructed to model the core interactions of a pathway of interest. We then discuss methods for modeling synthetic perturbations that can be used to investigate properties of the natural circuit. Keeping in mind the importance of the interplay between modeling and experimentation, we next describe experimental methods for constructing synthetic perturbations to test the computational predictions. Finally, we present a case study of the p53 tumor-suppressor pathway, illustrating the process of modeling the core network, designing informative synthetic perturbations in silico, and testing the predictions in vivo.

Key words Synthetic biology, Dynamical systems, Computational modeling, Experimental design, Signal transduction, Metabolism

1 Introduction

Synthetic biology provides a powerful approach for generating novel biological devices that operate on a molecular scale. These devices have most often been developed using genetically tractable microorganisms as a chassis into which biological parts orthogonal to the host organism can be engineered. Most synthetic biology efforts thus far have been focused on one of two broad tasks: creating small-scale devices that perform a desired computation or function, or engineering the metabolism of a microorganism for the synthesis of useful compounds. Predictive computational models have been instrumental in designing synthetic circuits that achieve these goals.

Based on the principle "What I cannot create, I do not understand" (Richard Feynman), synthetic biologists seek to design and experimentally implement circuits that produce a desired, predicted behavior. This, in turn, helps one understand how natural biological circuits generate such behaviors mechanistically. Several proof-of-principle devices have been constructed to perform simple tasks, including generating oscillatory outputs [1] and performing logic computations [2]. Quantitative modeling has aided the construction of even the simplest circuits during both initial circuit design and circuit refinement. For example, modeling the repressilator [1] indicated that the individual protein components of the circuit needed to be rapidly degraded. Accordingly, the addition of protein degradation tags was necessary to implement the circuit successfully in vivo. Subsequent, more detailed modeling has led to further improvement of minimal transcriptional oscillators [3, 4].

Metabolic pathways are complex even in the simplest organisms; therefore, manipulation of such pathways for novel purposes can lead to non-intuitive, unanticipated results. Quantitative models of metabolic pathways are required not only to identify the best avenues for pathway manipulation but also to interpret the large amounts of data generated by metabolic studies. Computational approaches, including flux balance analysis, have proven invaluable for a wide range of such applications, from identifying a naturally occurring secondary metabolite and increasing its production [5] to improving methods for synthesizing heterologous compounds such as anti-malarial drugs [6].

In addition to providing methods for constructing new biological devices and engineering novel metabolites, synthetic biology provides a means of analyzing complex naturally occurring cellular networks. By making directed synthetic perturbations, one can probe the structure and function of endogenous signaling and metabolic pathways. At the simplest level, such perturbations might involve genetic knockouts, transcript knockdowns by RNA interference, overexpression of pathway components, or temperature-sensitive mutations of proteins. Such perturbations have been widely used by biologists for decades. More complex perturbations can enable a more accurate probing of natural circuit function, allowing one to plug in new network connections, bypass or eliminate existing connections, or tune interactions between circuit parts. Such approaches include mutation of transcription factor binding sequences at target promoters, construction of hybrid allosteric switches in proteins, or introduction of novel multicomponent network feedbacks.

In this chapter, we describe general methods for using synthetic approaches to probe the function of endogenous signaling and metabolic networks, focusing on the role of computational modeling in this process. We detail procedures for defining the natural cellular circuit of interest, constructing a computational model of the natural circuit, identifying possible perturbations to generate a desired novel output or probe regulatory connections, and designing experiments to test in vivo the perturbations designed in silico.

2 Methods

2.1 Modeling the Natural Circuit

Before one can reengineer a natural biological system, one must have sufficient quantitative understanding of how the natural system works. The best way to express and assess this quantitative understanding is to recreate the system in silico with a mathematical model.

Here we take a "bottom-up" modeling approach, which involves experimentally examining individual parts of a system and their interactions as modules, then combining the tractable modules to construct a larger, more complete model. This approach contrasts with "top-down" modeling techniques, in which measurements of many biological quantities under different perturbations of the system are used to infer the underlying system structure (*see* ref. 7 for one review). Top-down modeling techniques assume by default that the measurements provided are sufficient to identify what matters in the structure of a system. By contrast, the bottom-up approach does not assume this, and it has been our experience that the process of bottom-up modeling identifies which important details are still unknown, enabling a more systematic and comprehensive analysis of the key components of a system. For this reason, we believe that bottom-up modeling is better for building the sort of model required to re-engineer a biological system, as such systems frequently contain redundancies that are not apparent during normal system operation but may make a difference when the system is synthetically modified.

Within bottom-up modeling, we will also restrict our discussion to systems in which all chemical species involved are present in large numbers (on the order of 10^2–10^3 or greater) and can be assumed to be well mixed within their containing compartment(s) [8]. Such systems can reasonably be modeled using ordinary differential equations (ODEs), which are relatively simple to construct and solve. Systems with some chemical species present in very small numbers are best described using stochastic models; systems in which diffusion through space is a concern can be modeled using partial differential equations. These more complex types of models are useful but beyond the scope of this discussion.

The first step in modeling a biological system from the bottom up is to identify the relevant known parts of the system of interest. We define a part as a distinct biochemical species located in a

Table 1
Processes of interest in a biological system and their representations in a mathematical model

Process	Diagram	Rate
Binding	$X + Y \rightarrow XY$	$k_b[X][Y]$
Unbinding	$XY \rightarrow X + Y$	$k_u[XY]$
Production (constant)	$\rightarrow X$	k_{pX}
Degradation	$X \rightarrow \emptyset$ (nothing)	$k_{dX}[X]$
Catalysis	$\overset{E}{\overset{\downarrow}{S \rightarrow P}}$	$\frac{k_{cat}[E][S]}{K_M + [S]}$
Catalysis with competitive inhibition	$\overset{E}{S \xrightarrow{I \dashv \downarrow} P}$	$\frac{k_{cat}[E][S]}{K_M\left(1 + \frac{[I]}{K_I}\right) + [S]}$
Catalysis with noncompetitive inhibition	$\overset{E}{S \xrightarrow{I \dashv \downarrow} P}$	$\frac{k_{cat}[E][S]}{(K_M + [S])\left(1 + \frac{[I]}{K_I}\right)}$
Passive transport	$X_A \leftrightarrow X_B$	$k_T([X_B] - [X_A])$
Dilution due to exponential growth		$k_{dil}[X]$, $k_{dil} = \frac{\ln(2)}{t_d}$ where t_d is doubling time

particular reaction compartment. We then identify the processes in the system that change the concentration of the individual parts.

Table 1 lists common processes that are important in signal transduction and metabolic pathways. The most fundamental processes of interest are two parts binding to form a complex and the complex unbinding to form its constituent parts. Binding and unbinding processes can be combined in complex ways to form higher-level processes such as production, degradation, and enzyme catalysis. Other important processes include passive transport, the diffusion of parts down a concentration gradient between compartments, and the dilution of parts due to the exponential growth of their containing compartments. As a general rule, the less detail one understands about a process, the higher level an abstraction one uses to model it.

Once parts and processes have been identified, this information can be organized by combining it into a diagram using the conventional notation of symbols connected by arrows, as shown in Table 1. A simple diagram can be a powerful tool for understanding a biological system beyond the reductionist perspective of considering one part at a time, while at the same time it provides a nonmathematical description of a system that is accessible to a nontechnical audience.

When parts and processes have been identified and organized, the next step is to translate this information into a set of differential equations. In general, for every part in the system, this will yield one differential equation of the form $\frac{\mathrm{d[part]}}{\mathrm{d}t} = \sum \text{process rates}$, where [part] is the concentration of the part and the right side of the equation is a summation of terms for the rate of each process acting on that part. Table 1 shows these rate terms for each of the common processes listed.

In general, the system of study will be of sufficient complexity that the set of differential equations describing it will not have an analytical solution. Instead, a numerical solution must be obtained. Several numerical computing packages, including MATLAB and Mathematica, can readily solve such systems of differential equations. Using the ODE solvers in these packages simply requires writing the differential equations in the appropriate coding format and specifying initial conditions for the concentration of each part. Several other software packages, including little b [9], BioNetGen [10], Kappa [11] (*see* Chapter 6), Simmune [12], and PySB [13], have been written specifically for the analysis of biochemical networks. These packages help to automate and organize the process of model-building, which is increasingly useful as more complex systems are considered. In general, the various solvers can be used to explore both steady-state and dynamical behavior of biological circuits in silico.

Biochemical systems are often *stiff*, meaning that they are composed of processes that operate on different time scales. In particular, binding and unbinding processes are generally much faster than production, degradation, catalysis, transport, etc. Stiff systems require special techniques to solve computationally in an efficient manner. One convenient solution is to perform a separation of time scales, in which one considers the faster processes to be at steady state. This simplification can reduce both the number of equations in the model and the number of parameters required, as a single dissociation constant can replace two rate constants for binding and unbinding. As a result, a numerical solution can be computed faster, and in some cases an analytical solution can be determined.

Finding realistic parameter values is often one of the most challenging aspects of model building. The best choice for a parameter is the one connected as closely as possible with the simplest, best understood physical reality. Ideally, this would be obtained from a direct biochemical measurement. However, biochemical parameters are often not available in the literature and cannot always be directly measured by the researchers performing the modeling. In such cases, one must choose parameters such that the solution of the equations matches experimental observations. This is one reason why a bottom-up approach to model construction is often more feasible than a top-down approach, which is by nature dependent on a greater number of unknown parameters. As more biochemical details become known, the model can be refined with additional complexity as needed.

One can assess the quality of a model by how well its solution matches experimental observations. It is important to remember that models are not "right" or "wrong" so much as they are more or less useful—any model is a deliberately simplified representation of reality. The chances that a model will be useful can be improved by testing it against different experimental observations, ideally including some observations that were not used to determine parameters.

2.2 Probing Possible Synthetic Changes In Silico

Having constructed a computational model of the natural circuit of interest, one can use the model to make testable predictions about circuit performance. Confirming these predictions experimentally validates one's understanding of the circuit. When designing a synthetic circuit for a specific engineering task, a model is useful for identifying necessary connections and parameter operating regimes that give rise to a desired functional output. Similarly, when studying a natural circuit, one can use the model to identify the parts or parameter values necessary for a desired biological outcome. How dependent is the yield of a metabolite on the degradation rate of a metabolic intermediate? If there were a negative feedback on the production of phosphatase A, would signaling through kinase B be attenuated at a later point? Such questions can readily be explored in silico and experimentally tested in vivo with synthetic perturbations.

We can consider synthetic perturbations as falling into the following broad categories:

Synthesis Perturbations: The synthesis rates of individual parts are modified, or novel methods for increasing the (real or effective) concentration of parts are generated.

Degradation Perturbations: The degradation rates of parts are modified, or new methods for decreasing the (real or effective) concentration of parts are generated.

Interaction Perturbations: The interactions between natural parts are modified.

Novel Regulatory Connections: New processes (possibly involving additional synthetic parts) connecting existing parts are generated. These new processes can be broadly classified as positive feedback, negative feedback, coherent feedforward, or incoherent feedforward based on which parts influence which other parts and whether that influence is activating or repressing (Fig. 1) [14].

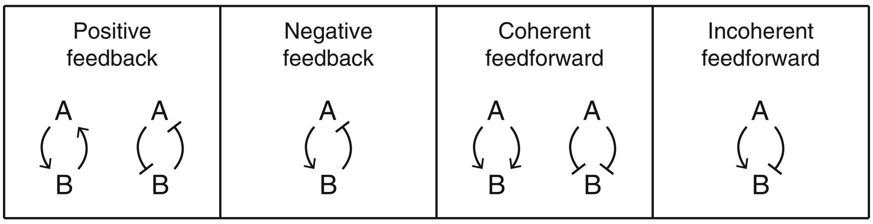


Fig. 1 Basic network structures. A → B means "A activates B"; A ⊣ B means "A inhibits B." Each → or ⊣ symbol represents at least one process, and possibly additional parts, by which part A influences part B or vice versa

Once a change to the system is conceived, it can be tested in the model by altering parameters or adding and removing rate terms as described earlier and simulating the new model.

During the process of model exploration, it is important to keep in mind which types of synthetic manipulations are experimentally feasible; otherwise, the model predictions will remain untestable hypotheses and little will be learned about the natural biological circuit. It is also important to keep synthetic perturbations as simple as possible. While a Rube Goldberg machine may look exciting on paper, it rarely yields informative results in the laboratory.

2.3 Implementing Changes In Vivo

Having developed a quantitative model of the biological circuit of interest and identified possible perturbations that will yield insight into the function of the circuit, one can perturb the actual circuit using a variety of experimental techniques. Several synthetic biology tools are readily available to modify a biological circuit at the level of individual genes, mRNAs, or proteins, and these perturbations can be combined to generate multi-component regulatory loops and modules. Here we briefly mention a few such tools for perturbing signaling or metabolic pathways.

The introduction of synthetic perturbations often depends on precise manipulation of a model organism's genome. Several methods are available for deleting or inserting DNA sequences. Random insertion of genetic material using electroporation, lipid-based transfection, or viral-mediated infection procedures is relatively easy. Directed genome manipulations can present more of a challenge, and the difficulty tends to scale with the complexity of the model organism being studied. Regardless of the target, the basic techniques are similar. The most widely used methods for directed editing are the phage recombinase systems, including lambda Red, FLP-FRT, and Cre/*lox* recombination [15, 16]. Recent advances in the use of engineered nucleases, such as TALENs and zinc-finger nucleases [17], offer other powerful methods for genome editing in a variety of organisms.

It is often desirable to control the synthesis rates of biological parts when testing model predictions. This control can be performed in vivo by altering expression of natural or synthetic parts through the use of regulatable promoters. Several different inducible or repressible expression systems are available from prokaryotic or eukaryotic origins. The well-characterized *lac*- and *ara*-inducible systems from bacteria are regulated by the presence of sugars and sugar analogs such as isopropyl β-D-1-thiogalactopyranoside (IPTG) [18]. The *GAL1* and *CUP1* promoters, which are induced in response to galactose and copper, respectively, are examples of systems effective in yeast [19]. In mammalian cells, one of the most widely used systems is the tetracycline-responsive system [20]. This system is a particularly effective tool in that versions for induction in either the presence or the absence of tetracycline/doxycycline have been developed, and they can be used in a variety of organisms. These are but a few of the promoters available, and they all provide powerful synthetic methods for regulating both the level and the timing of gene expression.

For changing the degradation rates of endogenous mRNAs, one can use RNA interference (RNAi) technologies. Originally developed in *C. elegans* [21] and now widely used in mammalian cell culture, RNAi can be used for transient knockdown of target mRNAs in a variety of systems. For example, transfection of small interfering RNAs (siRNAs) into mammalian cells enables the knockdown of endogenous mRNAs for up to several days. Long-term and regulatable knockdown of mRNAs can be achieved by engineering small hairpin RNAs (shRNAs), expressed from either constitutive or inducible promoters, directly into the model system's genome.

The degradation rate of proteins can also be modified synthetically. The addition of degradation tags such as PEST sequences [22] can shorten protein lifetimes. One notable way to regulate protein degradation more tightly is to use the *ssrA* proteasomal machinery from *E. coli* in an orthogonal organism [23]. By altering the concentration of the degradation machinery via an inducible promoter, this system enables tunable regulation of degradation rates for specific proteins within a network of interest.

Beyond direct control of protein degradation, it is possible to synthetically alter the effective concentration of a protein. In particular, changing protein localization can be a means of changing protein concentration in a subcellular compartment of interest. For example, mutating nuclear localization sequences or nuclear exclusion signals can alter protein transport into or out of the nucleus. This synthetic manipulation may be particularly useful for eukaryotic transcription factors, as their gene regulatory activity depends on their nuclear localization.

The next major class of synthetic perturbations one can consider is altering the interactions between endogenous cellular parts. Many mechanisms exist for making such alterations. For example, manipulating organelle localization tags and membrane tethers allows additional regulation of protein interactions by altering the effective protein concentration. Such changes can also be effected by using modified scaffold proteins or direct synthetic linkage of protein domains to increase the probability of interactions occurring. For example, Bashor et al. [24] used a scaffold modification approach to dissect the function of the yeast mating MAPK pathway in *S. cerevisiae*. Linking proteins or protein domains is also an effective way to generate novel allosteric switches for a variety of purposes.

Recent developments in optogenetics also hold great promise as tools for synthetic manipulation of biological circuits. Optogenetics was originally developed as a method to control nerve cell activation noninvasively using light-activated ion channels [25]. Recently, additional photoreceptor domains, such as the LOV domain from *A. sativa*, have been engineered to control signaling molecules, providing precise temporal and spatial control of protein activation. For example, Wu et al. [26] fused the LOV domain to forms of the GTPase Rac1, creating a hybrid protein that could modulate cell motility in a reversible and repeatable manner.

Ultimately, synthetic manipulations at the genetic, mRNA, and protein levels can be combined to engineer large-scale, multicomponent network connections. For example, new feedback and feedforward motifs can be wired into natural signaling circuits to control the magnitude or duration of a signal. The possibilities for generating such manipulations experimentally are limitless, although they depend on the natural circuit one is interested in modifying. For illustrative purposes, we will describe an example of such rewirings in the following case study.

3 Case Study: The p53 Signaling Pathway

The p53 signaling pathway in human cells has been extensively studied, as it plays a critical role in preventing tumor formation. p53 is a transcription factor that is activated by different forms of cell stress and regulates the expression of over a hundred genes, impacting several different processes including DNA repair, apoptosis, cell cycle arrest, and senescence [27]. One gene upregulated by p53 codes for the E3 ubiquitin ligase Mdm2, which tags p53 for degradation, thereby forming a negative feedback loop (Fig. 2a). When cells are γ-irradiated to generate DNA double-strand breaks (DSBs), the concentration of p53 and Mdm2 in the nucleus changes in a series of oscillatory pulses characterized by fixed amplitude and timing [28].

Toettcher et al. [29] sought to modify the p53 pathway to alter the expression dynamics of p53 and Mdm2. The goal of

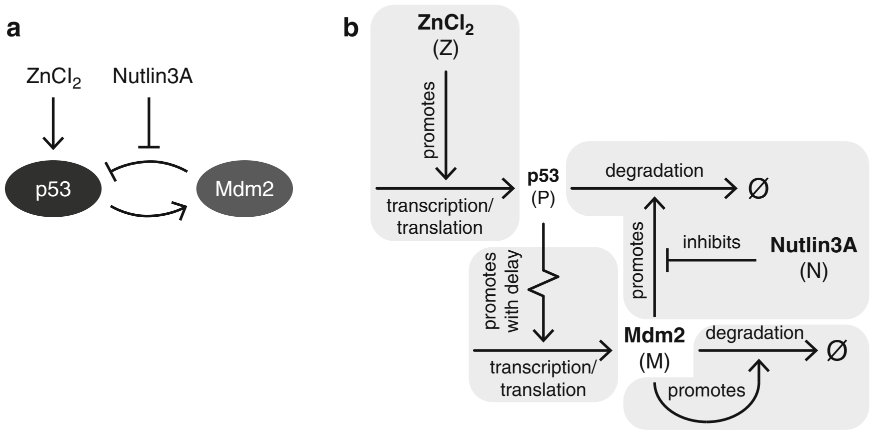


Fig. 2 Diagrams of the p53 signaling pathway. (**a**) A general depiction of the interactions between p53 and Mdm2 and how those interactions are modulated by external chemical signals, adapted from Toettcher et al. [29]. (**b**) A more detailed diagram, showing specific parts and processes, is often more useful for building a model

these modifications was to better understand the mechanisms generating the dynamics and to gain insight into their functional consequences. Since treating cells with γ-radiation triggers a complex response involving a larger network of components, they first sought a way to activate p53 that would bypass this complexity. This bypass took the form of a cell line in which p53 fused to CFP was expressed from the zinc-inducible rat metallothionein promoter. Using this synthetic system enabled direct manipulation of *p53* transcription independent of the endogenous *p53* promoter. Treating the cells with different concentrations of zinc chloride ($ZnCl_2$) maintained the oscillatory dynamics of p53 and Mdm2 expression, although the oscillations were damped, in contrast to the undamped oscillations observed in the response to γ-radiation. In this way, Toettcher et al. were able to use a synthetic bypass of the full natural DSB response, effectively narrowing the scope of the biological system they needed to consider.

Having established a simplified experimental system with which to study the core process of interest (Fig. 2a), Toettcher et al. next constructed a computational model to explore additional possible synthetic perturbations. The parts of the simplified system, in addition to p53 and Mdm2, included $ZnCl_2$, which promotes p53 production, and Nutlin3A, a small molecule that inhibits the interaction between Mdm2 and p53. Four key processes connected these four biochemical parts. First, p53 is produced at a constant rate plus an additional zinc-dependent rate. Second, Mdm2 catalyzes the ubiquitination of p53, tagging it for subsequent degradation; this catalysis is competitively inhibited by Nutlin3A. Third, p53 promotes the transcription of the *mdm2* gene, leading to an increased Mdm2 protein concentration with a delay due to protein translation, folding, and nuclear localization. Fourth, Mdm2 catalyzes its own ubiquitination, leading to its own degradation. Figure 2b graphically summarizes these parts and processes.

The next step was to translate this information about parts and processes into a set of differential equations:

p53 protein:	$\frac{d[P]}{dt} = \underbrace{\xi_p(t)}_{\text{noise}} \left(\underbrace{\alpha_p + \frac{p_z[Z]^3}{K_z^3 + [Z]^3}}_{\text{ZnCl}_2\text{-dependent production}} \right) - \underbrace{\frac{\delta_p[M][P]}{K_p(1+[N]) + [P]}}_{\text{Mdm2-mediated degradation}}$
mdm2 mRNA:	$\frac{d[M_0]}{dt} = \gamma_{m0} \left(\underbrace{\alpha_{m0} + \beta_{m0} \frac{[P]^2}{K_{m0}^2 + [P]^2}}_{\text{p53-upregulated transcription}} - \underbrace{[M_0]}_{\text{degradation}} \right)$
mdm2 mRNA → protein delays:	$\frac{d[M_i]}{dt} = \gamma_{m0}([M_{i-1}] - [M_i]), \quad i = 1 \ldots 4$
Mdm2 protein:	$\frac{d[M]}{dt} = \underbrace{\xi_m(t)[M_4]}_{\text{noisy production}} - \underbrace{\frac{\delta_m[M]}{K_m + [M]}}_{\text{autocatalytic degradation}} - \underbrace{\gamma_m[M]}_{\text{constant degradation}}$

Quantities described by these equations are defined in Table 2. Notably, the equations for p53 and Mdm2 protein incorporate the stochastic processes $\xi_p(t)$ and $\xi_m(t)$ to model noise in production of the p53 and Mdm2 proteins, respectively. To avoid using a stiff delay to model the time-delayed production of Mdm2, Toettcher et al. used a "boxcar" procedure of four separate equations representing intermediates in the processing and translation of *mdm2* mRNA (represented by the equations for $[M_i]$, $i = 1$–4) [29].

Toettcher et al. next found values for the model parameters. They first measured the relationship between zinc concentration and p53 transcription using a cell line in which production of CFP was driven by the same zinc-inducible promoter as that driving p53-CFP in this system. To obtain values for K_z and p_z, they fit a Hill function with $n = 3$ to the relationship between zinc and CFP fluorescence. For the remaining parameters, they used an optimization method to obtain parameter values that minimized the difference between simulated and experimental measurements of p53 and Mdm2 first pulse amplitude, p53 pulse frequency, and p53 pulse damping rate at five different concentrations of zinc. They further validated the optimal parameters by simulating the model

Table 2
Symbols and parameters used in the base model of p53 signaling

Symbol	Description
$[P]$	Concentration of p53 protein
$[M_0]$	Concentration of *mdm2* mRNA
$[M_i]$, $i = 1 \ldots 4$	Concentrations of intermediates in Mdm2 protein production
$[M]$	Concentration of Mdm2 protein
$[Z]$	Concentration of $ZnCl_2$
$[N]$	Concentration of Nutlin3A
Parameter	**Description**
$\xi_p(t)$	Noise in p53 protein production
α_p	p53 protein production rate
p_z	$ZnCl_2$-mediated p53 production rate
K_z	Saturation of $ZnCl_2$-mediated p53 production
δ_p	Mdm2-mediated p53 degradation rate
K_p	Saturation of Mdm2-mediated p53 degradation
γ_{m0}	Mdm2 production delay
α_{m0}	Basal transcription rate of *mdm2* mRNA
β_{m0}	p53-mediated transcription rate of *mdm2* mRNA
K_{m0}	Saturation of p53-mediated Mdm2 production
$\xi_m(t)$	Noise in Mdm2 protein production
δ_m	Mdm2-mediated Mdm2 degradation rate
K_m	Saturation of Mdm2-mediated Mdm2 degradation
γ_m	Mdm2 degradation rate

with noise and verifying that these simulations were consistent with experimental measurements.

Having constructed the base model, Toettcher et al. then sought to identify methods by which they could perturb various characteristics of p53 oscillations. They first simulated the addition of a synthetic positive (NPF, Fig. 3a) or negative (2NF, Fig. 3b) feedback loop to the core p53-Mdm2 network. Based on their knowledge of possible experimental perturbations, they modeled these synthetic loops as transcriptional targets of p53 that could increase or decrease p53 expression. To do this, they added equations describing concentrations of mRNA ($[F_0]$) and protein ($[F]$)

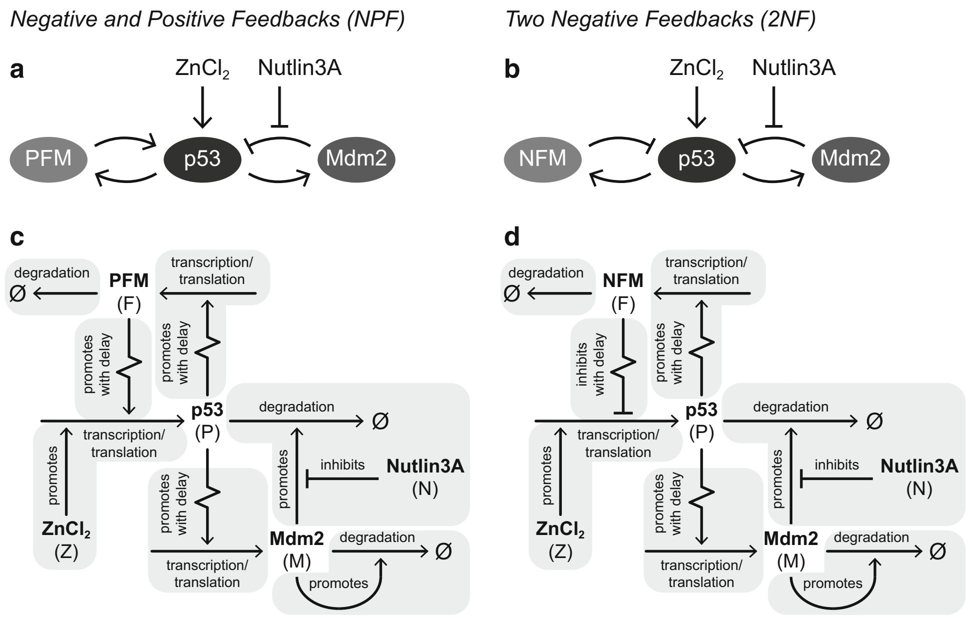


Fig. 3 Diagrams of two p53 signaling pathways with feedback loops added to the original pathway. (**a**) The NPF model contains an extra positive feedback loop: p53 promotes PFM (positive feedback mediator) production, which in turn promotes p53 production. Adapted from ref. 29. (**b**) The 2NF model contains an extra negative feedback loop: p53 promotes NFM (negative feedback mediator) production, which in turn inhibits p53 production. Adapted from ref. 29. (**c**) The detailed diagram of the NPF model includes one extra part (PFM) and three extra processes: p53-upregulated PFM production, PFM degradation, and PFM-upregulated p53 production. (**d**) The detailed diagram of the 2NF model is similar to that of the NPF model except that NFM inhibits rather than promotes p53 production

of a putative positive or negative feedback mediator (PFM or NFM), as well as equations modeling the delays between mRNA and protein ($[F_i]$, $i = 1 \ldots 4$) (Fig. 3c, d). Moreover, since p53 production was now part of a feedback loop, they made the model of p53 appropriately more complex, adding differential equations describing concentrations of p53 mRNA ($[P_0]$) and modeling the delays between mRNA and protein ($[P_i]$, $i = 1 \ldots 4$). The constant-rate degradation of Mdm2 was removed from the model, as the optimal value for γ_m in the old model was found to be 0, and the noise terms were removed as well to consider only the deterministic system. The new set of equations was nearly identical for the NPF and 2NF models, with the only difference being whether the PFM/NFM ($[F]$) activates or represses production of p53 mRNA.

p53 mRNA:	$\frac{d[P_0]}{dt} = \begin{cases} \gamma_{p0}\left(\underbrace{\alpha_{p0}+\beta_{p0}\left(\frac{[Z]^3}{K_z^3+[Z]^3}\right)\left(\frac{[F]^2}{K_f^2+[F]^2}\right)}_{\text{ZnCl}_2\text{-activated, feedback-activated transcription}} - \underbrace{[P_0]}_{\text{degradation}}\right) & \text{NPF model} \\ \gamma_{p0}\left(\underbrace{\alpha_{p0}+\beta_{p0}\left(\frac{[Z]^3}{K_z^3+[Z]^3}\right)\left(\frac{K_f^2}{K_f^2+[F]^2}\right)}_{\text{ZnCl}_2\text{-activated, feedback-repressed transcription}} - \underbrace{[P_0]}_{\text{degradation}}\right) & \text{2NF model} \end{cases}$
p53 mRNA → protein delays:	$\frac{d[P_i]}{dt} = \gamma_{p0}([P_{i-1}] - [P_i]), \quad i = 1\ldots4$
p53 protein:	$\frac{d[P]}{dt} = \underbrace{\alpha_p[P_4]}_{\text{production}} - \underbrace{\frac{\delta_p[M][P]}{K_p(1+[N])+[P]}}_{\text{Mdm2-mediated degradation}}$
mdm2 mRNA:	$\frac{d[M_0]}{dt} = \gamma_{m0}\left(\underbrace{\alpha_{m0}+\beta_{m0}\frac{[P]^2}{K_{m0}^2+[P]^2}}_{\text{p53-upregulated transcription}} - \underbrace{[M_0]}_{\text{degradation}}\right)$
Mdm2 mRNA → protein delays:	$\frac{d[M_i]}{dt} = \gamma_{m0}([M_{i-1}] - [M_i]), \quad i = 1\ldots4$
Mdm2 protein:	$\frac{d[M]}{dt} = \underbrace{[M_4]}_{\text{production}} - \underbrace{\frac{\delta_m[M]}{K_m+[M]}}_{\text{autocatalytic degradation}}$
Synthetic positive/ negative feedback mediator mRNA:	$\frac{d[F_0]}{dt} = \gamma_{f0}\left(\underbrace{\alpha_{f0}+\beta_{f0}\frac{[P]^2}{K_{f0}^2+[P]^2}}_{\text{p53-upregulated transcription}} - \underbrace{[F_0]}_{\text{degradation}}\right)$
Synthetic positive/ negative feedback mediator mRNA → protein delays:	$\frac{d[F_i]}{dt} = \gamma_{f0}([F_{i-1}] - [F_i])$
Synthetic positive/ negative feedback mediator protein:	$\frac{d[F]}{dt} = \underbrace{[F_4]}_{\text{production}} - \underbrace{\gamma_f[F]}_{\text{degradation}}$

All quantities described by these equations are defined in Table 3. In this modified model, Toettcher et al. kept the original model parameters governing the interactions between p53 and Mdm2, as this aspect of the model did not change. Using a range of feedback strengths and delay times, they simulated the model to probe the effects of the new feedbacks on p53 dynamics. They found that the additional feedback loops affected oscillation

Table 3
Symbols and parameters used in the modified model of p53 signaling

Symbol	Description
$[P_0]$	Concentration of *p53* mRNA
$[P_i]$, $i = 1 \ldots 4.$	Concentrations of intermediates in p53 protein production
$[P]$	Concentration of p53 protein
$[M_0]$	Concentration of *mdm2* mRNA
$[M_i]$, $i = 1 \ldots 4$	Concentrations of intermediates in Mdm2 protein production
$[M]$	Concentration of Mdm2 protein
$[F_0]$	Concentration of mRNA for synthetic positive/negative feedback mediator
$[F_i]$, $i = 1 \ldots 4$	Concentrations of intermediates in production of synthetic positive/negative feedback mediator
$[F]$	Concentration of synthetic positive/negative feedback mediator protein
$[Z]$	Concentration of $ZnCl_2$
$[N]$	Concentration of Nutlin3A
Parameter	**Description**
γ_{p0}	p53 production delay
α_{p0}	Basal transcription rate of *p53* mRNA
β_{p0}	Feedback-mediated transcription rate of *p53* mRNA
K_z	Saturation of $ZnCl_2$-mediated *p53* transcription
K_f	Saturation of feedback-mediated *p53* transcription
γ_{p0}	p53 production delay
α_p	p53 protein production rate
δ_p	Mdm2-mediated p53 degradation rate
K_p	Saturation of Mdm2-mediated p53 degradation
γ_{m0}	Mdm2 production delay
α_{m0}	Basal transcription rate of *mdm2* mRNA
β_{m0}	p53-mediated transcription rate of *mdm2* mRNA
K_{m0}	Saturation of p53-mediated *mdm2* transcription
δ_m	Mdm2-mediated Mdm2 degradation rate
K_m	Saturation of Mdm2-mediated Mdm2 degradation

(continued)

Table 3 (continued)

Symbol	Description
γ_{f0}	Production delay for synthetic positive/negative feedback mediator
α_{f0}	Basal transcription rate of mRNA for synthetic positive/ negative feedback mediator
β_{f0}	p53-mediated transcription rate of mRNA for synthetic positive/negative feedback mediator
K_{f0}	Saturation of p53-mediated transcription of synthetic positive/negative feedback mediator
γ_{f}	Degradation rate of synthetic positive/negative feedback mediator

amplitude and damping rate but not frequency. Further simulations revealed that changes to p53 pulse frequency could be obtained by modulating the concentration of Nutlin3A.

Toettcher et al. next established experimental systems to validate the predictions from the model of the synthetic-natural circuits. To generate a synthetic positive feedback loop, they constructed a cell line in which MTF1, the zinc-responsive transcription factor that acts directly on the zinc-inducible promoter, was expressed from a p53-responsive promoter (NPF, Fig. 3a). Thus, p53 upregulated MTF1 production, which in turn upregulated p53 production via the zinc-inducible promoter, forming the requisite synthetic positive feedback loop (Fig. 3c). In a separate design, to generate a synthetic negative feedback in addition to the core network, they used MTF1-KRAB, a dominant-negative form of MTF1, expressed from a p53-dependent promoter (2NF, Fig. 3b). In this cell line, p53 upregulated MTF1-KRAB production, and MTF1-KRAB repressed p53 production by competing with endogenous MTF1, forming a new negative feedback loop (Fig. 3d). Stimulation of these cells with $ZnCl_2$ confirmed that the damping rates of the oscillations, but not the frequency, were altered by the new feedback loops, as predicted by the computational model [29]. Additional experiments showed that perturbing the base experimental system with Nutlin3A altered oscillation frequency as predicted by the model [29].

4 Conclusions

While much attention in synthetic biology research has focused on engineering new biological devices, in this chapter we have described a method for using synthetic biology approaches to address questions in basic research. It is important to remember that the combined computational and experimental approach described here is iterative—experimental observations are used to build a quantitative model, which is used to generate new hypotheses to test experimentally, etc. It is hoped that through each iteration, one can better understand the network structure and the function of the natural biological circuit. For modeling and manipulating such circuits, the computational methods and experimental tools described here are by no means a complete catalog of the synthetic biology toolbox. However, they are broadly applicable for tackling a variety of problems in signal transduction and metabolic research, as illustrated in the p53 case study. As new methods and tools are developed, they can readily be applied to the general framework described here.

Acknowledgment

This work was supported by the Intramural Research Program of the Center for Cancer Research, National Cancer Institute, National Institutes of Health.

References

1. Elowitz MB, Leibler S (2000) A synthetic oscillatory network of transcriptional regulators. Nature 403(6767):335–338. doi:10.1038/35002125
2. Benenson Y (2012) Biomolecular computing systems: principles, progress and potential. Nat Rev Genet 13(7):455–468. doi:10.1038/nrg3197
3. Atkinson MR, Savageau MA, Myers JT, Ninfa AJ (2003) Development of genetic circuitry exhibiting toggle switch or oscillatory behavior in *Escherichia coli*. Cell 113(5):597–607. doi:10.1016/S0092-8674(03)00346-5
4. Stricker J, Cookson S, Bennett MR, Mather WH, Tsimring LS, Hasty J (2008) A fast, robust and tunable synthetic gene oscillator. Nature 456(7221):516–519. doi:10.1038/nature07389
5. Nguyen QT, Merlo ME, Medema MH, Jankevics A, Breitling R, Takano E (2012) Metabolomics methods for the synthetic biology of secondary metabolism. FEBS Lett 586(15):2177–2183. doi:10.1016/j.febslet.2012.02.008
6. Keasling JD (2012) Synthetic biology and the development of tools for metabolic engineering. Metab Eng 14(3):189–195. doi:10.1016/j.ymben.2012.01.004
7. Lee WP, Tzou WS (2009) Computational methods for discovering gene networks from expression data. Brief Bioinform 10(4):408–423. doi:10.1093/bib/bbp028
8. Chen WW, Niepel M, Sorger PK (2010) Classic and contemporary approaches to modeling biochemical reactions. Genes Dev 24(17):1861–1875. doi:10.1101/gad.1945410
9. Mallavarapu A, Thomson M, Ullian B, Gunawardena J (2009) Programming with models: modularity and abstraction provide powerful capabilities for systems biology. J R Soc Interface 6(32):257–270. doi:10.1098/rsif.2008.0205
10. Hlavacek WS, Faeder JR, Blinov ML, Posner RG, Hucka M, Fontana W (2006) Rules for modeling signal-transduction systems. Sci STKE 2006 (344):re6. doi:10.1126/stke.3442006re6
11. Danos V, Feret J, Fontana W, Harmer R, Krivine J (2008) Rule-based modeling, symmetries, refinements. In: Formal methods in systems biology, vol 5054, Lecture notes in bioinformatics 2008. Springer, Berlin, pp 103–122
12. Meier-Schellersheim M, Xu X, Angermann B, Kunkel EJ, Jin T, Germain RN (2006) Key role of local regulation in chemosensing revealed by a new molecular interaction-based modeling method. PLoS Comput Biol 2(7):e82. doi:10.1371/journal.pcbi.0020082

13. Lopez CF, Muhlich JL, Bachman JA, Sorger PK (2013) Programming biological models in Python using PySB. Mol Syst Biol 9:646. doi:10.1038/msb.2013.1
14. Alon U (2007) Network motifs: theory and experimental approaches. Nat Rev Genet 8 (6):450–461. doi:10.1038/nrg2102
15. Murphy KC (2012) Phage recombinases and their applications. Adv Virus Res 83:367–414. doi:10.1016/B978-0-12-394438-2.00008-6
16. Turan S, Zehe C, Kuehle J, Qiao J, Bode J (2012) Recombinase-mediated cassette exchange (RMCE)—a rapidly-expanding toolbox for targeted genomic modifications. Gene 515(1):1–27. doi:10.1016/j.gene.2012.11.016
17. Gaj T, Gersbach CA, Barbas CF 3rd (2013) ZFN, TALEN, and CRISPR/Cas-based methods for genome engineering. Trends Biotechnol 31(7):397–405. doi:10.1016/j.tibtech.2013.04.004
18. Terpe K (2006) Overview of bacterial expression systems for heterologous protein production: from molecular and biochemical fundamentals to commercial systems. Appl Microbiol Biotechnol 72(2):211–222. doi:10.1007/s00253-006-0465-8
19. Maya D, Quintero MJ, de la Cruz Munoz-Centeno M, Chavez S (2008) Systems for applied gene control in *Saccharomyces cerevisiae*. Biotechnol Lett 30(6):979–987. doi:10.1007/s10529-008-9647-z
20. Gossen M, Bujard H (1992) Tight control of gene expression in mammalian cells by tetracycline-responsive promoters. Proc Natl Acad Sci U S A 89(12):5547–5551
21. Fire A, Xu S, Montgomery MK, Kostas SA, Driver SE, Mello CC (1998) Potent and specific genetic interference by double-stranded RNA in *Caenorhabditis elegans*. Nature 391 (6669):806–811. doi:10.1038/35888
22. Rogers S, Wells R, Rechsteiner M (1986) Amino acid sequences common to rapidly degraded proteins: the PEST hypothesis. Science 234(4774):364–368
23. Grilly C, Stricker J, Pang WL, Bennett MR, Hasty J (2007) A synthetic gene network for tuning protein degradation in *Saccharomyces cerevisiae*. Mol Syst Biol 3:127. doi:10.1038/msb4100168
24. Bashor CJ, Helman NC, Yan S, Lim WA (2008) Using engineered scaffold interactions to reshape MAP kinase pathway signaling dynamics. Science 319(5869):1539–1543. doi:10.1126/science.1151153
25. Szobota S, McKenzie C, Janovjak H (2013) Optical control of ligand-gated ion channels. Methods Mol Biol 998:417–435. doi:10.1007/978-1-62703-351-0_32
26. Wu YI, Frey D, Lungu OI, Jaehrig A, Schlichting I, Kuhlman B, Hahn KM (2009) A genetically encoded photoactivatable Rac controls the motility of living cells. Nature 461 (7260):104–108. doi:10.1038/nature08241
27. Riley T, Sontag E, Chen P, Levine A (2008) Transcriptional control of human p53-regulated genes. Nat Rev Mol Cell Biol 9 (5):402–412
28. Geva-Zatorsky N, Rosenfeld N, Itzkovitz S, Milo R, Sigal A, Dekel E, Yarnitzky T, Liron Y, Polak P, Lahav G, Alon U (2006) Oscillations and variability in the p53 system. Mol Syst Biol 2(2006):0033
29. Toettcher JE, Mock C, Batchelor E, Loewer A, Lahav G (2010) A synthetic-natural hybrid oscillator in human cells. Proc Natl Acad Sci U S A 107(39):17047–17052. doi:10.1073/pnas.1005615107

Chapter 13

In Silico Control of Biomolecular Processes

Jannis Uhlendorf, Agnès Miermont, Thierry Delaveau, Gilles Charvin, François Fages, Samuel Bottani, Pascal Hersen, and Gregory Batt

Abstract

By implementing an external feedback loop one can tightly control the expression of a gene over many cell generations with quantitative accuracy. Controlling precisely the level of a protein of interest will be useful to probe quantitatively the dynamical properties of cellular processes and to drive complex, synthetically-engineered networks. In this chapter we describe a platform for real-time closed-loop control of gene expression in yeast that integrates microscopy for monitoring gene expression at the cell level, microfluidics to manipulate the cells environment, and original software for automated imaging, quantification, and model predictive control. By using an endogenous osmo-stress responsive promoter and playing with the osmolarity of the cells environment, we demonstrate that long-term control can indeed be achieved for both time-constant and time-varying target profiles, at the population level, and even at the single-cell level.

Key words Model predictive control, Gene expression, High-osmolarity glycerol (HOG) pathway, Computational biology, Quantitative systems and synthetic biology

1 Introduction

Understanding the information processing abilities of biological systems is a central problem for systems and synthetic biology [1–6]. The properties of a living system are often inferred from the observation of its response to perturbations. Currently it is not possible to control protein levels in a precise and time-varying manner, even though this would be instrumental in our understanding of gene regulatory networks. To deal with this problem, we present a novel experimental strategy to gain quantitative, real-time control on gene expression in vivo. We see the problem of manipulating gene expression to obtain given temporal profiles of protein levels as a model-based control problem. More precisely, we investigate the effectiveness of computerized closed-loop control strategies to control gene expression in vivo. In model based closed-loop control, a model of the system is used to constantly update the control strategy based on real-time observations.

We propose an experimental platform that implements such an *in silico* closed-loop in the budding yeast *Saccharomyces cerevisiae*. We show that gene expression can be controlled by repeatedly stimulating a native endogenous promoter over many cell generations for both time-constant and time-varying target profiles and at both the population and the single-cell levels.

2 Results

2.1 The Controlled System

We based our approach on the well-known response of yeast to an osmotic shock, which is mediated by the HOG (high osmolarity glycerol) signaling cascade. Its activation leads to the phosphorylation of the protein Hog1 (Fig. 1) which orchestrates cell adaptation through glycerol accumulation. Phosphorylated Hog1 promotes glycerol production by activating gene expression in the nucleus as well as by stimulating glycerol producing enzymes in the cytoplasm. Once adapted, the cells do not sense the hyperosmotic environment anymore, the HOG cascade is turned off and the transcriptional response stops [7–9]. In control terms, yeast cells implement several, short-term (non-transcriptional) and long-term (transcriptional) negative feedback loops (*see* Chapter 10) which ensure their perfect adaptation to the osmotic stress [10]. Because of these adaptation mechanisms, it is a priori challenging to control gene expression induced by osmotic stress. It is thus an excellent system to demonstrate that one can robustly control protein levels even in the presence of internal negative feedback loops. Several genes are

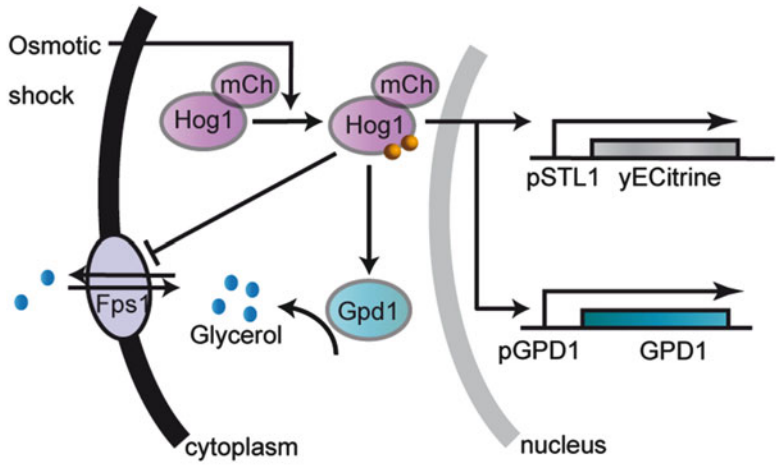


Fig. 1 Natural and engineered cell response to hyperosmotic shocks [14]. A hyperosmotic stress triggers the activation and nuclear translocation of Hog1. Short-term adaptation is mainly implemented by cytoplasmic activation of the glycerol-producing enzyme Gpd1 and closure of the aqua-glyceroporin channel Fps1. Long-term adaptation occurs primarily through the production of Gpd1. For our application, the expression of the protein of interest, yECitrine, is controlled by the osmo-responsive promoter pSTL1

up-regulated in response to a hyper osmotic stress. This includes the nonessential gene STL1 which codes for a glycerol proton symporter gene [12]. We decided to use its native promoter to drive the expression of yECitrine, a fluorescent reporter. Applying an osmotic stress transiently activated the HOG cascade and yECitrine levels reached modest values. Importantly, when short but repeated stresses were applied, pSTL1 could be repeatedly activated and much higher levels could be reached [14].

2.2 The Experimental Platform

To observe the cells and control their environment, we designed a versatile platform made of standard microscopy and microfluidic parts. The microfluidic device contained several 3.1 μm high chambers which were connected by both ends to large channels through which liquid media could be perfused (Fig. 2). Since the typical diameter of a *S. cerevisiae* cell is 4–5 μm, the cells were trapped in the chamber as a monolayer and their motion was limited to slow lateral displacement due to cell growth. This design allowed for long-term cell tracking (>15 h) and for relatively rapid media exchanges (~2 min). The HOG pathway was activated by switching between normal and sorbitol enriched (1 M) media.

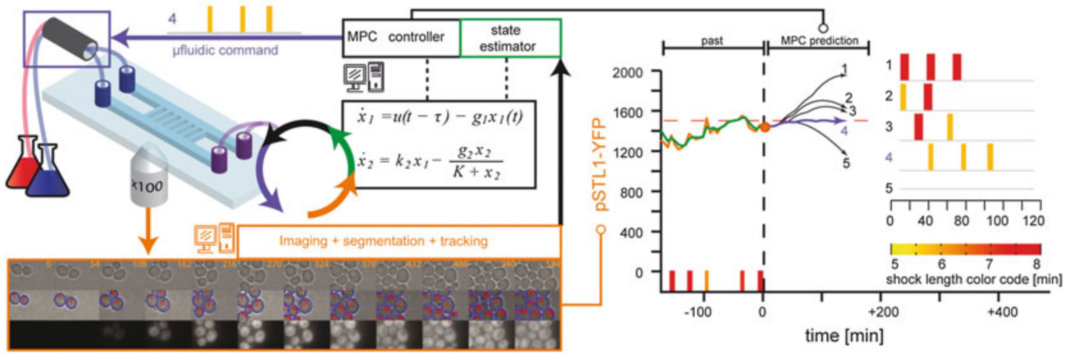

4
μfluidic command
MPC controller
state estimator
$\dot{x}_1 = u(t-\tau) - g_1 x_1(t)$
$\dot{x}_2 = k_2 x_1 - \frac{g_2 x_2}{K + x_2}$
x100
Imaging + segmentation + tracking
past
MPC prediction
pSTL1-YFP
2000
1600
1200
800
400
0
1
2
3
4
5
-100
0
+200
+400
time [min]
0
40
80
100
120
5
6
7
8
shock length color code [min]

Fig. 2 A platform for real-time control of gene expression in yeast [14]. (*Left*) Yeast cells grew as a monolayer in a microfluidic device which was used to rapidly change the cells' osmotic environment (valve, *blue frame*) and to image their response. Segmentation and cell tracking were done using a Hough transform (*orange frame*). The measured yECitrine fluorescence, either of a single cell or of the mean of all cells, was then sent to a state estimator connected to an MPC controller. A model (*center*, *black frame*) of pSTL1 induction was used to find the best possible series of osmotic pulses to apply in the future so that the predicted yECitrine level follows a target profile. (*Right*) At the present time point (*orange disk*), the system state is estimated (*green*) and the MPC searches for the best input (pulse duration, number of pulses) whose predicted effect (*blue* and *black curves*) minimizes its distance to the target profile (*red dashed line*) for the next 2 h. Here, the osmotic series of pulses that corresponds to the *blue curve* (#4) was selected and sent to the μfluidic command. This control loop is iterated every 6 min unless a stress is applied. *Solid lines* and their envelopes are the experimental means and standard deviations of the cells fluorescence

2.3 Model of pSTL1 Induction

To decide what osmotic stress to apply at a given time, we used an elementary model of pSTL1 induction. Many models have been proposed for the hyperosmotic stress response in yeast [10, 15–19]. We used a generic model of gene expression written as a two-variable delay differential equation system where the first variable denotes the recent osmotic stress felt by the cell and the second the protein fluorescence level (Fig. 2). Since our goal was to demonstrate robust control despite the presence of un-modeled feedback loops, the adaptation mechanisms described above were purposefully neglected. The choice of this model was also motivated by the trade-off between its ability to quantitatively predict the system's behavior (favors complexity) and the ease of solving state estimation problems (favors simplicity). Despite its simplicity, we found a fair agreement between model predictions and calibration data corresponding to fluorescence profiles obtained by applying either isolated or repeated osmotic shocks of various durations [14].

2.4 Closing the Loop

The fluorescence intensity either of a single cell, arbitrarily chosen at the start of the experiment, or averaged over the cell population, was sent to a state estimator (extended Kalman filter), connected to a model predictive controller (MPC) [14]. MPC is an efficient framework well adapted to constrained control problems. Schematically, given a model of the system and desired temporal profiles for system's outputs, MPC aims at finding inputs so as to minimize the deviation between the outputs of the model and the desired outputs. The control strategy is applied for a (short) period of time. Then the new state of the system is observed and this information is used to compute the control strategy to be applied during the next time interval. This receding horizon strategy yields an effective feedback control. In practice, every 6 min, given the current estimate of the system state, past osmotic shocks, and our model of gene expression, the controller searched for the optimal number of osmotic pulses to apply within the next 2 h and their optimal start times and durations (Fig. 2). If a shock had to be applied within the next 6 min, then it was applied. Otherwise, the same computation was reiterated 6 min later based on new observations. We dealt with short term cell adaptation by imposing a maximal stress duration of 8 min and a 20-min relaxation period between consecutive shocks. Under such conditions cells stay responsive to osmotic stress at all times.

2.5 Closed-Loop Population Control Experiments

First, we demonstrate that one can maintain the average fluorescence level of a cell population at a given constant value (set-point experiment) and force it to follow a time-varying profile (tracking experiment). Both types of experiments lasted at least 15 h, starting with a few cells and ending with 100–300 cells in the field of view. The control objective was to minimize the mean square deviations (MSD) between the mean fluorescence of the population of cells and the target profile. We succeeded in maintaining the average

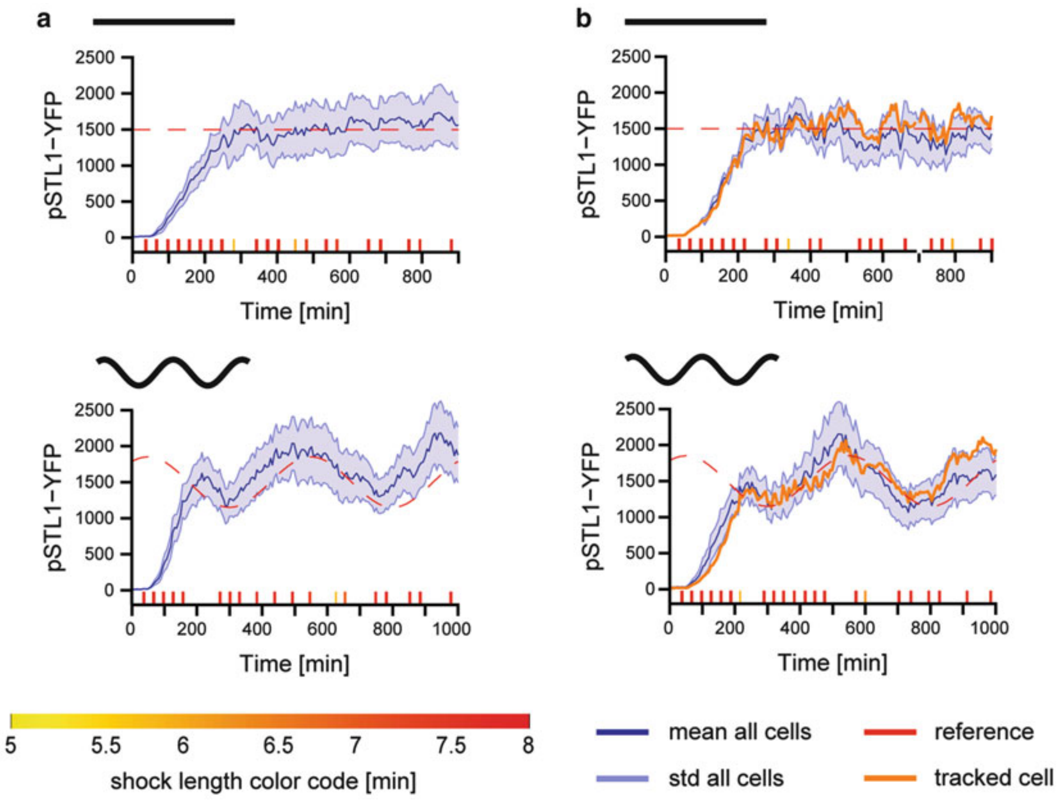


Fig. 3 Real-time control of gene expression [14]. (**a**) Control at the population level. Representative set-point control experiments and tracking control experiments are shown. Shock starting times and durations (see *color code*) were computed in real time. The measured mean cell fluorescence is shown as *solid blue* lines. The envelopes indicate standard deviation of the fluorescence distribution across the yeast population. (**b**) Control at the single cell level. The yECitrine fluorescence of the controlled cells are shown as *orange lines*. Note that the population follows the target profile but with less accuracy than the controlled single cell

fluorescence level at a given constant value, or in forcing it to follow time-varying profiles (Fig. 3) [14]. Admissible time-varying target profiles were obviously constrained by the intrinsic timescales of the system such as the maximal protein production and degradation rates. The effective control range spans an order of magnitude: set-point control can achieved between 200 and 2,000 fluorescence units [14]. Quantitative limitations of our experimental platform can originate from the model, the state estimator, the control algorithm and the intrinsic biological variability of gene expression. *In silico* analysis showed that applying the proposed control strategy to the (estimated state of the) system resulted in control performances that were significantly better than those obtained experimentally [14]. Therefore the control algorithm performed well, and future improvements should focus on system modeling and state estimation to better represent the experimental state of the system.

2.6 Closed-Loop Single-Cell Control Experiments

In a second set of experiments, we focused on the real-time control of gene expression at the single-cell level. We tracked one single cell over at least 15 h and used its fluorescence to feed the MPC controller. As shown in Fig. 3, we obtained results whose quality is out of reach of any conventional gene induction system, both for constant and for time-varying target profiles. Because of intrinsic noise in gene expression, single cell control was a priori more challenging than population control. And indeed, when compared with the mean fluorescence levels in population control experiments, the fluorescence levels of controlled cells in single-cell control experiments showed larger fluctuations around the target values. However, at the cell level, the mean square deviations of controlled cells obtained in single-cell control experiments were significantly smaller than that of a cell in population control experiments [14]. This shows that real-time control effectively improves control quality and counteracts the effects of noise in gene expression when performed at the single-cell level.

3 Discussion

3.1 Summary

We demonstrated that gene expression can be controlled in real-time with quantitative accuracy at both the population level and the single-cell level by interconnecting conventional microscopy, microfluidics, and computational tools. Importantly, we provided evidence that real-time control can dynamically limit the effects of biological noise when applied at the single-cell level. The fact that good control results can be obtained in a closed-loop setting with a relatively coarse model of an endogenous promoter suggests that extensive modeling will not be required to transpose our approach to other endo- and exogenous induction.

3.2 Related Works

The actual use of *in silico* feedback loops to control intracellular processes has been proposed only recently. In 2011, we showed that the signaling activity in live yeast cells can be controlled by an *in silico* feedback loop [20]. Using a proportional-integral (PI) controller we controlled the output of a signal transduction pathway by modulating the osmotic environment of cells in real time. A similar framework has been proposed by Menolascina et al. to control a large synthetic gene network [21]. More recently, Toettcher et al. used elaborate microscopy techniques and optogenetics to control in real time and at the single-cell level the localization and activity of a signal transduction protein (PI3K) in eukaryotic cells [22]. Interestingly, they were able to buffer external stimuli by clamping PIP_3 for short time scales. With this approach, the authors were able to reduce cell-to-cell variability of the cells output by applying different inputs to each cell (Fig. 4a). The most closely-related work is that of Milias-Argeitis et al. [23].

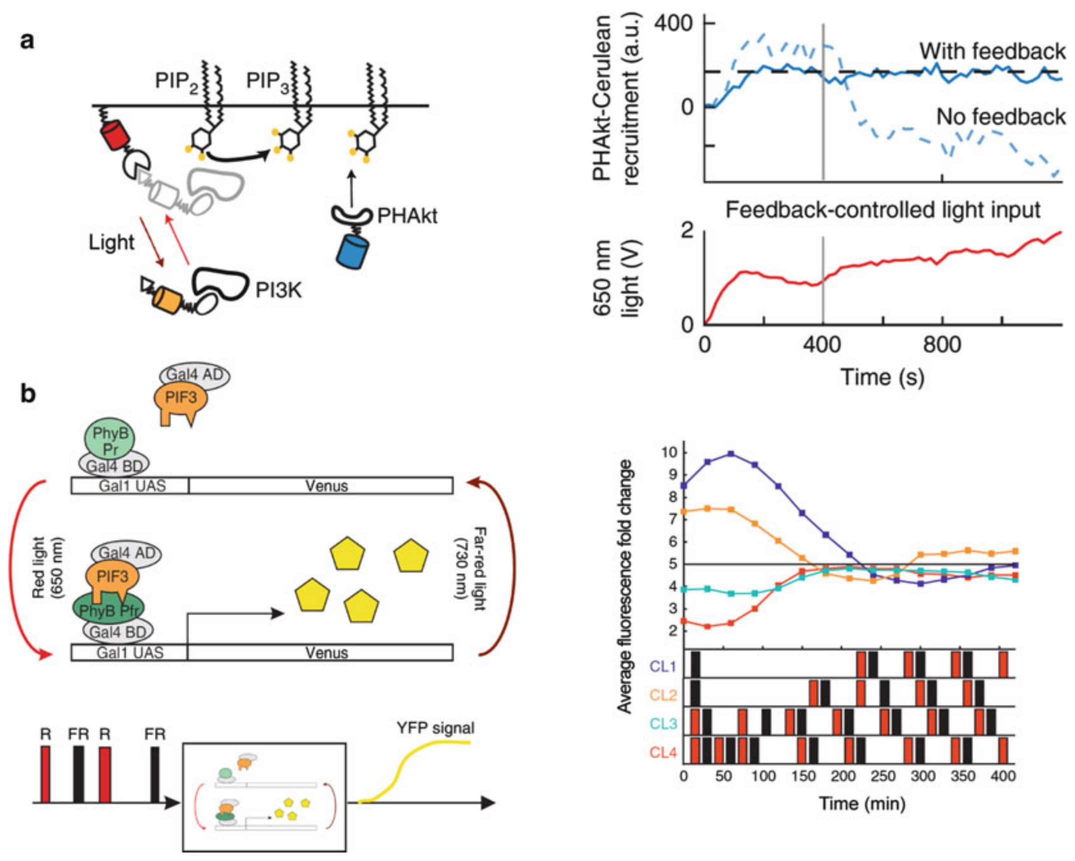


Fig. 4 Other real-time control platforms (**a**) Optogenetics control of localization and activity of PI3K in mammalian cells [22]. The amount of PI3K products, PIP3, was assayed by measuring PHAkt-cerulean recruitment to the plasma membrane. *Gray line* indicates the addition of a PI3K inhibitor at 400 s. (**b**) Optogenetics control of gene expression in chemostat using Venus as reporter protein in yeast [23]. Set-point control was achieved irrespectively of the initial state of the cell population

Using optogenetic techniques, they managed to control the expression of a yeast gene to a constant target value over several hours (Fig. 4b). In particular they are able to control the system to a fixed set point after they have sent a random series of pulses. Their approach is based on a chemostat culture and is therefore promising for many biotechnological applications such as the production of biofuels or small-molecule drugs, even if scaling up laboratory experiments to industry scale has proven difficult. However, because it does not allow for single-cell tracking and single-cell control, it is less adapted to probe biological processes in single-cell quantitative biology applications. These works have been reviewed in more depth by Chen et al. [24].

3.3 Perspectives

Connecting living cells to computers is a promising field of research both for applied and fundamental research. By maintaining a system around specific operating points or by driving it out of its standard operating regions, real-time control approaches offer unprecedented opportunities to investigate how gene networks process dynamical information at the cell level. We also anticipate that such platforms will be used to complement and help the development of synthetic biology via the creation of hybrid systems resulting from the interconnection of in vivo and *in silico* computing devices.

Acknowledgments

We acknowledge the support of the Agence Nationale de la Recherche (under the references DiSiP-ANR-07-JCJC-0001 and ICEBERG-ANR-10-BINF-06-01), of the Région Ile de France (C'Nano-ModEnv), of the Action d'Envergure ColAge from INRIA/INSERM (Institut Nationale de la Santé et de la Recherche Médicale), of the MechanoBiology Institute, and of the Laboratoire International Associé CAFS (Cell Adhesion France-Singapour).

References

1. Bhalla US, Ram PT, Iyengar R (2002) MAP kinase phosphatase as a locus of flexibility in a mitogen-activated protein kinase signaling network. Science 297:1018–23
2. Hooshangi S, Thiberge S, Weiss R (2005) Ultrasensitivity and noise propagation in a synthetic transcriptional cascade. Proc Natl Acad Sci U S A 102:3581–6
3. Cai L, Dalal CK, Elowitz MB (2008) Frequency-modulated nuclear localization bursts coordinate gene regulation. Nature 455:485–90
4. Celani A, Vergassola M (2010) Bacterial strategies for chemotaxis response. Proc Natl Acad Sci U S A 107:1391–6
5. Baumgartner BL, Bennett MR, Ferry M et al (2011) Antagonistic gene transcripts regulate adaptation to new growth environments. Proc Natl Acad Sci U S A 108:21087–92
6. O'Shaughnessy EC, Palani S, Collins JJ et al (2011) Tunable signal processing in synthetic MAP kinase cascades. Cell 144:119–31
7. de Nadal E, Alepuz PM, Posas F (2002) Dealing with osmostress through MAP kinase activation. EMBO Rep 3:735–40
8. Hohmann S (2002) Osmotic stress signaling and osmoadaptation in yeasts. Microbiol Mol Biol Rev 66:300–372
9. Miermont A, Uhlendorf J, McClean M et al (2011) The dynamical systems properties of the HOG signaling cascade. J Signal Transduct 2011:930940
10. Muzzey D, Gómez-Uribe C, Mettetal JT et al (2009) A systems-level analysis of perfect adaptation in yeast osmoregulation. Cell 138: 160–71
11. Yi TM, Huang Y, Simon MI et al (2000) Robust perfect adaptation in bacterial chemotaxis through integral feedback control. Proc Natl Acad Sci U S A 97:4649–53
12. Van Voorst F, Neves L, Oliveira R et al (2005) A member of the sugar transporter family, Stl1p is the glycerol/H + symporter in *Saccharomyces cerevisiae*. Mol Biol Cell 16: 2068–2076
13. O'Rourke SM, Herskowitz I (2004) Unique and redundant roles for HOG MAPK pathway components as revealed by whole-genome expression analysis. Mol Biol Cell 15:532–542
14. Uhlendorf J, Miermont A, Delaveau T et al (2012) Long-term model predictive control of gene expression at the population and single-cell levels. Proc Natl Acad Sci U S A 35:14271–14276
15. Klipp E, Nordlander B, Krüger R et al (2005) Integrative model of the response of yeast to osmotic shock. Nat Biotechnol 23: 975–82
16. Hao N, Behar M, Parnell SC et al (2007) A systems-biology analysis of feedback inhibition in the Sho1 osmotic-stress-response pathway. Curr Biol 17:659–67
17. Mettetal JT, Muzzey D, Gómez-Uribe C et al (2008) The frequency dependence of osmo-adaptation in *Saccharomyces cerevisiae*. Science 319:482–4
18. Zi Z, Liebermeister W, Klipp E (2010) A quantitative study of the Hog1 MAPK response to

fluctuating osmotic stress in *Saccharomyces cerevisiae*. PLoS One 5:e9522

19. Zechner C, Ruess J, Krenn P et al (2012) Moment-based inference predicts bimodality in transient gene expression. Proc Natl Acad Sci U S A 109:8340–8345
20. Uhlendorf J, Bottani S, Fages F, et al (2011) Towards real-time control of gene expression: controlling the hog signaling cascade. Pac Symp Biocomput 338–349
21. Menolascina F, di Bernardo M, di Bernardo D (2011) Analysis, design and implementation of a novel scheme for in-vivo control of synthetic gene regulatory networks. Automatica 47: 1265–1270
22. Toettcher JE, Gong D, Lim WA et al (2011) Light-based feedback for controlling intracellular signaling dynamics. Nat Methods 8:837–839
23. Milias-Argeitis A, Summers S, Stewart-Ornstein J et al (2011) In silico feedback for in vivo regulation of a gene expression circuit. Nat Biotechnol 29:1114–1116
24. Chen S, Harrigan P, Heineike B et al (2013) Building robust functionality in synthetic circuits using engineered feedback regulation. Curr Opin Biotechnol 24:790–6

Chapter 14

Stochastic Modular Analysis for Gene Circuits: Interplay Among Retroactivity, Nonlinearity, and Stochasticity

Kyung Hyuk Kim and Herbert M. Sauro

Abstract

This chapter introduces a computational analysis method for analyzing gene circuit dynamics in terms of modules while taking into account stochasticity, system nonlinearity, and retroactivity.

(1) *Analog electrical circuit representation for gene circuits*: A connection between two gene circuit components is often mediated by a transcription factor (TF) and the connection signal is described by the TF concentration. The TF is sequestered to its specific binding site (promoter region) and regulates downstream transcription. This sequestration has been known to affect the dynamics of the TF by increasing its response time. The downstream effect—retroactivity—has been shown to be explicitly described in an electrical circuit representation, as an input capacitance increase. We provide a brief review on this topic.

(2) *Modular description of noise propagation*: Gene circuit signals are noisy due to the random nature of biological reactions. The noisy fluctuations in TF concentrations affect downstream regulation. Thus, noise can propagate throughout the connected system components. This can cause different circuit components to behave in a statistically dependent manner, hampering a modular analysis. Here, we show that the modular analysis is still possible at the linear noise approximation level.

(3) *Noise effect on module input–output response*: We investigate how to deal with a module input–output response and its noise dependency. Noise-induced phenotypes are described as an interplay between system nonlinearity and signal noise.

Lastly, we provide the comprehensive approach incorporating the above three analysis methods, which we call *"stochastic modular analysis."* This method can provide an analysis framework for gene circuit dynamics when the nontrivial effects of retroactivity, stochasticity, and nonlinearity need to be taken into account.

Key words Stochastic process, Modularity, Synthetic biology, Noise propagation, Fan-out, Retroactivity

1 Introduction

Modularity is an important concept for engineering a composite system from a priori characterized components [1]. This concept enables efficient composition with predictability and module-interchangeability. In a real engineered system a connection between two components can cause interference, resulting in the loss of modularity. To minimize such interference, interfaces are designed to isolate any interfering effects. Synthetic biology is no exception. Connections between two biological components are mediated by transcription factors (TFs), which "bind" and "unbind" from their DNA specific binding sites. Thus, a circuit signal is "used" to establish a connection. In addition, gene circuits are significantly noisy due to the fact that TFs are often found in low copy numbers within single cells. Thus circuit signals—intracellular TF concentrations—can fluctuate in time significantly. The signal fluctuations—here we call "noise"—can propagate to the downstream gene-circuit components, making the signals in upstream and downstream statistically dependent, i.e., correlated.

In this chapter, we will provide a "modular" analysis method for gene circuits. First, we provide a brief review on our previous work on a gene circuit representation in terms of electrical circuit components, where component interference is represented explicitly by an increase in an input capacitance. Second, we describe noise propagation in a modular fashion. Third, the effect of noise on module properties is explained as an interplay between nonlinearity and stochasticity in the module. Lastly, these three different topics will be considered as a whole, to provide a modular framework for gene circuit dynamics that incorporate the effect of retroactivity, stochasticity, and system nonlinearity. This theoretical framework will provide a step toward understanding noise-induced phenotypes in terms of modular responses and will help exploit noise to enhance circuit performance.

2 Alternative Gene Circuit Representation in Terms of Electrical Circuit Components

This section represents gene circuits in terms of analog electrical circuit components: resistors and capacitors. To start with, we describe a circuit interface as a set of biological reactions that involve TFs that bind and unbind from their specific promoter sites to regulate downstream transcription processes. The TFs can be multimers and can self-regulate. Thus, the circuit interface can include a set of reactions of multimerization, TF binding and unbinding, transcription, and translation, as well as feedback reactions. For all these various types of interfaces, it was shown that the gene circuit reaction models can be mathematically mapped to analog electrical circuits [2] under the assumption that the binding and unbinding reactions are fast enough to be in equilibrium and bound TFs degrade much more slowly than the free TFs.

The effect of the downstream regulation by TFs has been experimentally shown to exist in protein signaling networks [3, 4] and gene regulatory networks [5–7]. Regulation can change the response speed of the regulating proteins as well as their concentration levels. These effects can be understood in terms of biological reactions. With respect to the speed of response, experimental verification in gene regulatory networks is still underway. The effect can be intuitively understood as follows: When bound TFs degrade more slowly than free TFs, perhaps because of the limited access of proteases, the degradation of the "total" TFs becomes slower. However, we have to note that if the bound TFs degrade with the same rate as the free form, the response speed will not change but the levels of the free TFs are just scaled down by the same factor for all transient time. This effect on the response speed was originally termed "retroactivity" [8] and the degree of tolerance "fan-out" [2].

The electrical circuit representation presented in [2] is more informative than the reaction model since the retroactive effect is explicitly shown in the former representation while the latter does not. Let us briefly review the work in [2]. We consider a protein synthesis and degradation process. Synthesis and degradation can be represented as a resistor–capacitor (RC) circuit and when the protein is allowed to regulate the downstream promoter sites, the RC circuit can be modified by adding an extra capacitor to the existing one in parallel as shown in Fig. 1. This parallel connection of the extra capacitor increases the total capacitance (that is the sum of the two capacitances), leading to a longer time to charge both the capacitors. Therefore, the response time increases. For a complex gene regulatory network the same mapping to the electrical circuit representation can be performed by adding an extra capacitor for each TF regulation. Thus, the retroactive effect in gene circuits can be systematically taken into account.

As an example, we can consider the repressilator, the first synthetic genetic oscillator. It is composed of three genes, *tetR*, *lacI*, and *cI*, which sequentially inhibit one another, making a negative feedback loop. This delayed negative feedback leads to oscillation in gene expression. Since there are three regulation reactions, three interfaces exist. For each interface, we need to add a capacitor in parallel with the existing one. Here we note that the additional capacitance is dependent on the voltage—TF concentration—applied and thus that the dependency of retroactivity on the TF concentration is taken care of. Thus, the electrical circuit representation of gene circuits can be illustrated as in Fig. 1.

In summary, when a gene circuit module makes a connection to another, the input capacitance of the downstream module will be increased depending on the concentration of TFs and the number of TF binding sites in the downstream module as shown

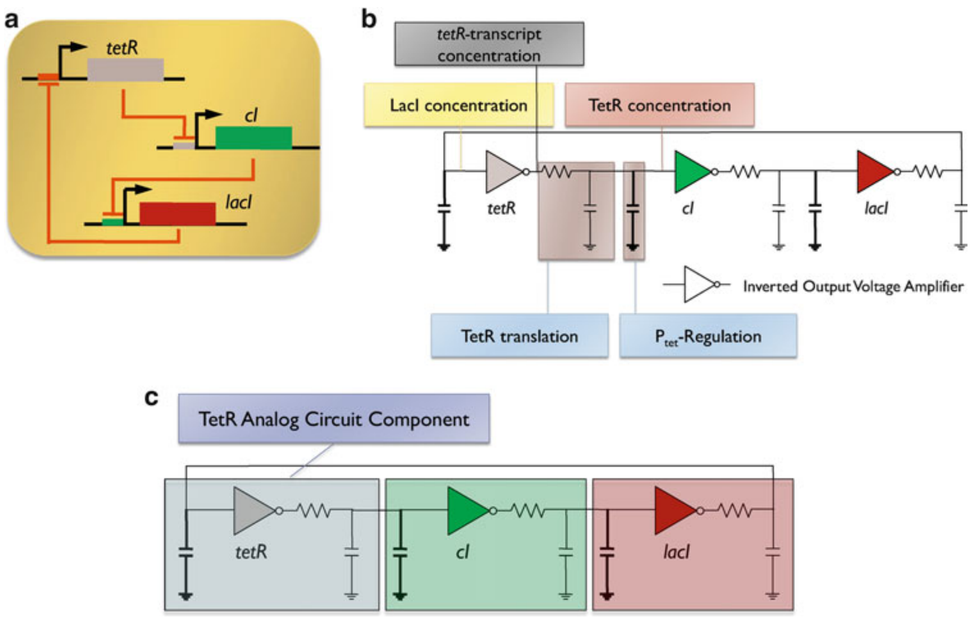


Fig. 1 Analog electrical circuit representation: (**a**) The repressilator is the first synthetic genetic oscillator. (**b**) It can be represented in terms of analog electrical circuit components such as resistors, capacitors, and inverted output voltage amplifiers. Here the inverted amplifiers represent the inhibitory genetic regulation of TetR, cI, and LacI, by inverting, scaling, and shifting the input voltages. Retroactivity can be explicitly described as an additional capacitor that depends on voltage applied. (**c**) Each gene expression and its regulation can be represented as an electrical circuit block or module, and retroactivity as an input capacitance increase of the module

in Figs. 1 and 2. Further analysis shows that the dependence on the number of TF binding sites (P_T in Fig. 2) turns out to be linear, which will be discussed in the next section.

3 Gene Circuit Fan-Out

For the case when retroactivity is significant, how can we quantify the degree of the retroactivity and furthermore provide certain limits in the downstream load to cap the retroactive effect? Electrical circuit components cannot tolerate unlimited connections. This is because with a finite amount of output current it is impossible to drive all the connected components. A similar phenomenon can occur in gene circuits. Thus, it is important to know how many downstream components can be connected, a number we call *fan-out* [2].

For example, consider an upstream module that is an oscillator, which needs to produce oscillatory signals faster than a certain frequency due to a biological limitation or desired design requirement. This maximum operating frequency will limit the number of the connections from the oscillator to downstream circuits because

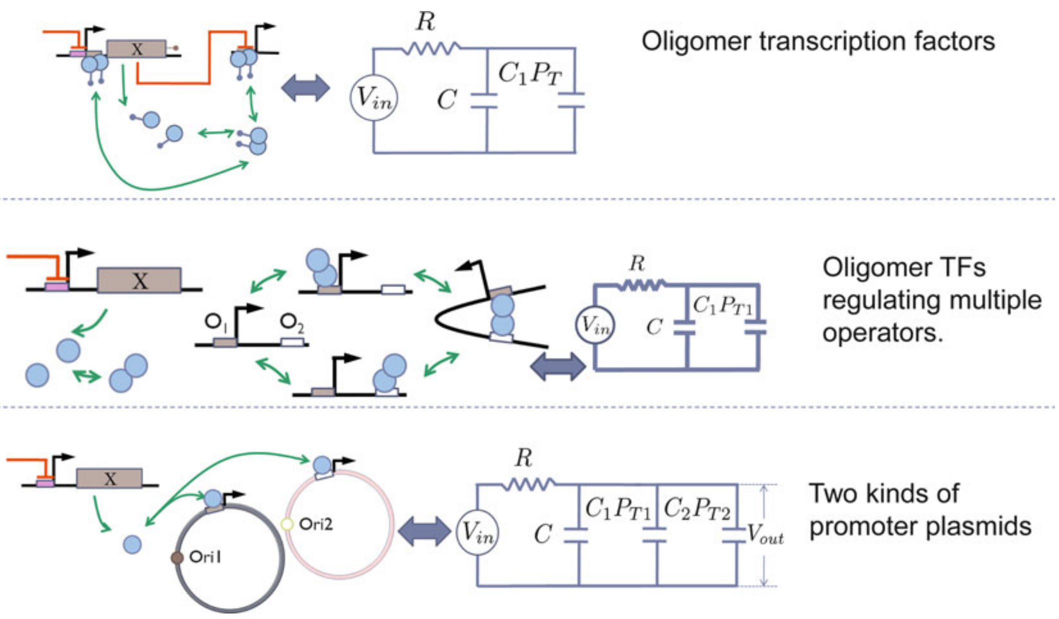

Oligomer transcription factors
X
R
V_{in}
C
C_1P_T
Oligomer TFs regulating multiple operators.
O_1
O_2
C_1P_{T1}
Two kinds of promoter plasmids
Ori1
Ori2
C_2P_{T2}
V_{out}

Fig. 2 Gene circuit representation in a wide range of circuit interfaces. *Top* Dimer transcription factors self-regulate their transcription processes, while regulating the downstream promoters as well. This interface can be transformed to an RC-circuit. Here the additional capacitance C_1P_T represents the input capacitance of the downstream circuit. Note that this capacitance is proportional to the number of the downstream promoters, P_T. *Middle* Dimer transcription factors regulate multiple operators (O_1 and O_2). *Bottom* Transcription factors regulate two types of plasmids having different copy numbers, P_{T1} and P_{T2}. In all the above cases and even the combinations of these, the additional capacitors are proportional to the number of the downstream promoters (P_T, P_{T1}, P_{T2}). This linearity can provide efficient characterization of gene circuit fan-out

as the number of the downstream interactions increase, the response time of the regulating protein increases due to retroactivity.

How can we measure the fan-out? Obviously the measurement requires the trend information on the response time vs. number of connections. If we have to repeat experiments for many different numbers of downstream connections, the characterization would be very tedious and perhaps even impractical. However, it was shown that the response time of the TFs show a certain linear relationship to the number of the downstream TF connections. This linearity was shown to be universal if each individual downstream promoter acts as an independent sequestrator of the TFs. Such linearity can be established for a wide range of circuit interface such as multimer TFs, feedback, multiple kinds of promoters, and multiple operator sites as shown in Fig. 2. Thus, fan-out can be characterized very efficiently due to the linearity.

What kinds of experiments need to be performed to measure fan-out? Two kinds, one at the population level using micro-plate readers and the other at the single cell level using fluorescence microscopy. The population level experiments can be performed, for example, by varying the strength of the downstream connections of the TFs either by adding small molecules that interact with the TFs or by changing the number of promoter sites that are specific to the TFs [5–7]. Single-cell level experiments can be performed by tagging TFs with fluorescent markers to directly monitor the fluctuations of the TF concentrations and by measuring the fluctuation speed (more precisely correlation time). For discussion on experimental characterization of response speed by using gene expression noise, refer to [9], and for more theoretical discussion, refer to [2, 10, 11].

4 Modular Description of Signal Noise

As briefly discussed in the previous section, gene circuits are noisy. Even within an isogenic cell population, each individual cell can show significantly different protein concentration levels. This is because transcription and translation events occur as a series of random bursts and the cells undergo division, differentiation, and apoptosis at random and even extra-cellular environments can show significant fluctuations. Due to the random nature of cellular dynamics and environmental effects, gene expression systems are often described as "stochastic" processes.

One important property of noisy systems is that noise passing through one system component can propagate to another and can lead to nontrivial "interference" between the components. Sometimes, this interference can be significant enough that the system properties are drastically changed, resulting in "noise-induced phenotypes."

As a first step to understand noise-induced phenotypes, we decompose the noise propagation in a modular way so that the phenotypes can be understood in terms of each module component effect. For this task, we propose the following modular analysis (*see* Subheading 5). This approach is based on a very simple fact that when noise is sufficiently small, the noise dominantly sees the linear components of nonlinear systems. In this case, a sinusoidal wave input (with a certain frequency ω) will propagate to an output with an identical-frequency sinusoidal wave without exciting other-frequency signals (because noise components can see only the "linear" system). This fact implies that in the frequency domain the output signal noise can be fully described by the input signal noise. Therefore, a modular description of noise propagation can be performed at the "linear noise approximation."

We will take the Langevin approach with the linear noise approximation [12, 13]. We will formulate the input–output noise response of a module by investigating self-correlation of each individual input and output noise signal [14, 15]. More precisely, the self-correlation can be mathematically described by an autocorrelation function that quantifies similarity between the values of signal pairs with a certain time lag. It turns out that the input autocorrelation function is sufficient to describe the output autocorrelation function if the module itself is fully characterized a priori.

5 Modular Description of Noise Propagation

To describe how internal and external (input) noise propagate through networks, we will investigate the properties of the autocorrelation functions of the noise. In particular, it can be shown that if the input noise autocorrelation function is known, the output noise autocorrelation function can be fully described by the input noise autocorrelation function for the stationary state of the internal dynamics, which are described by linear Langevin equations [16]. Furthermore, by taking the Fourier transforms of the autocorrelation functions, it can be shown the input–output noise response can be described by a transfer-like matrix equation. This analysis leads to modular analysis of complex stochastic networks.

The input signals of a module are denoted as $\boldsymbol{X}$, its output as $\boldsymbol{\Upsilon}$, and internal variables of the module as $\boldsymbol{Z}$. Internal dynamics is presumed to be described by

$$\frac{d\boldsymbol{Z}}{dt} = \boldsymbol{J} \cdot \boldsymbol{Z} + \xi + \boldsymbol{X}, \tag{1}$$

where ξ is a Gaussian white noise vector satisfying $\langle \xi_t \xi_{t'}^T \rangle = \boldsymbol{D}\delta(t - t')$ with $\boldsymbol{D}$ diffusion coefficient matrix and $\boldsymbol{J}$ is the Jacobian matrix. We assume that the correlation between the input noise and the internal noise (ξ) is negligible. We introduce an autocorrelation function to quantify the similarity between signal pairs separated by a time lag τ:

$$\boldsymbol{G}_Z(\tau) = \lim_{t_0 \to \infty} \langle \boldsymbol{Z}_{t_0} \boldsymbol{Z}_{t_0+\tau}^T \rangle. \tag{2}$$

Previous research [14, 15, 17, 18] has been focused on the properties of the diagonal terms of the above matrix. This is why the general structure of input–output noise relationship has been absent. The solution of Eq. 1 is obtained as

$$\boldsymbol{Z}_t = \exp[\boldsymbol{J}t] \int_0^t ds \ \exp[-\boldsymbol{J}s] \cdot (\xi_s + \boldsymbol{X}_s). \tag{3}$$

The autocorrelation matrix, expressed in terms of the correlation matrix of ξ and $\boldsymbol{X}$ by substituting Eq. 3 to Eq. 2,

can be simplified after Fourier transformations as:

$$\tilde{\boldsymbol{G}}_Z(\omega) = \left(\boldsymbol{J} + i\omega \boldsymbol{I}\right)^{-1} \cdot \left(\boldsymbol{D}/(2\pi) + \tilde{\boldsymbol{G}}_X(\omega)\right) \cdot \left(\boldsymbol{J}^T - i\omega \boldsymbol{I}\right)^{-1},$$

using $\langle (\xi_s + \boldsymbol{X}_s) \cdot (\xi_{s'} + \boldsymbol{X}_{s'})^T \rangle = \boldsymbol{D}\delta(s - s') + \boldsymbol{G}_X(s - s')$. $\tilde{\boldsymbol{G}}(\omega)$ is the Fourier transform of $\boldsymbol{G}(t)$. Finally, the output noise autocorrelation function can be expressed, by using a projection matrix $\boldsymbol{P}$ mapping the state space $\{\boldsymbol{Z}\}$ onto the output subspace $\{\boldsymbol{\Upsilon}\}$, as

$$\tilde{\boldsymbol{G}}_{\Upsilon}(\omega) = \boldsymbol{P} \cdot \tilde{\boldsymbol{G}}_Z(\omega) = \boldsymbol{P} \cdot \left(\boldsymbol{J} + i\omega \boldsymbol{I}\right)^{-1} \cdot \left(\frac{\boldsymbol{D}}{2\pi} + \tilde{\boldsymbol{G}}_X(\omega)\right) \cdot \left(\boldsymbol{J}^T - i\omega \boldsymbol{I}\right)^{-1}. \tag{4}$$

Equation 4 is important because it shows that the output autocorrelation function can be fully described by the input autocorrelation function and that there is a simple matrix relationship through the transfer-like function. Equation 4 also succinctly shows how output noise is affected by internal noise ($\boldsymbol{D}$) and input noise ($\tilde{\boldsymbol{G}}_X(\omega)$). This allows a systematic approach for noise propagation in complex gene and signaling networks using the *modularity* concept.

To take into account the effect of retroactivity in this modular analysis, we can consider an additional capacitance that corresponds to the effect of downstream loads and analyze the noise propagation with the input capacitance taken into account as illustrated in Fig. 1.

In this analysis method, we can understand the input–output noise response of a module for different cases such as (a) non-stochastic input and stochastic internal noise ($\tilde{\boldsymbol{G}}_X(\omega) = 0$); (b) stochastic input and non-stochastic internal noise ($\boldsymbol{D} = 0$ and $\tilde{\boldsymbol{G}}_X(\omega) \neq 0$); (c) both internal and external noise ($\boldsymbol{D} \neq 0$ and $\tilde{\boldsymbol{G}}_X(\omega) \neq 0$).

This approach can also be used to devise noise (de-)amplifiers and filters by taking into account different network topology and kinetics, for which the information can be found in the matrices $\boldsymbol{D}$ and $\boldsymbol{J}$. If the matrices are suitably chosen, noise (de-)amplifiers and filters can be designed.

6 Noise Effect on Nonlinear Responses

In the previous section, noise propagation was described in a modular fashion with the approximation that the first and second moments of a signal noise are sufficient to describe the system behaviors. Under the same level of approximation, we will describe the effect of noise on a module input–output transfer curve. First, consider a linear system with an input (i) and an output (o), satisfying the following relationship:

$$o = c\ i,$$

with c a proportionality constant. The mean values of i and o satisfy the same relationship:

$$\langle o \rangle = c\langle i \rangle.$$

However, when a system is nonlinear, the same relationship does not hold typically, simply because of Jensen's inequality: In the convex function $f(x)$, the function of the mean of the two x values, $f([x_1 + x_2]/2)$, is always smaller than the mean value of the two functional values of x_1 and x_2, $[f(x_1) + f(x_2)]/2$, because this is located on the secant line (*see* Fig. 3). In more general term, the expectation of a convex function of a random variable is always greater than the function of the expectation. Jensen's inequality explains that a sigmoidal response curve can be less sigmoidal (as shown in Fig. 3b) and in certain cases, can be more sigmoidal, resulting in ultrasensitivity [19, 20]. This has been shown that the interplay between system nonlinearity and stochasticity can cause nontrivial noise-induced phenotypes such as noise-induced bistability and noise-induced linearized response [20].

More mathematically, the expectation value of $f(x)$ can be approximated as

$$\langle f(x) \rangle \simeq f(\langle x \rangle) + \frac{f''(x)}{2} Variance(x). \tag{5}$$

This gives a simple explanation of Jensen's inequality by showing the sign dependence of $f''(x)$. The second term in Eq. 5 describes the interplay between nonlinearity and stochasticity: The double derivative $f''(x)$ corresponds to the former and the variance to the latter. The variance and mean values of x can be estimated with a moment closure approximation by neglecting the third and higher moments of x (refer to Appendix in [20]). This approximation is called the mass fluctuation kinetics in mass reaction systems [21].

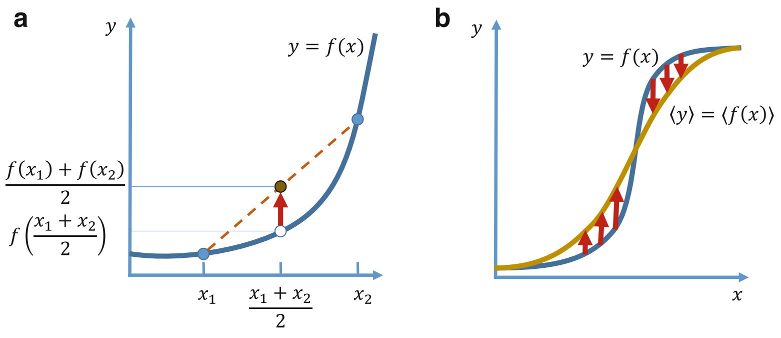

a
y
$y = f(x)$
$\frac{f(x_1)+f(x_2)}{2}$
$f\left(\frac{x_1+x_2}{2}\right)$
x_1
$\frac{x_1+x_2}{2}$
x_2
b
y
$y = f(x)$
$\langle y \rangle = \langle f(x) \rangle$
x

Fig. 3 Interplay between system nonlinearity and stochasticity

7 Stochastic Modular Analysis

We combine the modular noise propagation analysis provided in Subheading 5 to the mass fluctuation kinetics to consider the cases in the stronger noise regime where noise-induced phenotypes can appear. Mass fluctuation kinetics can describe the system behavior with more accuracy than linear noise approximation approaches by taking into account an interplay between system nonlinearity and stochasticity exactly as in Eq. 5. The variance (more in general, covariance) term in Eq. 5 can be described in terms of the input signal values in the modular noise propagation analysis, $\tilde{\boldsymbol{G}}_x(\omega)$, as follows:

$$Variance(x) = \int \tilde{\boldsymbol{G}}_x(\omega)\ d\omega\ .$$

This shows that it is possible to combine the modular noise propagation analysis with the mass fluctuation kinetics (*see* Fig. 4). Therefore, network-level dynamics can be described in terms of modules.

To take into account the effect of retroactivity, an additional input capacitance can be added per a module input that corresponds to the effect of the input to its connected output. Then, the noise propagation can be described in terms of modules as shown in Fig. 4.

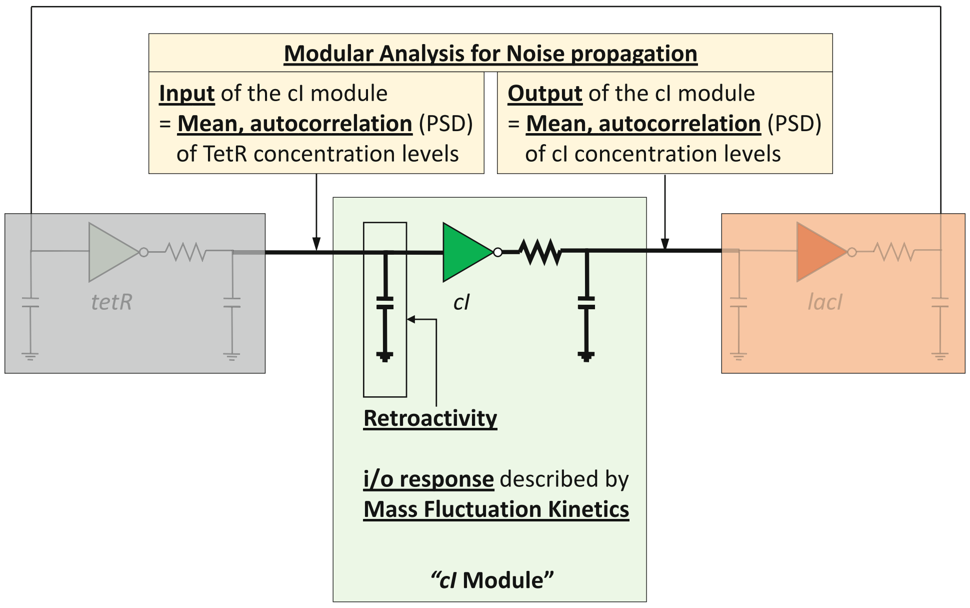

Modular Analysis for Noise propagation
Input of the cI module = Mean, autocorrelation (PSD) of TetR concentration levels
Output of the cI module = Mean, autocorrelation (PSD) of cI concentration levels
tetR
cI
lacI
Retroactivity
i/o response described by Mass Fluctuation Kinetics
"cI Module"

Fig. 4 Stochastic modular analysis: As a simple genetic circuit example, the repressilator is represented in terms of analog circuits. The noisy signals are analyzed in terms of modules by considering its first and second moments: mean and autocorrelation, or equivalently power spectral density (PSD). A module input–output (i/o) response is analyzed by using mass fluctuation kinetics by considering the system nonlinearity and stochasticity. The retroactivity is explicitly described by an additional parallel capacitance connection

Therefore, it is possible to combine the three different analyses that were discussed in the previous sections: (1) analog circuit representation for the explicit description of retroactivity; (2) modular description of noise propagation; (3) the effect of system nonlinearity and stochasticity. This combined global analysis, we name stochastic modular analysis, will provide an analysis framework that takes into account *retroactivity, stochasticity, system nonlinearity, and modularity*, all together in a system-wide level.

References

1. Baldwin CY, Clark KB (2000) Design rules: the power of modularity. MIT Press, Cambridge
2. Kim KH, Sauro HM (2010) Fan-out in gene regulatory networks. J Biol Eng 4:16
3. Jiang P, Ventura AC, Sontag ED, Merajver SD, Ninfa AJ, Del Vecchio D (2011) Load-induced modulation of signal transduction networks. Sci Signal 4(194):ra67
4. Ventura AC, Jiang P, Van Wassenhove L, Del Vecchio D, Merajver SD, Ninfa AJ (2010) Signaling properties of a covalent modification cycle are altered by a downstream target. Proc Natl Acad Sci USA 107:10032–10037
5. Buchler NE, Cross FR (2009) Protein sequestration generates a flexible ultrasensitive response in a genetic network. Mol Syst Biol 5:272
6. Jayanthi S, Nilgiriwala KS, Del Vecchio D (2013) Retroactivity controls the temporal dynamics of gene transcription. ACS Synth Biol
7. Daniel R, Rubens JR, Sarpeshkar R, Lu TK (2013) Synthetic analog computation in living cells. Nature 497(7451):619–623
8. Del Vecchio D, Ninfa AJ, Sontag ED (2008) Modular cell biology: retroactivity and insulation. Mol Syst Biol 4:161
9. Weinberger LS, Dar RD, Simpson ML (2008) Transient-mediated fate determination in a transcriptional circuit of HIV. Nat Genet 40 (4):466–470
10. Kim KH, Sauro HM (2011) Measuring retroactivity from noise in gene regulatory networks. Biophys J 100(5):1167–1177
11. Kim KH, Sauro HM (2012) Measuring the degree of modularity in gene regulatory networks from the relaxation of finite perturbations. In: 2012 I.E. 51st IEEE conference on decision and control (CDC), pp 5330–5335
12. Elf J, Ehrenberg M (2003) Fast evaluation of fluctuations in biochemical networks with the linear noise approximation. Genome Res 13 (11):2475–2484
13. Paulsson J (2004) Summing up the noise in gene networks. Nature 427(6973):415–418
14. Tănase-Nicola S, Warren PB, ten Wolde PR (2006) Signal detection, modularity, and the correlation between extrinsic and intrinsic noise in biochemical networks. Phys Rev Lett 97(6):68102
15. Warren PB, Tanase-Nicola S, ten Wolde PR (2006) Exact results for noise power spectra in linear biochemical reaction networks. J Chem Phys 125(14):144904
16. Kwakernaak H, Sivan R (1972) Linear optimal control systems. Wiley-Interscience, New York
17. Simpson ML, Cox CD, Sayler GS (2003) Frequency domain analysis of noise in autoregulated gene circuits. Proc Natl Acad Sci USA 100(8):4551
18. Austin DW, Allen MS, McCollum JM, Dar RD, Wilgus JR, Sayler GS, Samatova NF, Cox CD, Simpson ML (2006) Gene network shaping of inherent noise spectra. Nature 439:608–611
19. Paulsson J, Berg OG, Ehrenberg M (2000) Stochastic focusing: fluctuation-enhanced sensitivity of intracellular regulation. Proc Natl Acad Sci USA 97(13):7148–7153
20. Kim KH, Qian H, Sauro HM (2013) Nonlinear biochemical signal processing via noise propagation. arXiv:1309.2588 [q-bio.QM]
21. Gómez-Uribe CA, Verghese GC (2007) Mass fluctuation kinetics: capturing stochastic effects in systems of chemical reactions through coupled mean-variance computations. J Chem Phys 126(2):24109

Part IV

Distributed Systems and Automation

Chapter 15

Distributed Model Construction with Virtual Parts

Michael T. Cooling and Tommy Yu

Abstract

Here three models for a simple genetic device are constructed using modular mathematical models. The models are stored in an online system (the Physiome Model Repository) with distributed source and access control, demonstrating a collaborative method for creating mathematical models from reusable components.

Key words Mathematical modeling, Model, CellML, Distributed, Repository, Virtual parts, Collaborative

1 Introduction

An important goal in synthetic biology modeling is to develop a set of reusable, composable modular models that can be combined and re-combined to form models of genetic circuits. Considered individually, these models would quantify the behavior of biological components (as in ref. 1, for example), being abstractions of characterized biological parts suitable for use in computer-aided design of biology ("BioCAD"). While the highly successful Parts Registry [2] aims to describe and characterize the biological details of synthetic biology parts, we describe here the construction and use of modular mathematical models designed to augment the Parts Registry by providing *virtual* parts [3]. These Virtual Parts could be used to construct robust models of the genetic circuits and predict their behavior before construction of the circuits themselves, stimulating iterative design-and-test cycles reducing expensive or time-consuming laboratory work.

Virtual Parts are constructed in the model exchange protocol CellML [4], which is inherently modular [5] and has a strong track record in representing systems biology models (e.g., refs. [6, 7]), and multiscale models [8]. CellML is a declarative, XML-based modeling protocol that can be used to represent sets of ODEs, expressed as mathematical equations and accompanying variables. The equations are expressed in MathML (http://www.w3.org/Math/) expressions, which represent equations as a hierarchy of subexpressions in prefix form, encoded in XML. CellML also demands the explicit definition of units for all variables and constants, and thereby allows dimensional consistency checking of expressed equations. CellML allows the mathematics and associated variables to be partitioned into components, which can be considered as submodels of the original model. Variables in different components can be set as equivalent to one another, through a CellML connection. In this way, information can be shared between components. One might consider a component having inputs of some variables, containing an equation that calculates another variable from the "inputs", which is then made available to other components as the current component's output. The most recent version of CellML, version 1.1, includes the ability to spread a model across multiple CellML model documents, where an in-memory copy of a component (or unit definition) in one model can be made available in another model. This is known as "importing", and provides a way to share or replicate (the same component can be imported multiple times, each being considered a separate instance by the CellML solver) mathematical forms, units, and/or parameter values between models. A hierarchy of components can be defined, including those that are imported from other CellML models, and encapsulation of component variables can be set, as well as interfaces defined as to which variables in which components can have connections between them. In this way, submodel complexity can be hidden from higher levels of the larger model, which can aid model composition. While the CellML code will be described in sufficient detail in order to complete the processes described here, readers new to CellML and interested in learning to use the protocol more generally are invited to peruse the "Getting Started" material on the CellML web site at http://www.cellml.org.

CellML models using Virtual Parts are organized into a hierarchy of at least three levels. At the top of the hierarchy is the "Systems Model". This is the model of the biological system that one is interested in. It can be considered an aggregation (usually via CellML imports) of submodels that each describe one biological entity or process—the Virtual Parts or second level. Virtual Parts import their mathematical form from the third level, which are given the name "Templates" as they provide a mathematical Template for a set of Virtual Parts, each having the same mathematical form. A Template once imported is made more specific in the Virtual Part by ascribing particular parameter values to the mathematics. For example, a Template may contain the mathematical form for an activatable promoter, which includes several (unset) parameters. Virtual Parts that represent specific promoters import the aforementioned Template, but also contain parameter values specific to the particular promoter that they represent. The Systems Model may also directly contain other components that are used to perform helpful mathematical operations, such as aggregation of fluxes of species into net fluxes, or conversion between two variables that are of the same dimensions but different units.

The Systems Model may also import Templates directly, without an intervening Virtual Part. This happens where a mathematical form is reused, but does not have any accompanying parameterization. The most common example of this is in modeling biological molecular species, where a simple ODE for the species is computed on the basis of a single variable, such as net flux. Initial conditions for the variable are held in some other component (*see* **Note 1**). The other common use for the direct importing facility is in incorporating a variable to represent time, which in CellML is a special variable with no initial condition.

The Virtual Parts and Templates are stored in the Physiome Model Repository [9] (PMR) which allows multiple model authors from around the world to collaborate on their models securely in "Workspaces". The Workspaces are versioned, with changes by model developers tracked, through PMR's use of the Mercurial distributed version control system [10, 11]. Workspaces are only available to user logins to whom the owner of the Workspace gives access, but models can be exposed to the public through the forming of an "Exposure" which, as the name implies, exposes nominated contents (a particular changeset) of a Workspace to the online public, often with a presentation page describing the details of what is being exposed. The Repository can be found at http://models.cellml.org.

We will begin by describing how to again access to the PMR and how to use it to build a model of an example biological system, shown in Fig. 1.

Several Virtual Parts for this system already exist in PMR; the model can largely be constructed from Virtual Parts from the "Bugbuster" model held in PMR (see the latest Exposure of the Bugbuster workspace, which in turn can be found here http://models.cellml.org/workspace/bugbuster). The ribosome binding site (RBS) can be modeled using the same Template model as the Bugbuster RBS, but we shall assume that the RBS used here has different kinetics, so that we can create new Virtual Part using that Template. We will describe how the complete Systems Model is formed from the Virtual Parts (including that derived from the new promoter Template), building the model iteratively "backwards" from GFP—beginning with a simpler GFP-only model, then modifying this to include details of the genetic Device. You will then be shown how to substitute a new, more powerful RBS Virtual Part, which you will construct. We will also detail how to make the model

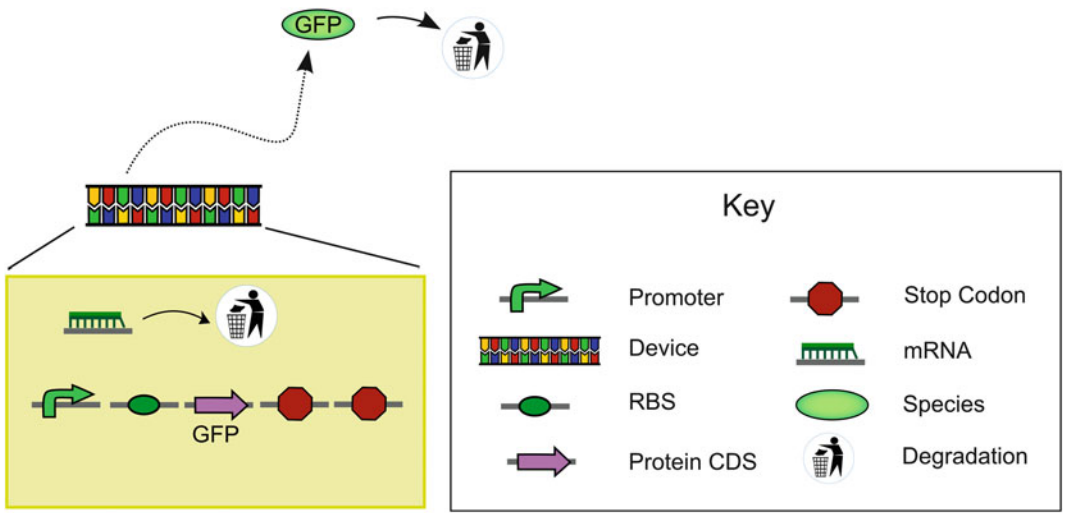


Fig. 1 A simple biological system is depicted on the *left*, consisting of a constitutively active device that produces GFP (green fluorescent protein). The device consists of a promoter, a ribosome binding site, and a coding sequence or gene for GFP, followed by two stop codons (stop codons are not modeled explicitly). RNA transcribed from the gene can be translated into the GFP species, or degraded by intracellular processes. GFP is assumed to have some known half-life before also being degraded

Workspace available to collaborators from around the world, and finally how to publically "expose" the completed model of the system to the general public, online.

2 Materials

It is assumed that you have a computer (a Windows 7 PC is assumed here, but other platforms such as Linux or MacOS are equally viable) with a functional internet connection. Most of the operations can be performed with an internet browser and your favorite text editor; however in order to upload code to PMR you will have to install a Mercurial client. You may also find it useful to try one of the CellML modeling environments available (*see* **Note 2**).

2.1 Install Mercurial

(if it is not already present on your system).

1. In your browser, navigate to the "downloads" part of the Mercurial web site. At the time of writing, this can be found at http://mercurial.selenic.com/downloads/.
2. Select the appropriate download for your computer system, and install it (*see* **Note 3**). A wide range of installers for different platforms, and source code distributions exist. If you are a Windows user, you may also like to download and install the Tortoise user interface (*see* **Note 4**), although in this chapter the existence of Tortoise will not be assumed.

3. In order to make commits to repositories, it is necessary to signal to Mercurial what your name and email address is, for tracking purposes. In Windows 7, this is done by saving the following as `Mercurial.ini` under %USERPROFILE% (usually "C:\Users\<username>"):

```
[ui]
username = Your Name<yourname@youremail.com>
```

3 Methods

3.1 Obtain Access to PMR

1. Navigate your internet browser to the CellML model repository (http://models.cellml.org).
2. Click on "Register" in the top right corner of the web page. You can also follow a link at the bottom of the page under "New user?"
3. Fill in the Full Name, User Name, and E-mail address and input the "Captcha" words, then click the Register button.
4. PMR will now send you a confirmation email to the email address that you specified. When you receive this email, click on the link provided.
5. You will be taken to a password setting web page. Please set your password for your PMR login on that page.
6. You can then Log in by clicking "Log in" (next to the "Register" link that you clicked on in **step 2** above), and log in with your username and password.
7. You can then follow the link to the main page of PMR, and you will now be logged in as your new user (*see* **Note 5**).

3.2 Create a New Workspace

1. From the button bar at the top left, select "My Workspaces".
2. A list of Workspaces that you can access would be shown here, if you had any. Follow the first link to the Workspace container.
3. On the top right, click the "add new..." option.
4. A drop-down should appear: click the "pmr2 workspace" option.
5. On the web page that appears, give your new Workspace an Id, Title, and Description. "Id" will determine the URI of your Workspace, and may become quite important as you will use this URI to share your Workspace with others, or perhaps to cite your models electronically. Thus it is recommended that you choose your Workspace's Id with care (*see* **Note 6**).
6. Click the "Add" button to create the Workspace. You will be taken to the page for the new Workspace once it is created (*see* **Note 7**).

3.3 Derive a Model of GFP Degradation

Here you will create a simple model of GFP degrading in the cytosol (the "right end" of the model depicted in Fig. 1). To do this, you are going to reuse model components from the Bugbuster model. The easiest way is to obtain those from the latest Exposure of that model (*see* **Note 8**).

1. Navigate in your browser to http://models.cellml.org/workspace/bugbuster and navigate down the page until you find an entry for

   ```
   Bugbuster_AllFiles.zip
   ```

2. Hit the [download] link on the right and download the .zip file to a convenient folder.
3. On your local file system, create a model-building folder where you will create the new model.
4. Create a new model file in the model-building folder. It should be a simple XML file with the basic CellML <model/> element, as shown here:

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <model name="GFPModel"
     xmlns="http://www.cellml.org/cellml/1.1#"
     xmlns="http://www.cellml.org/cellml/1.1#"
     xmlns:cellml="http://www.cellml.org/cellml/1.1#"
     xmlns:xlink="http://www.w3.org/1999/xlink">
   </model>
   ```

   and save it as "GFPModel.cellml".
5. The new model will need a concept of time. Copy the "Time.cellml" file from the unzipped folder to your model-building folder. Time can then be imported by adding the following code inside the <model/> tag of "GFPModel.cellml":

   ```
   <import xlink:href="Time.cellml">
     <component name="Time"
   component_ref="Time" />
   </import>
   ```

6. The model will need a concept of species to represent the GFP. Copy the "Template_Species.cellml" file to your model-building folder. Import an instance of the Species template that you will use to represent GFP with the following code placed inside the <model/> tag:

   ```
   <import xlink:href="Template_Species.cellml">
     <units name="nM_per_s" units_ref="nM_per_s" />
     <units name="nM" units_ref="nM" />
     <component name="GFP" component_ref="Template_
     Species" />
   </import>
   ```

Note that here you have imported units from the Template (*see* **Note 9**). You have also set the name of the local version of the Template_Species component to "GFP".

7. You will need to initialize your imported species component with an initial value. Create a separate component to hold initial concentrations of species, by placing the following code inside the <model/> tag:

```
<component name="SpeciesInitialConcentrations">
  <variable name="GFP" units="nM" initial_value
  ="1" public_interface="out" />
</component>
```

   In this simple model, there is only one initial condition; however in a more complex model there is more than one variable to set and the separation of initial conditions into a separate component can be very useful.

8. To introduce the concept of degradation, copy the "Template_RxR1P1.cellml" file to your model-building location. Copy also the "Bioenvironment_Degradation_GFP.cellml" file, which already imports the Template and parameterizes it for degradation of GFP. Degradation can now be added to the model by placing the following code inside the <model/> tag:

```
<import xlink:href="Bioenvironment_Degradation_
GFP.cellml">
  <component  name="Bioenvironment_Degradation_
  GFP" component_ref="Bioenvironment_Degradation_
  GFP" />
</import>
```

   That component itself imports the "Template_RxR1P1" component; hence you do not have to import that one explicitly.

9. You should define a component that aggregates fluxes of GFP—whether GFP is produced from some process (a positive flux) or removed/consumed (a negative flux). Thus it is useful to define an "interface" component for GFP as follows:

```
<component name="GFP_interface">
  <variable name="JGain" units="nM_per_s" public
  _interface="out"/>
  <variable name="JMinusDegradation" units= "nM_
  per_s" public_interface="in"/>
     <math  xmlns="http://www.w3.org/1998/Math/
     MathML">
       <apply>
         <eq/>
         <ci>JGain</ci>
         <apply>
           <minus/>
```

```
          <ci>JMinusDegradation</ci>
        </apply>
      </apply>
    </math>
</component>
```

The first part declares variables for the net gain, and for the degradation flux (*see* **Note 10**). The second part declares the MathML for the equation defining the flux aggregation equation—in this case the `JGain` variable—as the negative of the `JMinusDegradation` flux.

10. At present, while you have everything necessary to construct our model, none of the components are connected together. The imported GFP component, for example, needs to link to the model's concept of time. It also needs to have the initial concentration set. You achieve these goals by connecting them to your imported Time, and the species concentration you defined earlier, via the addition of the following code:

```
<connection>
  <map_components component_1="GFP" component_2
  ="Time"/>
  <map_variables variable_1="time" variable_2
  ="time"/>
</connection>

<connection>
  <map_components component_1="GFP" component_2
  ="SpeciesInitialConcentrations"/>
  <map_variables variable_1="concentration
  InitialValue" variable_2="GFP"/>
</connection>
```

You should link the output of the degradation reaction to the "GFP_interface" component that you defined, with the following code:

```
<connection>
  <map_components component_1="GFP" component_2
  ="GFP_interface"/>
  <map_variables variable_1="JGain" variable_2
  ="JGain"/>
</connection>
```

11. Finally, the degradation component needs to know which species it is degrading. You therefore add the following connection so that it acts on the GFP species that you defined earlier:

```
<connection>
  <map_components component_1="Bioenvironment_
  Degradation_GFP" component_2="GFP"/>
```

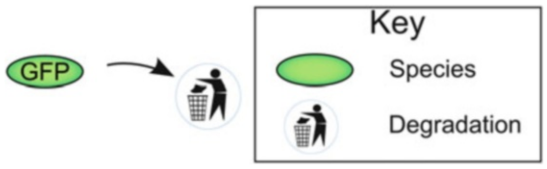


Fig. 2 A schematic of the GFP model. GFP is assumed to be present in the system, and is degraded according to mass-action kinetics

```
  <map_variables    variable_1="concentration"
  variable_2="concentration"/>
</connection>
```

Also, the output of the degradation component, or the calculated degradation flux, needs to be connected to the "JMinus" variable you defined in the "GFP_interface", thus:

```
<connection>
  <map_components  component_1="GFP_interface"
  component_2="Bioenvironment_Degradation_GFP"/>
  <map_variables variable_1="JMinusDegradation"
  variable_2="J"/>
</connection>
```

12. You now have a model of the system shown in Fig. 2. It may have seemed complicated to make such a small model; however you now have a set of reusable pieces that make it easier to add other reactions and species, of similar types, to your model. You may like to try simulating your new model by loading the "GFPmodel.cellml" file into a CellML simulation tool (*see* **Note 2**). If you run the model for 1,000 s and plot GFP. concentration over Time.time you should get a decreasing function as the concentration asymptotically approaches zero, over the course of approximately 5,000 s.

3.4 Commit the Model to a Local Repository, and Then to PMR

Now that you have a model, you can commit it to PMR. These instructions assume that you have Mercurial installed as per above (*see* **Note 11**). First you will make a local copy of the Workspace (even though it is empty) in a local (to your file system) repository via Mercurial. Then you will put your files in that local folder, and add and commit them to your local repository. You will then push your local repository back to the PMR workspace, updating it with the new files.

1. Obtain the URI for the workspace you defined in Subheading 2 above. Do this by logging in to PMR, clicking "My Workspaces", and then following the link to your Workspace. The URI will be listed under "URI for mercurial clone/pull/push".
2. Open a command window and navigate to where you would like your local repository to be. Issue the following command to make a local copy of the Workspace:

```
hg clone <Workspace URI>
```

Where <Workspace URI> is replaced with the URI you found in **step 1**. You will be required to log into PMR via Mercurial in order for this to occur—use your PMR login and password. Your local repository should be created with a folder name specified by the Id that you specified when you created the Workspace.

3. Copy the files that you produced in Subheading 3.3 to the folder that has been created for your Workspace.
4. Issue the following command to add them to your local Mercurial repository:

   ```
   hg add
   ```

   You should see some confirmation messages as the files are added.
5. Any changes that you are happy with, such as adding files here, should be committed to your local repository. Issue the following command to do this (*see* **Note 12**):

   ```
   hg commit
   ```

   On Windows, a Notepad file is opened. This is for you to add a comment to the commit explaining what the purpose of the commit is (such as "Initial adding of files"). The commit message will be saved along with your commit to help others (and yourself, in the distant future) to determine what has changed and why.
6. The local repository is now updated with the files. However, nothing is on PMR as yet. Issue the following command to push your changes from your local repository to the Workspace on PMR:

   ```
   hg push
   ```

   Once again, logging into PMR with your user name and password is required, this time in order to authorize the update of the remote repository.
7. Navigate with your web browser to the Workspace URI on PMR and you should now see that the model files have all been added.

3.5 Grant Access to Collaborators

It is possible to grant access for collaborators to a Workspace, as long as they have a PMR user login.

1. From the Workspace web page click the "sharing" button in the button bar in the upper middle of the page.
2. Permissions for various users and groups are displayed in tabular form. You can search for a particular user if you know their PMR username. If you have a collaborator, do this and

then select the permissions that you would like to grant them. At the time of writing these are `add`, `edit`, `hg push`, and `view`. Add, edit, and view relate to the workspace itself, whereas hg push relates to the ability to push changesets to a local repository with the Workspace (*see* **Note 13**).

3. Click "Save" when you are happy with the changes you have made, or "Cancel".

3.6 Model Simple Gene Regulation

Now you will add the elements for simple gene regulation, including a promoter, messenger RNA, ribosome binding site (RBS), and a protein coding sequence (CDS). These will be added to a new model, which begins as a copy of the old one. The genetic components will be reused from the Bugbuster model. These and subsequent steps could be performed by collaborators that have been granted access from anywhere in the world, once they have cloned your Workspace.

1. Create a new model file by copying "GFPModel.cellml" to "SimpleModel.cellml".
2. Change the "name" attribute of the <model/> tag in "SimpleModel.cellml" to "SimpleModel".
3. Add the concept of a CDS by copying the file "Template_ProteinCDS.cellml" to your local repository folder. Copy also the file "Bugbuster_ProteinCDS_GFP.cellml". This latter file imports the Template for the CDS. Add a CDS to the model by placing the following code inside the <model/> tag of "SimpleModel.cellml":

```
<import xlink:href="Bugbuster_ProteinCDS_GFP.
cellml">
  <component name="ProteinCDS_GFP" component_
  ref="Bugbuster_ProteinCDS_GFP" />
</import>
```

   (*see* **Note 14**).

4. Add the concept of an RBS by copying the files "Bugbuster_RBS_GFP.cellml" and "Template_RBS.cellml" to your local repository folder. Add an RBS to the model by placing the following code inside the <model/> tag of "SimpleModel.cellml":

```
<import xlink:href="Bugbuster_RBS_GFP.cellml">
  <component name="RBS_GFP" component_ref=
  "Bugbuster_RBS_GFP" />
</import>
```

5. Add RNA to the model with the following code:

```
<import xlink:href="Template_Species.cellml">
  <component name="RNA_GFP" component_
  ref="Template_Species" />
</import>
```

Note that you don't have to add any more files to do this—since you have already added the concept of a Species to the model, you can simply re-import the Template_Species component under the name "RNA_GFP" (*see* **Note 15**).

6. Add an initial condition for the "RNA_GFP" species by adding the following code to the previously created "SpeciesInitialConcentrations" component:

```
<variable name="RNA_GFP" units="nM" initial_
value="0" public_interface="out" />
```

7. Add the concept of a constitutive promoter by copying the files "Template_Promoter_Constitutive.cellml" and "Bugbuster_Promoter1.cellml" to your local repository folder. Add the following code to your model, inside the <model/> tag:

```
<import xlink:href="Bugbuster_Promoter1.cellml">
  <component name="Constitutive_Promoter"
  component_ref="Bugbuster_Promoter1" />
</import>
```

8. The promoter will need a concept of cellular volume in order to operate. To introduce the concept of a cell, copy the "Template_Chassis_WellStirredBag.cellml" file and the "Chassis_Bacillus.cellml" files to your model-building folder. Then add the following code inside the <model/> tag:

```
<import xlink:href="Chassis_Bacillus.cellml">
  <component name="ChassisBacillus" component_
  ref="Chassis_Bacillus" />
</import>
```

The Chassis_Bacillus file imports the Template_Chassis file and parameterizes it for a Bacillus cell type.

9. Connect the imported Bacillus cell volume to the promoter by adding the following connection:

```
<connection>
  <map_components component_1="Constitutive_
  Promoter" component_2="ChassisBacillus"/>
  <map_variables variable_1="localVolume"
  variable_2="cellVolume"/>
</connection>
```

10. In a similar manner to adding the interface component for GFP earlier, you should add an interface component for the RNA_GFP:

```
<component name="RNA_GFP_interface">
  <variable name="JPlusProduction" units=
  "nM_per_s" public_interface="in"/>
  <variable name="JGain" units="nM_per_s"
  public_interface="out"/>
```

```
    <variable name="JMinusDegradation" units="nM_
    per_s" public_interface="in"/>
      <math xmlns="http://www.w3.org/1998/Math/
      MathML">
        <apply>
          <eq/>
          <ci>JGain</ci>
          <apply>
            <minus/>
            <ci>JPlusProduction</ci>
            <ci>JMinusDegradation</ci>
          </apply>
        </apply>
      </math>
  </component>
```

This interface component has both a JPlus and a JMinus input. The JPlus input will come from the promoter, and the JMinus from an RNA degradation process.

11. Add the concept of RNA degradation by copying the file "Bioenvironment_Degradation_BugbusterRNA1.cellml" to the local repository folder. The degradation component should be imported by adding the following code to your model:

```
<import xlink:href="Bioenvironment_
Degradation_BugbusterRNA1.cellml">
  <component name="Bioenvironment_Degradation
  _RNA_GFP" component_ref="Bioenvironment_
  Degradation_BugbusterRNA1" />
</import>
```

12. Now the model contains all the elements for simulation, but many are not connected yet. Add the following three connections:

```
<connection>
  <map_components component_1="RNA_GFP"
  component_2="Time"/>
    <map_variables variable_1="time" variable_
    2="time"/>
</connection>

<connection>
  <map_components component_1="RNA_GFP"
  component_2="SpeciesInitialConcentrations"/>
    <map_variables variable_1="concentration
    InitialValue" variable_2="RNA_GFP"/>
</connection>

<connection>
```

```
  <map_components component_1="RNA_GFP"
  component_2="RNA_GFP_interface"/>
    <map_variables variable_1="JGain"
    variable_2="JGain"/>
</connection>
```

These connections, respectively, add the model's notion of time to the RNA, the initial concentration, and also connect our RNA_interface component to the RNA, so that the sum of the JPlus and JMinus variables becomes the net RNA flux.

13. Add the following connections, so that the production of RNA from the promoter becomes the JPlus flux for the RNA_interface, and the degradation of RNA becomes the JMinus flux:

```
<connection>
  <map_components component_1="RNA_GFP_
  interface" component_2="Constitutive_
  Promoter"/>
    <map_variables variable_1="JPlusProduction"
    variable_2="JRNA"/>
</connection>

<connection>
  <map_components component_1="RNA_GFP_
  interface" component_2="Bioenvironment_
  Degradation_RNA_GFP"/>
    <map_variables variable_1="JMinus
    Degradation" variable_2="J"/>
</connection>
```

14. Add the following connection, so that the RNA Degradation component acts on the amount of RNA_GFP present:

```
<connection>
  <map_components component_1="Bioenvironment_
  Degradation_RNA_GFP" component_2="RNA_GFP"/>
    <map_variables variable_1="concentration"
    variable_2="concentration"/>
</connection>
```

15. The RBS component needs to connect to the model's concept of cellular volume, and to know which RNA concentration to use in order to calculate the genetic translation rate. Add the following two connections to the model to provide this information:

```
<connection>
  <map_components component_1="RBS_GFP"
  component_2="ChassisBacillus"/>
    <map_variables variable_1="localVolume"
    variable_2="cellVolume"/>
</connection>
```

```
  <connection>
    <map_components      component_1="RBS_GFP"
    component_2="RNA_GFP"/>
      <map_variables variable_1="RNA" variable_
      2="concentration"/>
</connection>
```

16. The CDS component also needs to know the model's concept of cellular volume, and which RBS is directly upstream of it (via translating ribosomes per second or RiPs). Provide this information with the following connections:

```
  <connection>
    <map_components    component_1="ProteinCDS_
    GFP" component_2="ChassisBacillus"/>
      <map_variables   variable_1="localVolume"
      variable_2="cellVolume"/>
  </connection>

  <connection>
    <map_components    component_1="ProteinCDS_
    GFP" component_2="RBS_GFP"/>
      <map_variables   variable_1="hostRNARiPs"
      variable_2="RNARiPs"/>
  </connection>
```

17. Previously, the amount of GFP in the cell was set at 1 nM, and this degraded over time to zero. Now that the model contains genetic elements that produce GFP, the initial concentration of GFP should be set to zero, and a JPlus flux from the CDS should be added to the GFP_interface component. Therefore, adjust the GFP variable declaration in the SpeciesInitialConcentrations component to be the following:

```
  <variable   name="GFP"   units="nM"   initial_
  value="0" public_interface="out" />
```

The GFP_interface component declaration should be modified by the following:

```
  <component name="GFP_interface">
      <variable  name="JGain"  units="nM_per_s"
      public_interface="out"/>
      <variable name="JMinusDegradation" units=
      "nM_per_s" public_interface="in"/>
      <variable  name="JPlusProduction"  units=
      "nM_per_s" public_interface="in"/>
     <math  xmlns="http://www.w3.org/1998/Math/
     MathML">
      <apply>
        <eq/>
        <ci>JGain</ci>
```

```
            <apply>
              <minus/>
              <ci>JPlusProduction</ci>
              <ci>JMinusDegradation</ci>
            </apply>
          </apply>
        </math>
    </component>
```

and connect the new JPlus variable to the output of the CDS:

```
<connection>
  <map_components component_1="GFP_interface"
   component_2="ProteinCDS_GFP"/>
  <map_variables variable_1="JPlusProduction"
  variable_2="J"/>
</connection>
```

If the model is now saved and simulated in a CellML-ready tool, it will produce an increasing concentration of GFP that goes to a steady state of about 1,380 nM at around 5,000 s.

3.7 Commit and Push All Changes

Local commits can of course be performed at any time during local development. If new files have been added, such as in the previous section then

1. An
   `hg add`
   command should be issued, as you did after the construction of the previous model.
2. An
   `hg commit`
   command should be issued whenever you would like to perform a local commit, as it was in Subheading 3.4. You will have to enter some commit comments.
3. An
   `hg push`
   command should be issued when development has progressed to the point that the model builder would like to push the changes back to the PMR Workspace. Assuming all previous steps have been performed, do this now. You will be required to log in with your PMR username and password.

3.8 Replace the RBS Virtual Part

Assume that you wish to replace the RBS in the model with a more powerful one. You could edit the RBS model file and adjust the rate constants therein; however in a larger model this would also change other RBSes in the model that rely on an import of that same Virtual Part. Here you will create a new, more powerful part in a separate file and replace the existing RBS with the new one

1. Copy the “Bugbuster_RBS_GFP.cellml” file to a new file “MorePower_RBS_GFP.cellml”.
2. Open the “MorePower_RBS_GFP.cellml” file in a text editor and replace the name attribute of the model tag with `name="MorePower_RBS_GFP"`
3. Remove the cmeta:id attribute of the model tag.
4. Remove the <rdf:RDF> tag containing author information (*see* **Note 16**).
5. Rename the <component/> element from “Bugbuster_RBS_GFP” to “MorePower_RBS_GFP”.
6. In that same component, increase the `initial_value` of the “k” variable from 0.145 to 14.5 (a 100-fold increase in translation speed!).
7. In the <group/> tag, ensure that the top-level “component _ref” element points to “MorePower_RBS_GFP”.
8. In the <connection/> element, ensure that the “component_2” attribute of the <map_components/> tag is set to “MorePower_RBS_GFP”.
9. Copy the “SimpleModel.cellml” file to a new file: “MorePowerModel.cellml”. You now have a new model file.
10. In the “MorePowerModel.cellml”, change the “name” attribute of the <model/> tag to “MorePowerModel”.
11. In that same file, find the element that is declared:

    ```
    <import xlink:href="Bugbuster_RBS_GFP.cellml">
      <component name="RBS_GFP" component_ref=
      "Bugbuster_RBS_GFP" />
    </import>
    ```

    and change it to

    ```
    <import xlink:href="MorePower_RBS_GFP.cellml">
      <component name="RBS_GFP" component_ref=
      "MorePower_RBS_GFP" />
    </import>
    ```

    This will replace the old Bugbuster RBS with the more powerful version.
12. Be sure to use `hg add`, `hg commit` and `hg push` to commit your changes back to the PMR Workspace.
    If simulated now, the MorePower model produces a steady-state value of GFP of about 138,000 nM, or 138 μM.

3.9 Expose the Workspace to the General Public

Once a set of models in a Workspace is ready for release to the general public, one or more Exposures can be made. Exposures provide links to view and download models, as well as provide an online documentation source for the model. Here you will create an Exposure for the first model that you created above. Additional Exposures can be created for the other models if desired.

1. Documentation for the Exposure can be provided by a sample . html page that you can include in your Workspace. Go to your local repository and create a file “exposure.html”.
2. Create a documentation file for your models in “exposure. html”. Here is an example template that you can start with:

```
<html>
<head>
<title>Example GFP Model</title>
</head>
<body>
<h1>Documentation</h1>
Here is some documentation.
<h2>Sub-Documentation</h2>
..and some sub-documentation.
</body>
</html>
```

3. Add, commit, and push the “exposure.html” to your Workspace (as detailed above).
4. Ensure that your Workspace is public by navigating to the Workspace page and selecting the element in top right of the bar with the caption: “state private”. Select the drop-down option “submit for publication”.
5. When your Workspace has been published (meaning that now it and the code that it contains is accessible to the general public—*see* **Note 17**), click the “history” button in the menu bar.
6. A list of changesets will be displayed. You can make an Exposure that is linked to any changeset. Select the [files] part of the changeset that you would like to make an Exposure for (for this example, the latest one).
7. The top right menu bar element “workspace actions” will now display “create exposure” when clicked. Do this and select “create exposure”.
8. In the “Exposure main view” box, fill out “HTML annotator” for View Generator, and “exposure.html” for Generator Source.
9. Exposures provide the option to nominate specific files of interest in a Workspace. To do this for the first model “GFPModel.cellml”, select that file in the New Exposure File Entry area as “File”, and select “CellML file” as “File Type”. Click on the Add button.
10. The next screen allows you to provide more information about that file. For now, you can skip adding more information, but it would be good to similarly add the other two models in your

Workspace. To add another model, click on the "Add File" button. Another New Exposure File Entry section will appear—fill this one out with "SimpleModel.cellml" and "CellML file" and click the "Update" button.

11. Similarly, add "MorePowerModel.cellml" to the Exposure via a New File Entry section. You should now have three of those sections.
12. Click "Build" at the bottom of the screen.
13. Your new exposure should be displayed. Note that the Exposure is private by default, so you will have to "submit it to make it public" in order for it to be viewable by the outside world, much as you did for the Workspace previously. Note that on the right-hand side there is a "Navigation" panel with the three model file names displayed. These can be clicked on to take viewers to pages where models can be viewed with a range of different plug-ins, depends on what is installed on PMR. In the future, it may also be possible to simulate models from PMR, or view the models in new ways with different plug-ins. Therefore adding the main files of interest to the Navigation area in the manner described in the above steps is recommended for future utility.

4 Notes

1. However, if a species and initial condition combination was useful to reuse (e.g., the known physiological initial condition under "normal" conditions for a species in a particular cell type), then a Virtual Part could be made to encapsulate both the variable's ODE and the initial condition in order to facilitate reuse of both in concert.
2. At the time of writing, several CellML modeling environments are available. These can be found at the CellML web site (www.cellml.org) under Tools, and then Modelling Environments. "OpenCell" is capable of building any of the models in this chapter; however some users find it difficult to use. "COR" is more user-friendly, but does not cater for CellML 1.1 models (i.e., with imports). However, it may still be useful for producing monolithic models which can be federated into CellML 1.1 models at a later time. "OpenCOR" is currently under development (available at www.opencor.ws), and aims to combine the advantages of both OpenCell and COR, and incorporate new useful features. For the more programmatically inclined, it is also possible to script model creation using the CellML API (see http://www.cellml.org/tools/api). This chapter will not assume any particular tool but will describe the raw CellML

code that might be produced by such a tool, or entered directly into a file with your favorite text editor.

3. Installing Mercurial or any CellML-editing tool will require administrator access to your machine. If your working environment does not grant you such access, you may have to enlist the assistance of your local IT department before starting the examples in this chapter.
4. Tortoise provides a GUI (Graphical User Interface) to Mercurial, which can be helpful for new users or for when projects become complex. However, since it is only available for Windows users, we will not describe its use in this chapter.
5. You will need to enable cookies in your web browser, if they are not already enabled, in order to navigate PMR while logged in.
6. Workspace Ids are unique to the PMR user (hence two users can have the same Id, but these would point to different Workspaces), and are case sensitive.
7. Your new Workspace will contain no files (and therefore no model files). It is also possible to create a new Workspace as a "fork" of an existing Workspace. This is particularly useful if you will be extending or modifying models from another Workspace. You achieve this by navigating to the Workspace of interest, and pushing "fork" in the button bar in the upper middle of the page. You will be prompted to give the Id of the new Workspace, and you can then press the "fork" button to add the new Workspace as a fork of the existing one. This fork will contain files, being a fork of the underlying Mercurial repository that implements the Workspace that was forked.
8. There are a number of other options here. One is to use PMR to create a fork as described in **Note 7**. Another is to use Mercurial to make a local clone of the Bugbuster workspace. That workspace is set up to use the "Embedded Workspaces" feature, which means that it contains links to other workspaces (for the Template models, mainly) to form a hierarchy of Workspaces which can all be updated from your local repository. Both local clones and Embedded Workspaces are advanced uses of PMR and will not be covered directly in this chapter.
9. If a unit is used by the top-level model, it needs to be either defined in that model, or imported from one of the imported models. As far as the code is concerned, it does not matter where the unit is defined or imported; however for maintainability you may find it useful to keep the definitions somewhere specific, such as near the start or end of the model file. A common source of error during validation (which is often performed prior to simulating the model) is to have declared the same unit definition twice, so be sure to only declare it once, or import it once, but not both.

10. One could follow any convention that they liked with the naming of these variables. From experience, we have found that a useful custom is to define a net flux as "JGain", and other fluxes beginning with "JPlus" or "JMinus" depending on how they influence the net flux. Naming the individual fluxes in this way can assist ensuring that we add or subtract each flux appropriately when coding the flux aggregation equation.
11. If you are unsure if Mercurial is installed correctly you can try issuing the command

    ```
    hg
    ```

    at the command prompt. If you get some Mercurial information, then it is installed correctly.
12. If you get an error message about the user name not being defined, ensure that you have created an Mercurial.ini file as in Subheading 2.1
13. We suggest using the view permission to allow "read" access, and "hg push" to allow write/update/delete.
14. It does not matter where in the model you place this code, as long as it is a direct child of the <model/> tag.
15. You could have called the component simply "RNA"; however since a model could be extended to include several RNA for different proteins, we recommend naming it more specifically: thus "RNA_GFP".
16. If you prefer you could change the cmeta:id attribute of the model tag to something appropriate and edit the <rdf:RDF/> tag to contain author information (e.g., about you).
17. If you are concerned about making your entire Workspace public, another option is to place your releasable models in a new Workspace, either by the processes described in this chapter, or by "forking" an existing Workspace (which copies the revision history up to the current point as well), and Exposing your models from there.

Acknowledgments

This work was supported by the Maurice Wilkins Centre for Molecular Biodiscovery, New Zealand.

References

1. Marchisio MA, Stelling J (2008) Computational design of synthetic gene circuits with composable parts. Bioinformatics 24: 1903–1910
2. Peccoud J, Blauvelt MF, Cai Y, Cooper KL, Crasta O, DeLalla EC, Evans C, Folkerts O, Lyons BM, Mane SP, Shelton R, Sweede MA, Waldon SA (2008) Targeted development of registries of biological parts. PLoS One 3:e2671
3. Cooling MT, Rouilly V, Misirli G, Lawson J, Yu T, Hallinan J, Wipat A (2010) Standard virtual biological parts: a repository of modular

model components for synthetic biology. Bioinformatics 26:925–931

4. Cuellar AA, Lloyd CM, Nielsen PF, Bullivant DP, Nickerson DP, Hunter PJ (2003) An overview of CellML 1.1, a Biological Model Description Language. Simulation 79:740–747
5. Cooling MT, Hunter P, Crampin EJ (2008) Modeling biological modularity with CellML. IET Syst Biol 2:73–79
6. Hunter PJ, Borg TK (2003) Integration from proteins to organs: the Physiome Project. Nat Rev Mol Cell Biol 4:237–243
7. Cooling MT, Hunter P, Crampin EJ (2009) Sensitivity of NFAT cycling to cytosolic calcium concentration: implications for hypertrophic signals in cardiac myocytes. Biophys J 96:2095–2104
8. Nickerson DP, Nash MP, Nielsen PF, Smith N, Hunter P (2006) Computational multiscale modeling in the IUPS Physiome Project: modeling cardiac electromechanics. IBM J Res Dev 50:617–630
9. Yu T, Lloyd CM, Nickerson DP, Cooling MT, Miller AK, Garny A, Terkildsen JR, Lawson J, Britten RD, Hunter PJ, Nielsen PMF (2011) The Physiome Model Repository 2. Bioinformatics 27:743–744
10. Miller AK, Yu T, Britten R, Cooling MT, Lawson J, Cowan D, Garny A, Halstead MDB, Hunter PJ, Nickerson DP, Nunns G, Wimalaratne SM, Nielsen PMF (2011) Revision history aware repositories of computational models of biological systems. BMC Bioinformatics 12. doi:10.1186/1471-2105-12-22
11. O'Sullivan B (2007) Distributed revision control with Mercurial, Mercurial project

Chapter 16

The Synthetic Biology Open Language

Chris Myers, Kevin Clancy, Goksel Misirli, Ernst Oberortner, Matthew Pocock, Jacqueline Quinn, Nicholas Roehner, and Herbert M. Sauro

Abstract

The design and construction of engineered organisms is an emerging new discipline called synthetic biology and holds considerable promise as a new technological platform. The design of biologically engineered systems is however nontrivial, requiring contributions from a wide array of disciplines. One particular issue that confronts synthetic biologists is the ability to unambiguously describe novel designs such that they can be reengineered by a third-party. For this reason, the *synthetic biology open language* (SBOL) was developed as a community wide standard for formally representing biological designs. A design created by one engineering team can be transmitted electronically to another who can then use this design to reproduce the experimental results. The development and the community of the SBOL standard started in 2008 and has since grown in use with now over 80 participants, including international, academic, and industrial interests. SBOL has stimulated the development of repositories and software tools to help synthetic biologists in their design efforts. This chapter summarizes the latest developments and future of the SBOL standard and its supporting infrastructure.

Key words Synthetic biology, Standards, Engineering design

1 Introduction

Within the synthetic biology community, there is a pressing need for the ability to share and communicate designs in a simple and unambiguous manner. Typically, users need software support for a number of design steps, including *genetic design automation* (GDA) tools and workflows to construct genetic designs, analyze and visualize their behavior, explore design alternatives, and edit and optimize their DNA sequences, as well as, repositories for saving and sharing designs. Clearly, this software must be capable of exchanging information about these designs in a consistent manner.

The biological community has produced many excellent and long-lived standards, include FASTA format [1], GenBank [2], SwissProt [3], PDB [4], and SBML [5], to name a few. Part of the reason for the long-term success of these formats has been a broad agreement on the scope of the data that the format captures and a clear description of how this data is encoded and subsequently interpreted by compliant software. Another aspect of the success of these standards, however, has concerned the ability of software developers to present, visualize, and support user interaction with standardized data in the context of their individual software packages. A well-defined standard allows developers to create software environments that support both the scientific and software development needs of users' projects, and to enable users to accomplish group projects with minimal loss or miscommunication of data during inter-software exchange.

The needs of the synthetic biology community go beyond the scope of the standards already developed by the biological community and include the ability to communicate designs that combine data on biological sequences, functional composition, molecular interactions, expected performance, and experimental context, as well as standardization of the visual representation of these data. To meet these complex data exchange needs, the community has developed a new standard, the *synthetic biology open language* (SBOL).

SBOL is an open standard in that participation in standardization activities is unrestricted to all people or organizations who wish to be involved, all essential information is publicly accessible on the web, and both the standards and all supporting materials can be used, without cost, by anybody, for any purpose. This information includes specifications, use case descriptions, and a Java library for SBOL that is utilized by many of the tools described in this chapter.

2 SBOL Data Model

SBOL is a standard being developed by the synthetic biology community to describe designs of biological *components* and *modules*. The first version of SBOL was ratified by the SBOL Developers Group in 2011 [6, 7]. The current version of the SBOL standard, Version 1.1, is depicted using the *unified modeling language* (UML) in Fig. 1. Version 1.* centers around the hierarchical composition of *DNA components* by annotating their sub-components and *DNA sequences*. The *DNA components* are typed using the *Sequence Ontology* (SO) [8] and can be grouped into *Collections*.

Recently, several extensions to SBOL have been proposed to enable representation of further information about a biological design. Examples include design-specific information about intended and observed regulatory interactions, characterization data and performance measurements, host context information, system and modeling artefacts, and information about assembly and synthesis. Figure 2 shows an abstract UML diagram for a draft of SBOL Version 2.0 with an emphasis on connecting the SBOL extensions more easily.

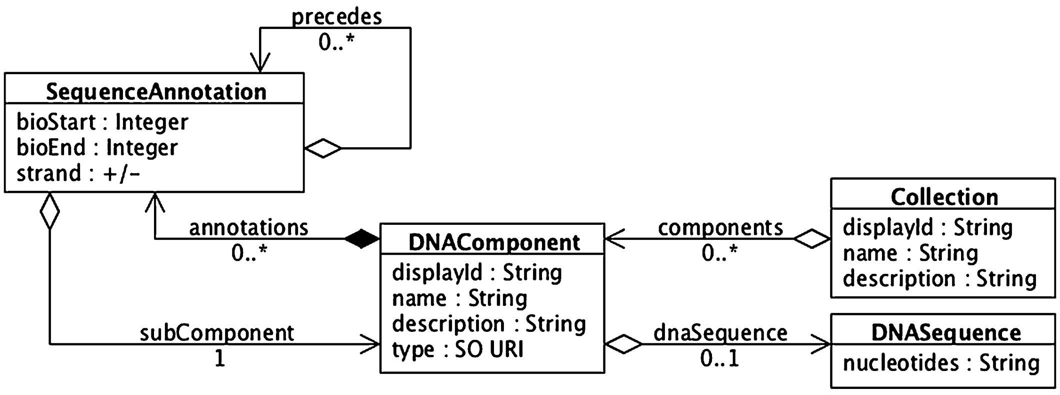


Fig. 1 UML diagram for SBOL Version 1.1. SBOL can document each DNA component with an identifier, name, description, and SO type. The DNA sequence for a DNA component may also be provided and locations on this DNA sequence can be annotated. Each *sequence annotation* is a reference to a DNA sub-component that includes its starting and ending positions on the DNA sequence and which strand it appears on. When the exact DNA sequence of a sub-component or its parent component is not known, its location relative to other sub-components can be specified using a *precedes* relation between annotations. Finally, a collection can be used to group DNA components on the basis of whether, for example, they come from a common source or share the same SO type

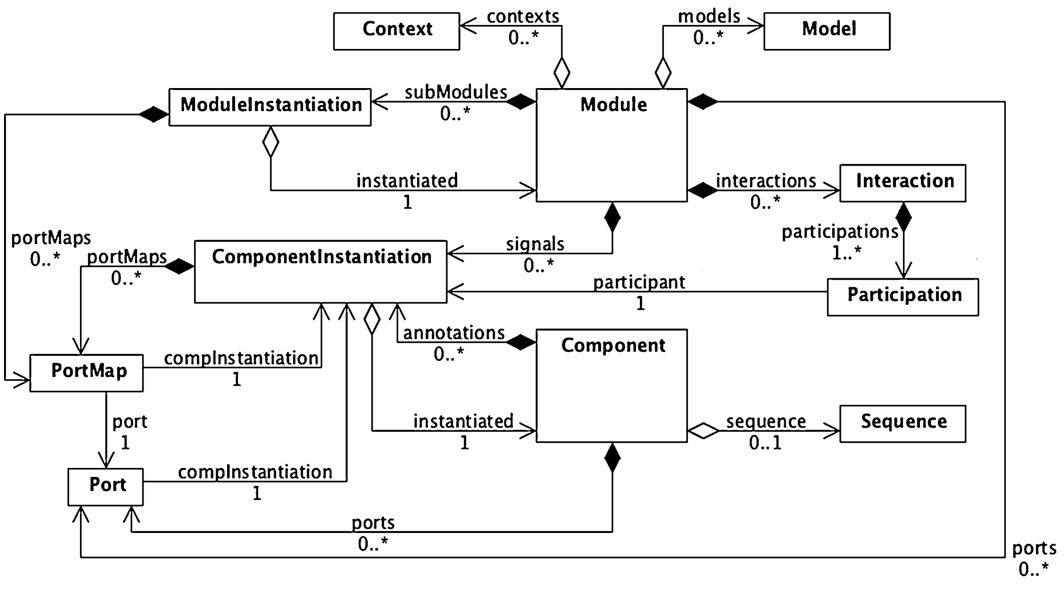


Fig. 2 A proposed UML diagram for SBOL Version 2.0. Components are generalized to represent sequenced components such as DNA, RNA, and protein components, as well as components without a biological sequence such as small molecules and light. Modules are used to instantiate and hierarchically group components and other modules on the basis of their intended function in a biological design. Modules may also include data on interactions between their component instantiations, connections to mathematical models of their behavior, and descriptions of experimental context. Lastly, both components and modules may have ports that allow them to be connected using port maps upon instantiation

In SBOL Version 2.0, components are generalized to represent other sequenced components, such as RNA and protein components, as well as components without a biological sequence, such as small molecules, and even environmental inputs such as light. This generalized model of components captures the elements that make up a design.

In addition to describing the static components that make up a design, SBOL 2.0 can describe biological modules which capture the *dynamic behavior* of a design. A module represents a group of components that together are intended to perform some intended biological function. Complex modules can be composed from simpler modules, just as complex sequence components can be composed from simpler ones.

These components are referred to via *signals*, which instantiate the component within the context of a particular module. For example, a protein component that is a transcription factor may repress a DNA component that is a promoter. Such relationships between signals can be defined using *interactions*. Each interaction has a type taken from the *systems biology ontology* (SBO) [9] and a list of *participations*. Each participation refers to a signal and the role that the signal plays in the interaction. In the previous example, the protein component's role would be repressor while the DNA component's role would be repressed.

Connections between the components in a module and those in its sub-modules are made through *ports* and *port maps*. A port exposes a signal as part of a module's interface, annotating it with directionality (*input*, *output*, or both). A port map wires a signal in a parent module to a port in a sub-module. By wiring the same signal in the parent module to multiple signals from different sub-modules, these sub-modules can be plugged together.

Finally, a module may include *experimental context* and connections to any number of *models*. Context elements include information necessary to deploy the design, such as the strain of the host, the growth medium and any added or absent nutrients required for correct function, the container in which the medium is stored, the environmental conditions, and the measurement device used to study the system. Model elements point to a mathematical model of the module defined outside of SBOL, the language that the model is written in, the type of the modeling framework used, and the role of the model. It is best practice to provide models for each module where possible, and for at least one of these models to be in a standard modeling language, such as SBML.

3 SBOL Visual

SBOL Visual (SBOLv) is the graphical counterpart to SBOL. While SBOL enables the exchange of design data among software programs, SBOLv enables standardized, abstracted, human-readable visualization of designs. SBOLv was first published in 2009 [10], and an updated version was ratified by the SBOL Developers Group in 2013 [11].

The SBOLv standard is comprised of a set of *symbols* that represent common functional abstractions for DNA components used in synthetic biology, such as promoter, coding sequence, and terminator. Each SBOLv symbol corresponds to terms in the SO, complimenting the SBOL data model and enabling straightforward representation of SBOL encoded designs in SBOLv. Symbols are inspired by symbology used in synthetic and molecular biology practice. SBOLv aims to catalog and standardize the depiction of biological components and modules in order to promote greater clarity in graphical representation of biological design both in software tools and free-hand diagrams.

Several software applications have adopted SBOLv as a core component of *graphical user interfaces* (GUIs). Four such applications are represented in Fig. 3. In such applications, SBOLv is used in graphical editors that allow users to compose modules, or as computer generated displays of the output of design tools. SBOLv has also been used in publication figures both composed manually and generated via visualization software [12, 13].

Normative image files for SBOLv symbols are available on the SBOL website at http://sbolstandard.org/visual. While these icons are made available for reference and convenience, use of these particular files is not required for adoption of SBOLv. Software programs that can assist in generating figures in SBOLv can be found at http://sbolstandard.org/visual/tools.

As the concepts, constructs, and terminology of synthetic biology evolve, the SBOLv symbol set is expected to expand and mature. The scope of SBOLv is also likely to expand to cover concepts such as molecular interaction, providing a framework for the representation of designs encoded in SBOL 2.0.

4 SBOL Adoption

The adoption of SBOL has been greatly facilitated by being developed by a representative group of stakeholders, drawn from all sections of the synthetic biology community, each with diverse and varied needs. The *SBOL Developers Group* is composed of a diverse group of researchers, developers, and other stakeholders from academic, government, and commercial organizations. At

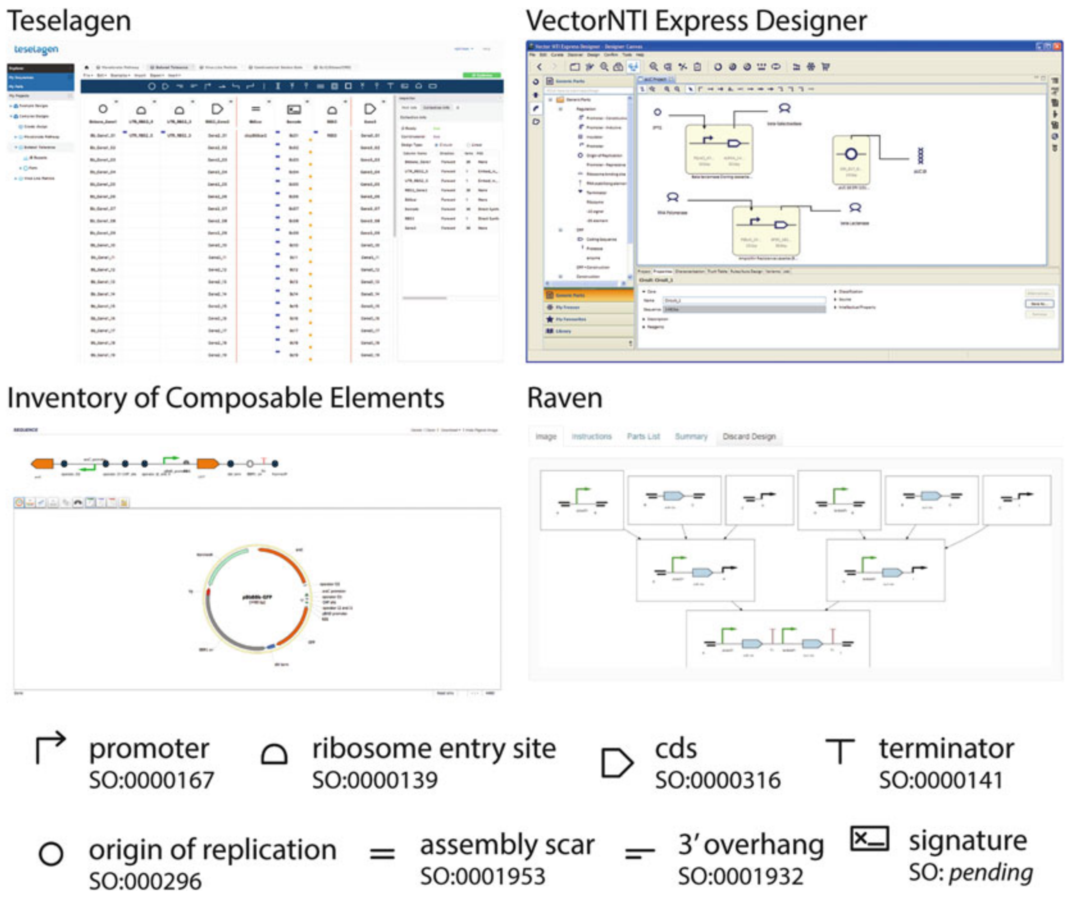


Fig. 3 Screenshots from software applications that have adopted SBOLv, and an abridged glossary of symbols appearing in the screenshots. Example applications (*starting at top left moving clockwise*) that use SBOLv symbols as GUI editable elements are Teselagen's DNA design and assembly platform (www.teselagen.com), Life Technologies' VectorNTI Express Designer (www.lifetechnologies.com), the Joint BioEnergy Institute's ICE data repository (public-registry.jbei.org), and Boston University's Raven assembly planner (www.ravencad.org)

the time of writing, the SBOL Developers Group has more than 80 members from 29 organizations (16 academic, 11 commercial, and 2 government labs), who work across organizational and international boundaries to set priorities and reach agreement on the vision, development, and future for the standard. While the SBOL Developers Group is led by a chair and five elected editors, all decisions are made democratically by the SBOL Developers Group members.

To date, SBOL and SBOLv have been adopted by nearly 20 synthetic biology software tools. This section briefly describes the use of SBOL in a few of these tools to illustrate how SBOL adoption has benefited these efforts. It is not meant as an exhaustive list as the tools that support SBOL is changing rapidly.

4.1 Repositories

The first class of software tools that have added support for SBOL are repositories. These repositories allow information about genetic designs to be both stored and retrieved in the SBOL format. This section briefly describes the Virtual Parts Repository and JBEI-ICE repository.

4.1.1 Virtual Parts Repository

The Virtual Parts Repository (http://www.virtualparts.org) publishes information about biological parts and their interactions. These are represented by modular models, called Standard Virtual Parts (SVPs–*see* Chapter 15) [14]. SVPs have defined inputs and outputs, and can be composed computationally to create simulatable models for the composite biological systems. This enables *GDA* tools to simulate the behavior of these composite models, enabling model-driven design. Additionally, tools can automate combinatorial searches of SVPs where simulations achieve user-specified behavior. The selected models act as blueprints for genetic circuits and can be used to derive DNA sequences necessary to encode desired behaviors [15]. DNA sequences of the resulting genetic circuits, along with their models, can themselves be deposited at the Virtual Parts Repository.

The repository is publicly accessible via a Web interface. This allows manual browsing of the SVPs and manual downloading of associated data files. Additionally, genetic circuit designs are visualized using Pigeon [16], a design visualizer for synthetic biology that conforms to the SBOLv standard. SVPs can be queried computationally using a REST-based Web service interface. A Java client to the REST-based Web service interface is also available. While internally SVPs are represented in a custom data structure, SBOL and SBML are used to exchange genetic descriptions of these parts and their models.

4.1.2 Joint BioEnergy Institute Inventory of Composable Elements

The *Joint BioEnergy Institute Inventory of Composable Elements* (JBEI-ICE) [17] is an open source registry platform for managing information about biological parts. The software implementation, GD-ICE (http://code.google.com/p/gd-ice/), is a web accessible repository designed to handle synthetic biology parts, devices, and constructs. GD-ICE has recently been modified to support import and export of SBOL data files. This has permitted users to import their data from other software tools into the repository and share their data with other project participants. GD-ICE is currently the platform of choice for hosting synthetic biology repositories so it was a relatively simple matter to export and import the SBOL design files into other partner repositories for local use.

4.2 GDA Tools

One of the major challenges for synthetic biology is to automate the design of biological systems. Without intelligent computational tools to automate the processes of selecting, composing, and analyzing biological components and modules to meet a specification, it is much more difficult to efficiently and rigorously design biological systems of ever increasing complexity. The usefulness of any GDA software, however, is limited in the absence of standards for computationally representing synthetic biological designs and enabling their exchange between different tools. From a GDA perspective, one of the central purposes of standards, such as SBOL, is to enable an ecosystem in which software tools can focus on different aspects of the same design and complement each other's strengths. This section examines two GDA tools, Eugene [18] and iBioSim [19], and discusses how their use of SBOL enables them to be interfaced with other tools as both providers and consumers of design services.

4.2.1 Eugene

Eugene [18] is a design specification language for synthetic biology that can be used to specify genetic components at different levels of abstraction, define constraints on how these components can be composed, and manage the flow of control when synthesizing designs. In addition, Eugene includes an API that can be used to automatically generate all possible composite genetic components (devices) from a library of components and user-defined constraints on their composition. Prior to its use of SBOL, Eugene already possessed a genetic data model combining both structural and functional data. However, at this time there was no suitable community standard capturing both structural and functional genetic data in a hierarchical, modular fashion, hampering the use of Eugene by third-party tools. When the SBOL 1.1 standard was developed, Eugene became an early SBOL adopter, enabling its use as a service in a wide range of tools.

After the mapping of its data model to SBOL Version 1.1, Eugene can now import and export data on genetic components between a wider variety of software tools and repositories for synthetic biology. As demonstrated in [7], Eugene can *import* partial SBOL designs from SBOL Designer—a tool for creating and visualizing both partial and complete designs natively in SBOL. Next, it can generate many complete variant designs by permuting additional genetic components imported from the iGEM Registry. The generated variants can then, for example, be *exported* to SBOL, making it possible to import them in iBioSim (as described below).

While SBOL 1.1 does not capture certain classes of functional data such as regulatory interactions between genetic components, SBOL 2.0 is currently under development to extend its representation of function for design. Based on its data model and provided constraints, Eugene is already able to take regulatory interactions into account when synthesizing designs.

4.2.2 iBioSim

iBioSim [19] is a GDA software suite with a wide range of design capabilities, including construction of SBML models, ODE and stochastic simulation, stochastic model checking, model learning from time series data, and more recently, model annotation and genetic technology mapping (matching and composing genetic modules from a library to satisfy a specification). Prior to its incorporation of SBOL, iBioSim's data model primarily described genetic circuits using biochemical models, mathematical equations, and parameters. Missing from these functional descriptions, however, were structural data such as the nucleotide sequences of DNA components and their hierarchical relationship via sequence annotations.

Following its extension with SBOL, iBioSim's data model now represents both the function and structure of genetic circuits in a hierarchical, modular fashion that is useful for engineering biological systems. This representation is accomplished by annotating SBML models with the identifiers for SBOL DNA components [20], thereby directly associating elements in the functional layer of a genetic circuit design with elements in its structural layer. Consequently, iBioSim can be used to build libraries of genetic modules from SBML and SBOL. Furthermore, it draws upon these libraries during genetic technology mapping to find the set of library modules that optimally satisfy a specification for a genetic circuit and compose these modules to obtain the hypothetically best genetic circuit design. Through SBOL and the Java library, libSBOLj, iBioSim's capabilities have been expanded in a manner that is conducive to the exchange of its design output. For example, the SBML models and SBOL DNA components of iBioSim's genetic circuit designs can now be stored in repositories such as the Virtual Parts [14] repository, and the DNA components can also be stored in the JBEI-ICE repository [17].

Lastly, in terms of tool interoperability, iBioSim can now interface via SBOL with sequence editors/optimizers and other GDA tools that can be used to explore possible structural implementations of a design. In particular, iBioSim has interfaced with Eugene and with the Vector NTI Express Designer. In the case of Eugene, iBioSim can import collections of composite DNA components that have been combinatorially generated with Eugene. iBioSim can then be used to annotate models of these components' behavior with the components themselves. The end result is a library of genetic modules that can later serve as an input to iBioSim's technology mapping tool. In the case of Vector NTI (as described below), iBioSim can export the structural portion of the output of its technology mapping tool to Vector NTI, which can then further refine the DNA-level design, for example, by performing host-specific codon optimization.

4.3 Sequence Editors and Optimizers

After a genetic circuit has been designed, sequence editors and optimizers are used to produce the final DNA sequence for construction or synthesis. This section briefly describes Vector NTI and the j5 software tools which can be used for this task.

4.3.1 Vector NTI Express Designer

Vector NTI Express Designer is an example of traditional molecular biology desktop software that has been extended to support synthetic biology design (http://www.lifetechnologies.com/us/en/home/life-science/cloning/vector-nti-software/vector-nti-express-designer-software.html). Under traditional molecular biology practices, sequences are annotated with features and these features are manipulated into cloned constructs. Many of the target users are unfamiliar with synthetic biology practices or concepts, so the software needs to provide support for both of these approaches while allowing users to easily transition between the two. In order to facilitate this transition, sequence features are associated with SO terms in the same way as SBOL DNA components. Thus, SO terms provide a bridge between traditional and synthetic biology models.

Vector NTI Express Designer provides a wide range of sequence analysis and manipulation tools, with special emphasis on (1) tools that can assist with the design of new parts, devices, and circuits, (2) refactor existing biological sequences and operons, (3) perform codon optimization [21], and (4) evaluate cloning approaches for designs. By applying these tools, users of Vector NTI Express Designer can develop parts collections into designs for devices and circuits. Implementing SBOLv as a standardized visual metaphor for synthetic biology data has allowed these users to compare designs from other tools and to determine that they have smoothly transferred into Vector NTI.

The use of SBOL has enabled inter-tool exchange with Vector NTI Express Designer. For example, it can now codon optimize the DNA sequences for SBOL DNA components exported by GDA tools, such as iBioSim. Vector NTI Express can then export the optimized design in SBOL format and send it to a repository such as JBEI-ICE or Virtual Parts.

4.3.2 j5 Tools

The Joint BioEnergy Institute (JBEI) has supported the development and deployment of the web-based software tool, j5 [22], which automates the design of scar-less multipart DNA assembly protocols including SLIC, Gibson, CPEC, and Golden Gate. Based upon the imported design files, J5 can be used to evaluate cloning methods that would permit construction of the planned devices by considering cost optimization, the enforcement of design specification rules, hierarchical assembly strategies to mitigate likely assembly errors, and the instruction of manual or automated construction of scar-less combinatorial DNA libraries.

The j5 tools also include Device Editor [23], a web-based bioCAD software with a graphical front end that is used with either GD-ICE repositories or the J5 software. Device editor was used to demonstrate how biological designs from j5 are developed and to display the elements of these designs. Use of the SBOLv symbols again simplified the process of comparing design outcomes between the different software to validate the correct representation of the designs.

4.4 Synthetic Biology Workflow Tools

Workflows are sequences of connected activities where each activity represents one step that needs to be taken in the design phase of novel biological systems. Every activity is enabled after the completion of another activity. Today, several synthetic biology software tools exist that tackle the involved problems of the individual design steps or activities. The tools are mostly loosely coupled, and automated data exchange is hard to achieve if the tools' data models differ. This section presents two synthetic biology workflows, Clotho and TASBE, and explains their relation to the SBOL standard. The Clotho platform and the TASBE tool-chain are initial exploratory approaches toward connecting various synthetic biology design tools. Both efforts demonstrate the need for a standardized data model to enable the data exchange between each step in the workflow, to enable full automation.

4.4.1 Clotho

Clotho [24, 25] motivates the need for community standards, modular design tools, and rigorous design flows. The Clotho platform represents biological objects using an underlying data model that is loosely based on SBOL. A key feature of the Clotho platform-based approach is the decoupling of tools (so-called "Apps") from the data. The platform provides APIs enabling the development of Apps, where each App is tailored to specific design problems and activities in synthetic biology. The APIs enable creating, reading, updating, and deleting objects in the data model, and persistence of this data in a relational database.

The current Clotho repertoire comprises various Apps to import, view, modify, assemble, create, and export novel DNA sequences, supporting biologists to design novel genetic systems. Clotho's *Hermes* App enables it to (1) import data from SBOL files and to store the data in Clotho's database and (2) to export data from Clotho's database to SBOL files. Hence, the *Hermes* App enables Clotho for a round-trip mapping between the SBOL and the Clotho data objects.

4.4.2 TASBE

TASBE [26] is a free, open-source tool-chain for a complete end-to-end design and construction of synthetic biological systems. In general, the TASBE tool-chain divides the whole design problem into tractable sub-problems necessary to lower the level of abstractions.

The TASBE tool-chain connects various synthetic biology design tools, specifically the Proto BioCompiler for specification and compilation [27], MatchMaker for part assignment [28], Puppeteer [29], and BioCAD for physical assembly. The TASBE tool-chain utilizes the Clotho platform and its data model [24, 25] behind the scenes to exchange the data among the involved software tools. Proto BioCompiler is SBOL compliant software tool that is able to import biological data stored in an SBOL file. Due to the Clotho utilization and Clotho's SBOL compliance, the TASBE tool-chain is able to import and export biological data to and from SBOL files. The TASBE research team evaluates the tool-chain on a sensor/actuator test program in two different cellular platforms: mammalian HEK293 cells and *E. coli* bacteria.

5 Discussion

SBOL offers a standard way to represent the design of synthetic biology constructs, and has gained widespread use in both industry and academia. While these results are a great beginning, there are still a number of critical next steps. First, SBOL must be extended as described above to include more of the important information about synthetic biology designs. The incorporation of general components and design modules coupled with experimental context information and models will greatly enhance SBOL's utility and encourage further adoption. To assist in this adoption by more potential developers and interested parties, the community must develop SBOL libraries for frameworks other than Java, such as C/C++ and.NET. In order to help developers improve compliance, the community is developing a suite of SBOL example files that can be used during development, as well as an online web portal to validate SBOL files (http://www.virtualparts.org/sbolvalidator.jsp). Finally, a rising concern in the science and engineering fields is the poor state of reproducibility of published work. Standards, such as SBOL, can help by capturing information critical to enabling reproducibility in an unambiguous and standard format, providing a precise description of a given design and associated requirements such as host and culturing details. Ultimately, we would like to see journals associate synthetic biology articles with the corresponding SBOL details.

Finally, the long-term vision of much of this work is to enable the design of biological constructs using computer aided design. Such designs would be transmitted in SBOL to DNA fabrication centers or even local desktop DNA synthesis machines and returned to the designer for testing. This ability would enable much more rapid prototyping and testing, and thus, accelerating progress in synthetic biology.

Acknowledgments

We would like to thank the members of the SBOL Developers Group for their work in making the SBOL standard a reality. The SBOL project also acknowledges the generous support of a "Computational Challenges in Synthetic Biology" Award from Microsoft. In addition we wish to also thank the National Science Foundation under Grant Number CCF-1218095, the UK Engineering and Physical Sciences Research Council, the Agilent Technologies Applications and Core Technology University Research (ACT-UR) program, AutoCAD, the CIDAR Group at Boston University, IWBDA, Newcastle University Center for Synthetic Biology, and Raytheon BBN Technologies for their generous support in helping develop SBOL. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

1. Pearson WR, Lipman DJ (1988) Improved tools for biological sequence comparison. Proc Natl Acad Sci 85(8):2444–2448
2. Benson DA, Karsch-Mizrachi I, Lipman DJ, Ostell J, Sayers EW (2010) Genbank. Nucleic Acids Res 38(Suppl 1):D46–D51
3. Boeckmann B, Bairoch A, Apweiler R, Blatter M-C, Estreicher A, Gasteiger E, Martin MJ, Michoud K, O'Donovan C, Phan I et al. (2003) The swiss-prot protein knowledgebase and its supplement TrEMBL in 2003. Nucleic Acids Res 31(1):365–370
4. Berman H, Henrick K, Nakamura H, Markley JL (2007) The worldwide protein data bank (wwpdb): ensuring a single, uniform archive of pdb data. Nucleic Acids Res 35(Suppl. 1): D301–D303
5. Hucka M, Finney A, Sauro HM, Bolouri H, Doyle JC, Kitano H, Arkin AP, Bornstein BJ, Bray D, Cornish-Bowden A et al. (2003) The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. Bioinformatics 19(4):524–531
6. Galdzicki M et al. (2012) Synthetic biology open language (SBOL) version 1.1.0. BBF RFC #87
7. Galdzicki M et al. (2013) SBOL: a community standard for communicating designs in synthetic biology
8. Eilbeck K, Lewis S, Mungall CJ, Yandell M, Stein L, Durbin R, Ashburner M (2005) The sequence ontology: a tool for the unification of genome annotations. Genome Biol 6(R44)
9. Courtot M et al. (2011) Controlled vocabularies and semantics in systems biology. Mol Syst Biol 7(543)
10. Rodriguez C et al. (2009) Bbf rfc 16: Synthetic biology open language visual (sbolv) specification. BBF RFC #16
11. Quinn J et al. (2013) Synthetic biology open language visual (SBOL Visual) version 1.0.0. BBF RFC #93
12. Temme K, Zhao D, Voigt CA (2012) Refactoring the nitrogen fixation gene cluster from *Klebsiella oxytoca*. Proc Natl Acad Sci USA 109:2–7
13. Stevens JT, Myers CJ (2013) Dynamic modeling of cellular populations within ibiosim. ACS Synth Biol 2(5):223–229
14. Cooling MT, Rouilly V, Misirli G, Lawson J, Yu T, Hallinan J, Wipat A (2010) Standard virtual biological parts: a repository of modular modeling components for synthetic biology. Bioinformatics 26(7):925–931
15. Misirli G, Hallinan JS, Yu T, Lawson JR, Wimalaratne SM, Cooling MT, Wipat A (2011) Model annotation for synthetic biology: automating model to nucleotide sequence conversion. Bioinformatics 27(7):973–979
16. Bhatia S, Densmore D (2013) Pigeon: a design visualizer for synthetic biology. ACS Synth Biol 2(6):348–350
17. Ham TS, Dmytriv Z, Plahar H, Chen J, Hillson NJ, Keasling JD (2012) Design, implementation and practice of jbei-ice: an open source biological part registry platform and tools. Nucleic Acids Res 40(18):e141

18. Bilitchenko L, Liu A, Cheung S, Weeding E, Xia B, Leguia M, Anderson JC, Densmore D (2011) Eugene–a domain specific language for specifying and constraining synthetic biological parts, devices, and systems. PLoS ONE 6(4): e18882
19. Madsen C, Myers CJ, Patterson T, Roehner N, Stevens JT, Winstead C (2012) Design and test of genetic circuits using iBioSim. IEEE Des Test Comput 29(3):32–39
20. Roehner N, Myers CJ (2013) A methodology to annotate systems biology markup language models with the synthetic biology open language. ACS Synth Biol. doi:10.1021/sb400066m
21. Notka F, Liss M, Wagner R (2011) Industrial scale gene synthesis. In: Voigt C (ed) Synthetic biology, part B computer aided design and DNA assembly. Methods in enzymology, Chap 11, vol 498. Academic Press, London, pp 247–275
22. Hillson NJ, Rosengarten RD, Keasling JD (2012) j5 dna assembly design automation software. ACS Synth Biol 1(1):14–21
23. Chen J, Densmore D, Ham TS, Keasling JD, Hillson NJ (2012) DeviceEditor visual biological CAD canvas. J Biol Eng 6(1), 1
24. Xia B, Bhatia S, Bubenheim B, Dadgar M, Densmore D, Anderson JC (2011) Developer's and user's guide to clotho v2.0: A software platform for the creation of synthetic biological systems. In: Voigt C (ed) Synthetic biology, part B computer aided design and DNA assembly. Methods in enzymology, Chap 5, vol 498. Academic Press, London, pp 97–135
25. Densmore D, Van Devender A, Johnson M, Sritanyaratana N (2009) A platform-based design environment for synthetic biological systems. In: The Fifth Richard Tapia celebration of diversity in computing conference: intellect, initiatives, insight, and innovations, TAPIA '09. ACM, New York, pp 24–29
26. Beal J, Weiss R, Densmore D, Adler A, Appleton E, Babb J, Bhatia S, Davidsohn N, Haddock T, Loyall J, Schantz R, Vasilev V, Yaman F (2012) An end-to-end workflow for engineering of biological networks from high-level specifications. ACS Synth Biol 1(8):317–331
27. Beal J, Lu T, Weiss R (2011) Automatic compilation from High-Level Biologically-Oriented programming language to genetic regulatory networks. PLoS ONE 6:e22490+
28. Yaman F, Bhatia S, Adler A, Densmore D, Beal J (2012) Automated selection of synthetic biology parts for genetic regulatory networks. ACS Synth Biol 1(8):332–344
29. Vasilev V, Liu C, Haddock T, Bhatia S, Adler A, Yaman F, Beal J, Babb J, Weiss R, Densmore D (2011) A software stack for specification and robotic execution of protocols for synthetic biological engineering. Talk presented at the international workshop on bio-design automation (IWBDA)

Chapter 17

Computational Methods for the Construction, Editing, and Error Correction of DNA Molecules and Their Libraries

Ofir Raz and Tuval Ben Yehezkel

Abstract

The field of synthetic biology is fueled by steady advances in our ability to produce designer genetic material on demand. This relatively new technological capability stems from advancements in DNA construction biochemistry as well as supporting computational technologies such as tools for specifying large DNA libraries, as well as planning and optimizing their actual physical construction. In particular, the design, planning, and construction of user specified, combinatorial DNA libraries are of increasing interest.

Here we present some of the computational tools we have built over the past decade to support the multidisciplinary task of constructing DNA molecules and their libraries. These technologies encompass computational methods for [1] planning and optimizing the construction of DNA molecules and libraries, [2] the utilization of existing natural or synthetic fragments, [3] identification of shared fragments, [4] planning primers and overlaps, [5] minimizing the number of assembly steps required, and (6) correcting erroneous constructs.

Other computational technologies that are important in the overall process of DNA construction, such as [1] computational tools for efficient specification and intuitive visualization of large DNA libraries (which aid in debugging library design pre-construction) and [2] automated liquid handling robotic programming [Linshiz et al., Mol Syst Biol 4:191, 2008; Shabi et al., Syst Synth Biol 4:227–236, 2010], which aid in the construction process itself, have been omitted due to length limitations.

Key words Synthetic biology, DNA editing, Computer-aided design, DNA libraries

1 An Algorithm for the Construction of Single DNA Molecules

Divide and Conquer (D&C) solves a problem (in our case, planning the construction of a long DNA molecule) by dividing it in silico into two smaller subproblems, solving each subproblem recursively using D&C, and combining the solutions to the subproblems into a solution to the original problem. If the problem is small enough (in our case, the DNA molecule is short enough), it is not divided further but is solved directly (in our case, planned as a synthetic oligo). Solving problems with D&C is naturally implemented using recursive procedures.

A fundamental prerequisite of a recursive procedure is that its output is of the same type as its inputs. Our construction procedure [1] is designed so that it accepts two overlapping ssDNA (single-stranded DNA) molecules as input and produces an elongated ssDNA molecule as an output.

The D&C recursive algorithm receives a user-specified target sequence as input and returns as output a list of oligos to be synthesized and a protocol in the form of a robot control program that can be used to construct the desired DNA molecule using the specified set of oligos. The basic recursive subroutine of the algorithm takes as input the sequence of a target molecule and returns as output a construction protocol and its associated cost.

This subroutine divides the target sequence into two overlapping sequences and calls itself recursively with these subtarget sequences as new input. The cost of constructing the target molecule by this protocol is computed by adding the cost of assembling the two overlapping subfragments to the cost of constructing these two individual subfragments. The computed cost accounts for the various features of the construction process, including the number and length of oligos, number of reactions, and the total number of levels in the protocol. The recursive division ends if the subroutine's target is short enough to be synthesized directly as an oligonucleotide. Division points are not chosen so that oligos are of equal length, as usually practiced in polymerase cycling assembly (PCA) methods. Instead, division points are selected to minimize the cost of constructing the target and to respect a set of constraints, including whether good PCR primers exist for each of the subtargets and whether the two subtargets can be elongated together efficiently and specifically in the elongation reaction described in Fig. 1. Validation of specificity and affinity of elongation overlaps and PCR primers is performed using sequence alignment algorithms and Tm (temperature at which 50 % of DNA molecules are annealed to their reverse complement sequence) calculations, respectively. The optimized recursive protocol is then transformed into a robot control program that instructs the robot to construct the molecule bottom-up. It starts with the leaves of the recursive construction tree and iteratively executes the basic chemical step all the way up to the root of the tree until the target molecule is constructed. The hierarchal structure of the resulting construction tree, as also the basis of our error-correction procedure, is described in the following.

The algorithm starts with the target molecule given by the user and searches for an optimal and valid division point. Such a point fulfills a set of constrains such as the existence of specific primers and overlap specificity for the division point in addition to the existence of two valid protocols which may build the two divided parts. Once a valid division point is found, the algorithm recursively searches a protocol to build its two parts. The recursion stops when

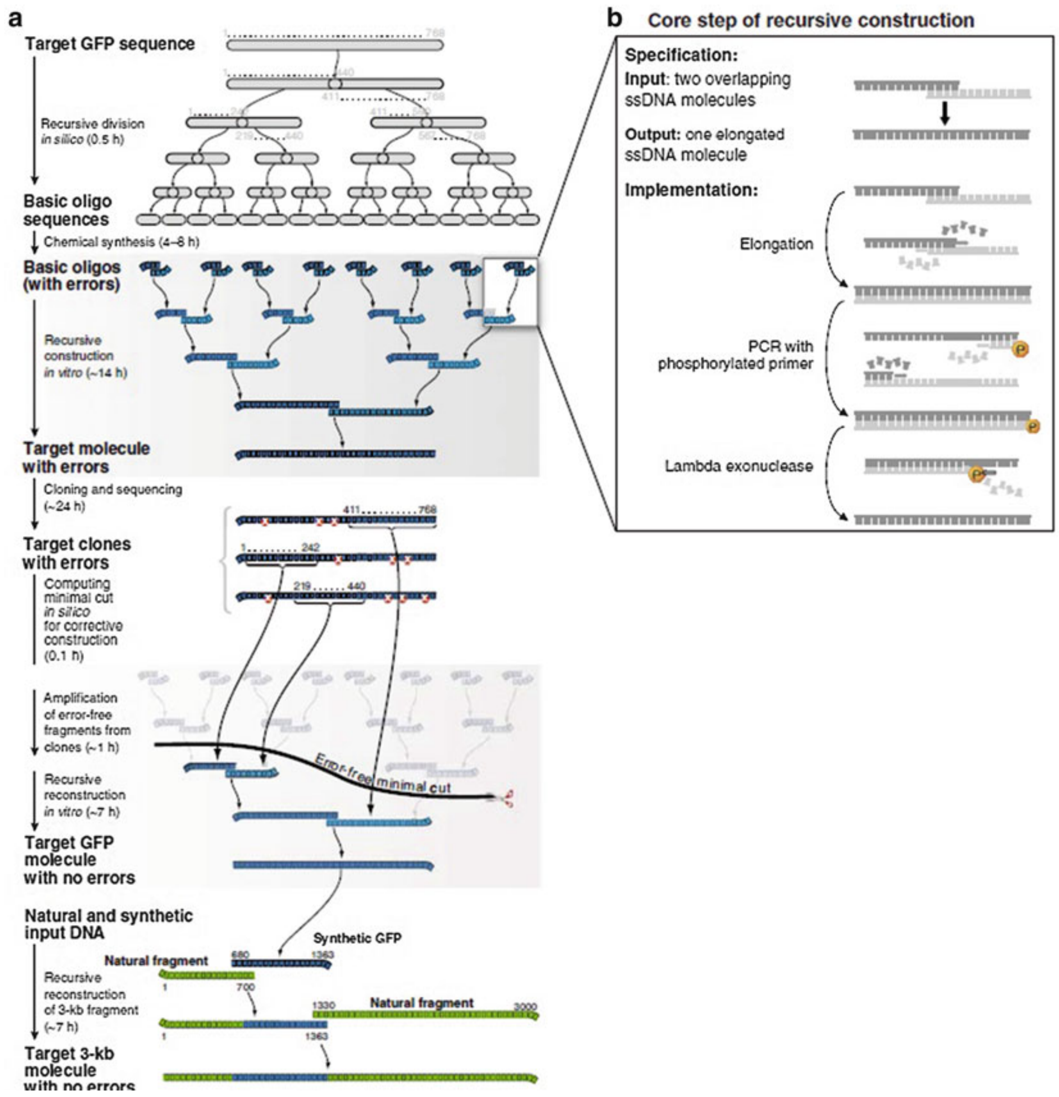

a
Target GFP sequence
Recursive division in silico (0.5 h)
Basic oligo sequences
Chemical synthesis (4–8 h)
Basic oligos (with errors)
Recursive construction in vitro (~14 h)
Target molecule with errors
Cloning and sequencing (~24 h)
Target clones with errors
Computing minimal cut in silico for corrective construction (0.1 h)
Amplification of error-free fragments from clones (~1 h)
Error-free minimal cut
Recursive reconstruction in vitro (~7 h)
Target GFP molecule with no errors
Natural and synthetic input DNA
Natural fragment
Synthetic GFP
Natural fragment
Recursive reconstruction of 3-kb fragment (~7 h)
Target 3-kb molecule with no errors
b
Core step of recursive construction
Specification:
Input: two overlapping ssDNA molecules
Output: one elongated ssDNA molecule
Implementation:
Elongation
PCR with phosphorylated primer
Lambda exonuclease

Fig. 1 Recursive construction of error-free DNA molecules from error-prone oligonucleotides. (**a**) Recursive construction of the Green Fluorescent Protein (GFP) DNA. The Divide and Conquer procedure, as applied to the construction of the 768-nt GFP, is illustrated from *top* to *bottom*. The target sequence is recursively divided in silico into overlapping oligonucleotide sequences (16 oligos of average size of 75 bp). The specified oligos are synthesized by conventional means and serve as inputs (in *blue*) for recursive construction performed in vitro. Construction proceeds by recursively combining pairs of overlapping ssDNA molecules into longer ssDNA molecules, as described in (**b**) until the target molecule is formed. Target molecules typically have the same error rate as their source oligos and are corrected using recursive error correction (*see* Subheading 3). A certain number of target molecules are cloned and sequenced (seven in the case of GFP) and errors (marked in *red*) are identified. Error-free segments found in the erroneous clones are then amplified using PCR and used as inputs to a new recursive reconstruction of the same target molecule. In this case construction started from one half of the target molecule and two quarters of the target molecule that were found to be error free in the sequenced clones. The error-free segments are chosen to contain nodes in the recursive construction tree, so that they can be amplified with the same primers used in the initial procedure. This second iteration of the

the target molecule is short enough to be produced by oligo synthesizer. A cost function is computed for each sub-protocol based on the number of oligos and their lengths, the number of reactions, and the number of protocol tree levels required to build its target. The smallest cost protocol is selected as the optimal protocol. Since the protocol space is very large dynamic programming algorithm is used to keep previously computed sub-protocols in a cache and is reused when needed in a different search path. In addition, branch and bound algorithm is used to trim the search space when the intermediate cost shows that the current best cost for a protocol cannot be improved. Specificity of fragments in elongation reactions and of PCR primers is evaluated using sequence alignment algorithms and Tm formulas from the MatLab bioinformatics toolbox.

2 Processing DNA Molecules as Text

While the electronic representation of text in computers allows composing new text and processing an existing piece of text within the same framework, DNA composition and processing are handled completely separately and using unrelated methods. DNA composition, also called de novo DNA synthesis, uses several methods for assembling synthetic oligonucleotides into ever longer pieces of DNA. DNA processing, the modification of existing fragments, on the other hand, has no systematic solution to date, and the various DNA processing tasks are performed by a plethora of manual labor-intensive methods such as site-directed mutagenesis. Generally, as most of these methods require iterative steps of mutagenesis, cloning, sequencing, and selection, they become inefficient if multiple non-random sequence manipulations are required. Moreover, many of these methods require several steps that are not easily automatable; therefore the time and effort necessary to create libraries of different mutations scale with the size of the library. Alternatively, these methods impose restrictions on the types of changes that are

Fig. 1 (continued) procedure typically (as in this case and all our experiments to date) results in an error-free clone. However, if errors remain, another error-correcting iteration of the procedure can be performed. The figure further demonstrates the construction of a 3-kb DNA fragment by combining, with the same construction procedure, the synthetically produced GFP molecule and DNA from a natural source (bacterial plasmid, in *green*). This yielded an error-free molecule. Expected optimal times for each step using state-of-the-art standard equipment are shown on the *left*. The cloning step could potentially be replaced by single molecule PCR. (**b**) The core step of recursive construction receives two overlapping ssDNA molecules as inputs and produces the elongated ssDNA molecule as output in the following way: the overlapping ssDNA molecules hybridize and prime each other such that an overlap extension elongation reaction returns an dsDNA molecule. This is then amplified by PCR with one of the two primers phosphorylated at its 5' end. The phosphate-labeled PCR strand is then degraded with Lambda exonuclease, yielding an elongated ssDNA molecule as output possible, which limits the scope of their usefulness, e.g., restriction enzymes require specific sites to be present. So far no universal method that overcomes these limitations has been proposed and consequently no engineering discipline that eliminates this manual labor has emerged. A general DNA processing method should enable extensive manipulation of a DNA molecule while maximizing the use of existing DNA and minimizing the need for synthesizing new DNA, similarly to the way a text editor enables efficient editing of an existing text by minimizing the need to retype pieces of text that are already available. A general method should also be amenable to full automation and thus enable the creation of large libraries with a small additional effort.

In this work we present a summary of the computational methods for a uniform framework for DNA processing that encompasses DNA editing, DNA synthesis, and DNA library construction [2]. The framework is based on one core biochemical operation, called Y, that takes as input two DNA fragments, A and B, and produces the concatenated DNA molecule AB (Fig. 2). The input fragments A and B can be two individual DNA molecules or two DNA fragments embedded either in one or two longer DNA molecules, and they can be in single-strand (ssDNA) or double-strand (dsDNA) form. They must, however, be amenable to amplification by a PCR. The output molecule AB of the Y operation is double stranded. This allows the process to be iterated as many times as needed to perform the DNA editing task, as the output of one step is used as the input for the following step. Also, an output of one step can be used as the input for many different processing operations. This property enables the efficient reuse of intermediate DNA fragments. We developed a Divide and Conquer algorithm to find an optimal set of Y operations to produce the target molecules from the input molecules.

The input to the algorithm is the sequence T of the desired target DNA molecule, as well as a set of sequences S of the available input DNA molecules, which could be naturally available or the result of a previous synthesis or processing task. As output, the algorithm produces a DNA processing plan consisting of a set of Y operations. For a single target molecule, the plan has the form of a binary tree of Y operations, where the leaves are either fragments of the input molecules (with valid PCR primers) or synthetic oligos. Internal nodes correspond to intermediate dsDNA molecules built using Y operations and the root is the target molecule T. If there are multiple target molecules, for example a combinatorial variant library, the plan has the form of a directed acyclic graph in which each internal node has two inputs and one or more outputs. A node with multiple outputs represents a DNA molecule that is used as the source of multiple Y operations. The output of the algorithm includes the list of primers and oligos needed to execute the plan as well. Naturally, the plan need not be executed sequentially: all Y

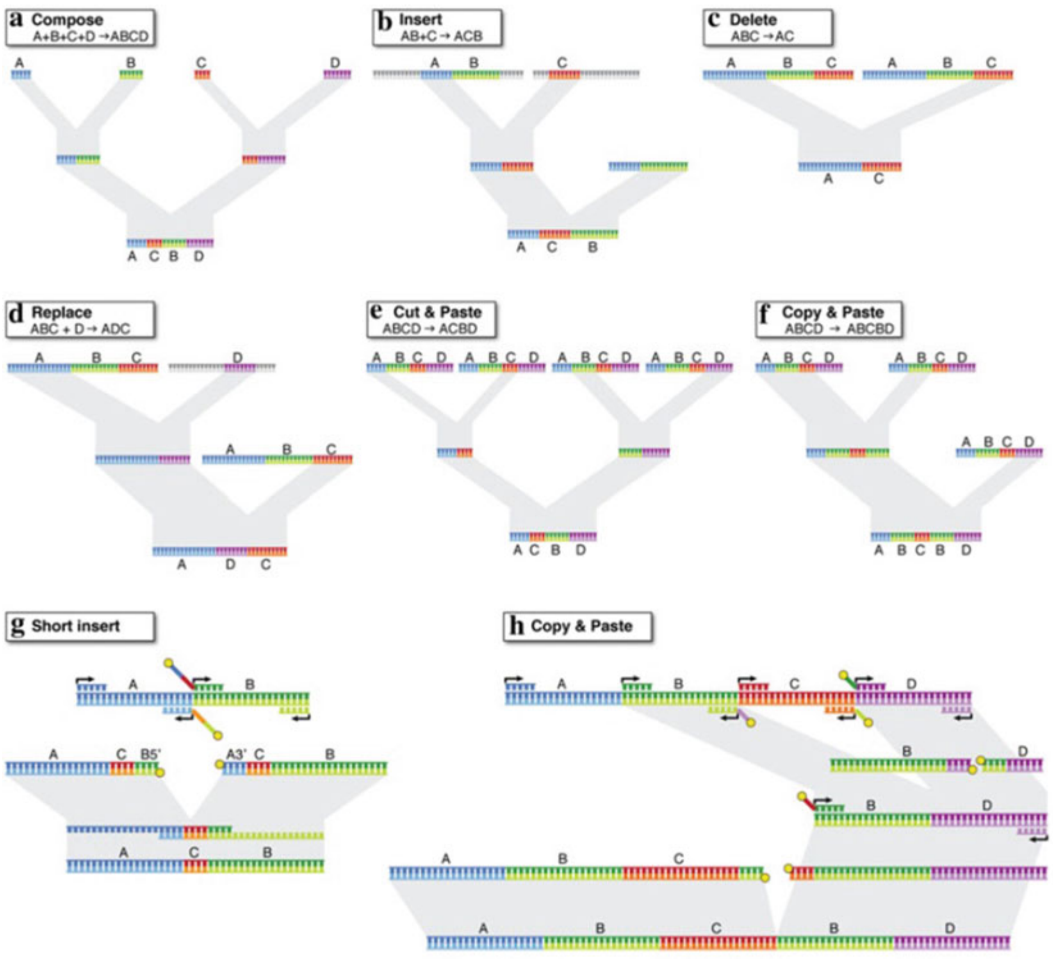


Fig. 2 DNA processing operations. (**a**) Simple composition of DNA fragments done by combining Y operations. In addition, simple DNA edit operations can be performed by composing Y operations. (**b**) Insertion of an existing DNA fragment into another existing DNA fragment. (**c**) Deletion of an internal fragment. (**d**) Replacement of an internal fragment by a new fragment. (**e**) Cut and Paste in which a fragment is deleted and then inserted in another location. (**f**) Copy and Paste, where a fragment B is copied from one location (between A and C) and the copy is inserted in another location (between C and D). (**g**) Short insertions or substitutions can be accomplished simply with a single Y operation, as the modified sequence can be embedded in the overlap. (**h**) Detailed description of the Copy and Paste operations including primers, their overlap extensions, and the required phosphorylation

operations at the same level of the tree or graph can be executed in parallel, so the overall time of executing the plan is typically a function of its depth rather than of its size.

For a target molecule, T, the algorithm computes the DNA processing plan as follows. First we identify in T so-called “input fragments,” which are maximal fragments in T that occur also in one of the input molecules. Clearly any part in T that does not occur in any input molecule has to be synthesized de novo. The algorithm tries to minimize de novo synthesis by maximizing the use of input fragments in composing T.

Next, all end points of the input fragments and all their midpoints in T are marked. At each recursive application of the planning procedure, the marked target sequence is divided into two adjacent parts at a point selected as follows: All potential division points are sorted according to whether or not they occur in an input fragment. Points which fall between input fragments are preferred division points as they do not disrupt the potential use of an input fragment. The points are further sorted according to their absolute distance from the closest middle point of the neighboring input fragments, as this consideration leads to a balanced division and to better concurrency. In this sorting, points that are at the exact ends of an input fragment are preferred over their close neighbors. This allows maximizing the utilization of input fragments by ensuring that their end points are preferred division point.

Once the candidate division points are sorted from best to worst, the first division point is selected and the algorithm tries to plan a basic step reaction that will combine the two sub-fragments induced by the division point into the target molecule. The necessary primers are planned and validated for specificity, affinity (Tm), dimerization, and length constraints for both PCR amplification and elongation reactions of the basic. Should a division point not satisfy any of these constraints, it is removed and the algorithm tries the next potential division point. If a division point satisfies all chemical constraints, the left and right sub-fragments of T, Tl and Tr, are considered new targets and the same algorithm is used recursively to plan their construction. Should none of the division points satisfy the chemical constraints, the algorithm returns a failure. A division point where either the planning of Tl or Tr fails is also excluded. The recursive division ends when the target can be extracted from one of the input fragments or when it is small enough to be produced synthetically. The algorithm produces an efficient DNA processing plan that enables parallel steps on the one hand and makes efficient use of input DNA on the other hand.

3 Correction of Erroneous DNA Clones

The molecules produced in the first iteration of any DNA construction are error prone, reflecting the error rate of the building blocks and the polymerase used. Our recursive construction procedure [1] enables a novel error-correction strategy that employs the very same construction methodology and reagents to produce error-free molecules. Like previous DNA construction protocols, our error-correction procedure uses cloning and sequencing to identify faults, but unlike previous protocols it does not require additional or external methods or reagents to turn the error-prone DNA into error-free DNA.

3.1 General Description of Error Correction

In general, a composite object constructed from potentially faulty basic components is expected to have a higher number of errors than each of its components. However, if errors are randomly distributed among the basic components and occur randomly during construction, and if sufficiently many copies of an object are constructed, it is expected that some of the copies may contain error-free composite parts. If such parts could be identified and extracted from the faulty objects, they could be reused as inputs to recursively reconstruct the object. Since the reconstruction starts from larger parts that are error free, the number of errors in the resulting object is expected to decrease, possibly down to zero. Even if the objects produced in this way have errors, they are expected to have fewer errors than their predecessors and hence to have even larger error-free parts, which can be reused in another iteration of the recursive construction process, until an error-free object is formed (Figs. 1, 3, 4, and 5).

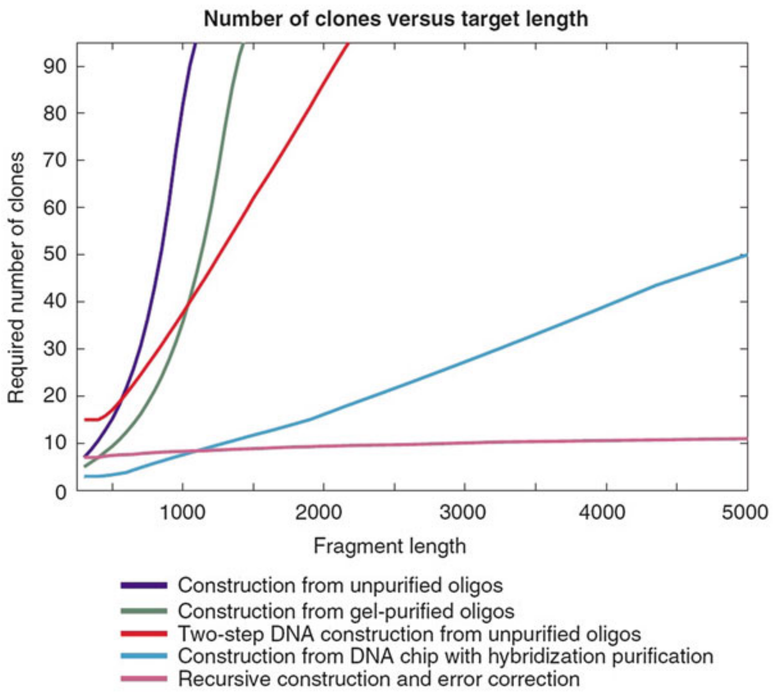


Fig. 3 Comparative analysis of error-correction methodologies—error correction of a single molecule. The required number of clones that have to be sequenced to obtain an error-free synthetic DNA molecule as a function of its length is shown for different methods of construction: naive construction from synthetic oligos with no error correction (*blue*); construction from gel-purified oligos (*green*); a two-step DNA construction, where in the first step molecules of length 500 are constructed, cloned, sequenced, and in the second step these error-free molecules are used as building blocks for larger molecules (*red*); a two-step construction from oligos purified by hybridization (*cyan*); and recursive construction with iterative error correction (*purple*)

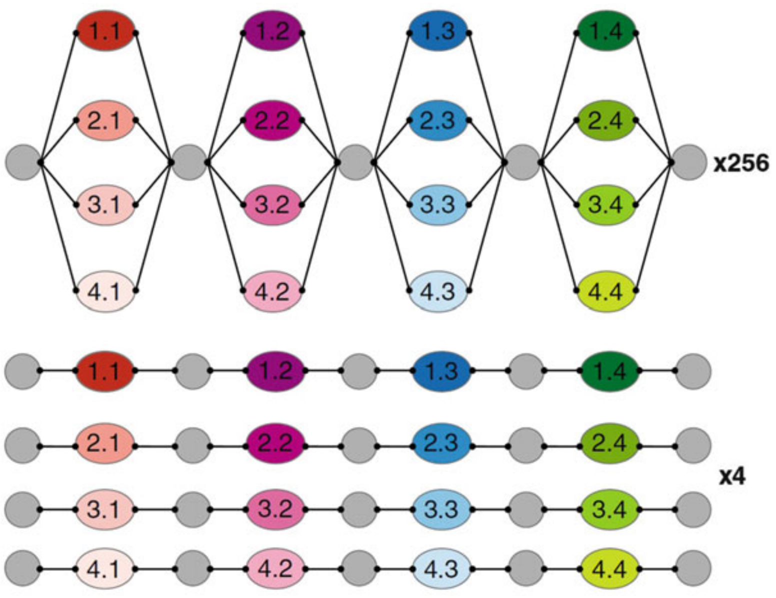


Fig. 4 Error correction of libraries: a graph representing a DNA library with four variable sites, each containing four variants, for a total of 256 possible library members (*top*). Using recursive construction one can first construct and error correct a representative set of only four library members, which constitute a minimal cut through the construction graph of the entire library. A subsequent iteration of the protocol can use error-free fragments obtained from these four library members to efficiently construct the entire 256-strong library. This dramatically economizes the error correction of libraries compared to the correction of each library member separately, as presented in Fig. 3

3.2 Minimal Cut

A cut in a tree is a set of nodes that includes a single node on any path from the root to a leaf. Let T be a recursive construction (RC) protocol tree and S a set of strings. We say that S covers T if there is a set of strings C such that every string in C is a substring of some string in S and C is a cut of T. In such a case we also say that S covers T with C.

Claim: If S covers T, then there is a unique minimal set C such that S covers T with C. Proof: Easy.

Error-free reconstruction algorithm: Given an RC protocol T and a set of sequences (of molecular clones) S, find a minimal C such that S covers T with C. Then we amplify C with PCR and do the recursive construction starting with C.

3.3 Computing the Minimal Cut

We use a recursive approach for computing the minimal cut of a protocol tree. Each node in the tree represents a biochemical process with a product and two precursors. The algorithm starts with the root of the tree (target molecule) and for each node checks whether its product sequence exists with no errors in one of the clones. If such a clone exists this product is marked as a new basic

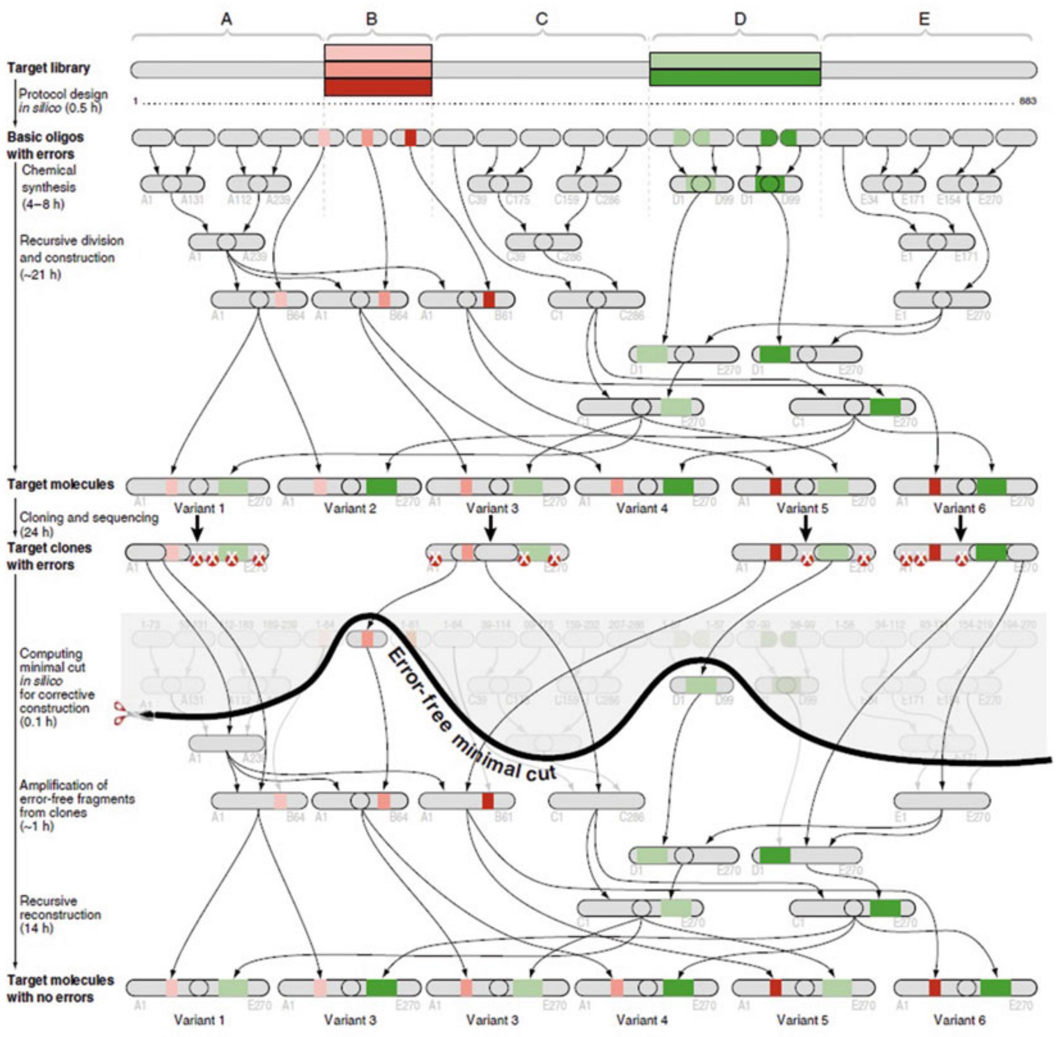

A
B
C
D
E
Target library
Protocol design in silico (0.5 h)
Basic oligos with errors
Chemical synthesis (4–8 h)
Recursive division and construction (~21 h)
Target molecules
Variant 1
Variant 2
Variant 3
Variant 4
Variant 5
Variant 6
Cloning and sequencing (24 h)
Target clones with errors
Computing minimal cut in silico for corrective construction (0.1 h)
Error-free minimal cut
Amplification of error-free fragments from clones (~1 h)
Recursive reconstruction (14 h)
Target molecules with no errors

Fig. 5 Recursive construction and error correction of a simple combinatorial library. The recursive construction of six p53 variants is illustrated *top* to *bottom*. A diagram describes the shared (A, C, E in *gray*) and unique (B, D, *colored*) components of the target p53 combinatorial DNA library. A library construction protocol is computed, where target library sequences are recursively divided into shared and unique components and then further divided into basic oligonucleotide sequences that are then synthesized conventionally. *Gray* oligos are shared by all library variants, and *colored* segments are used by variants with the corresponding colors. Oligos are recursively combined in vitro to form the six target p53 variants. These variants were cloned and sequenced, and errors were identified (marked in *red* on top of clones). An error-free minimal cut (the non-faded part of the graph below the minimal cut *black line*) of the library construction graph was computed from only four error-prone clones (variants 1, 3, 5, and 6). Error-free segments out of these clones were used as inputs for another iteration of the recursive reconstruction protocol, this time producing error-free clones of all six target library members. Expected execution times for each step using standard equipment are shown on the *left*

building block for reconstruction of the target molecule and its primer pair and relevant clone (as template) are registered as its generating PCR. If there is no clone that contains an error-free sequence of the node product, the reaction is registered as existing reaction in the new protocol and the algorithm is recursively executed on the two precursors of the product. The output of such a protocol is a tree of reactions that comprises a minimal cut of the original tree. It contains leaves for which error-free products exist and that all its internal nodes are error free in the clones that contain them. An automated program that utilizes these new error-free building blocks for recursive construction of the target molecule is generated for the robot.

3.4 Computing the Required Number of Clones

For a fragment of size L under mutation rate R the probability of having an error-free fragment in a single clone is taken from a Poisson distribution with $\lambda = L \times R$ (the probability to have 0 errors when the average expected number of errors is $L \times R$). For example, a 1,000 bp fragment (L) with an average error rate of 1/200 (R) would have a λ value of 5 ($L \times R$). Consequently, following a Poisson distribution the probability of obtaining a clone with exactly 0 errors in this case is 0.7 %, while the probability of obtaining a clone with more than 0 errors is 99.3 %. To find the smallest number of clones required to get an error-free fragment with probability larger than 95 % we use a binomial distribution and compute the probability of having at least one error-free fragment out of N clones.

In the Divide and Conquer approach the length of the pure fragment can be reduced to the size of an oligo (~80 bp) at the expense of having to perform more steps during reconstruction. Thus, to guarantee error-free coverage of the target molecule, the probability of having an error-free fragment of size L in N clones (P Success(L,N)) is multiplied by the number of fragments of size L that are required to construct the target molecule (since multiple L-sized fragments are required to reconstruct the target molecule). We compute this number after considering the overlap, which reduces the contribution of each oligo to be smaller than its actual size (~55 bp). Then, we find the smallest number of clones which satisfies the requirement that the total probability of having a minimal cut will exceed 95 %.

4 An Algorithm for DNA Library Construction

Both combinatorial DNA assembly and de novo DNA synthesis can be improved considerably (e.g., made cheaper, ameliorate error rates) by maximizing DNA reuse—identifying shared parts and utilizing them during construction. Most DNA generally needed in biological and biomedical research, and in synthetic and systems biology in particular, are extensions, variations, and combinations of existing DNA. This provides ample opportunities for DNA reuse. De novo synthesis relies on synthetic oligos that are inherently error prone, and therefore reusing existing error-free DNA in constructing new DNA provides an inherent advantage. Moreover, even in applications that require completely new DNA, for example due to change of codon usage, many molecules are often needed, typically a library of variants of a given molecule. The production of these variants, once an error-free prototypical instance of the DNA molecule is isolated or synthesized, includes opportunities for DNA reuse almost by definition.

4.1 High-Level Description of the Algorithm

Combinatorial DNA libraries (graphically represented in Fig. 6) can be assembled algorithmically in an efficient manner from pre-existing and newly synthesized biological parts. This is achieved by recognizing that consecutive combinations of input parts appear multiple times within the output molecules (targets). Ordering the necessary assembly steps into stages, in which a judicious selection of intermediates is assembled earlier, facilitates reuse of assembled parts, leading to an overall reduction in the number of biochemical operations required to assemble the library.

A DNA library assembly task includes a set of input molecules, a set of output molecules (targets), and assumes a binary part concatenation operation that can select two parts within existing (input or intermediate) molecules to produce their concatenation. An assembly plan that produces the outputs from the inputs can be viewed as a graph representing the binary part concatenation operations leading to the set of targets from the set of inputs through a number of intermediates. Figure 7 visualizes several possible plans for constructing a simple library composed of four targets. The quality of a plan can be categorized by the pair of integers (*stages*, *steps*), which allows comparing the outputs of assembly-planning algorithms.

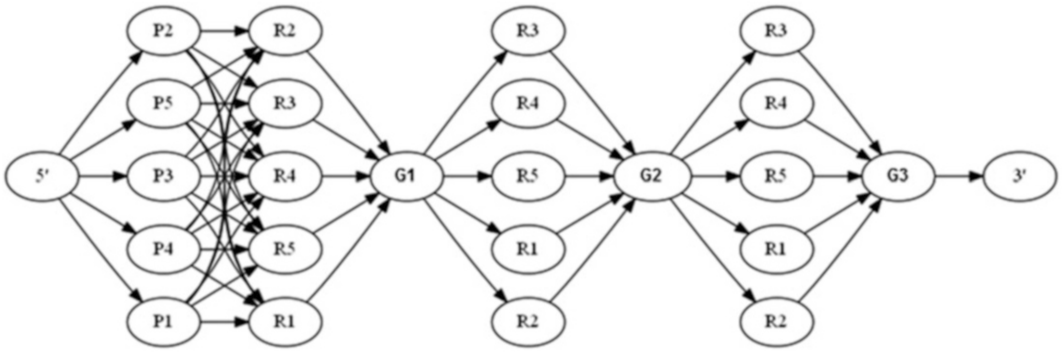


Fig. 6 Schematic DNA library of 625 targets where stacked parts represent combinatorial regions. Every path of the graph represents a target. Note that many parts are shared between several targets, some are shared throughout the library

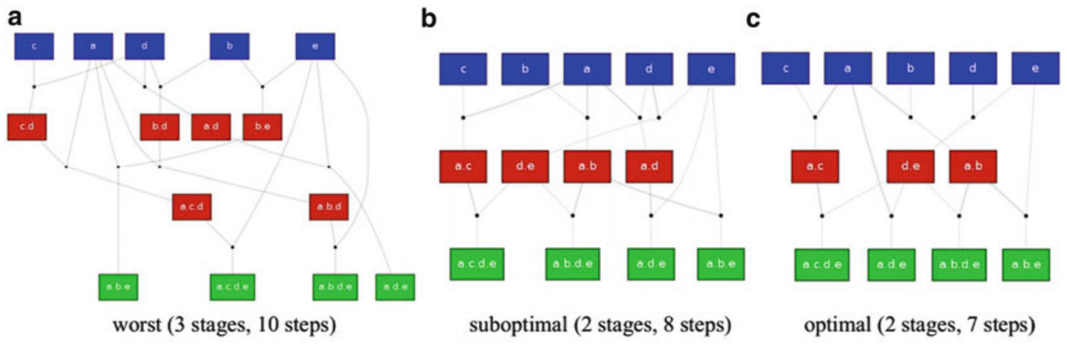


Fig. 7 Graphs of three alternative valid assembly plans for a set of four sequences {acde, abde, ade, abe}. Primitive parts are shown in *blue*, intermediate composites in *red*, and targets in *green*. From the top downwards, each step is represented by a *black dot* concatenating two parts from an earlier stage into a new intermediate or target part. The set of vertically aligned steps forms a stage. The *thicker line* from a part to a *dot/step* indicates the left component of the concatenation

A plan is considered to be better if it has fewer stages and fewer steps. There may be a trade-off between stages and steps. However, whether a suboptimal number of stages with significantly fewer steps is preferable or not is a technology-dependent manufacturing decision.

Here we present the computational methods for an algorithm that directly addresses whole library assembly by operating, at each stage, on all potential binary concatenations [3]. The algorithm assembles all targets simultaneously from their individual parts, akin to the visualization of an assembly graph (as can be seen in Fig. 7). The central idea is recognizing that maximal reuse of an intermediate part can result in fewer biochemical steps (as illustrated by the examples B and C in Fig. 7). This is achieved by greedily concatenating the most frequently occurring pairs of adjacent parts, in the hope of producing only those members of the smallest intermediate parts set. It is well known that greedy algorithms, applying locally optimal choices at each stage, can quickly obtain solutions that approximate the global optimum, even for some hard problems.

Starting from a data structure where each library target is decomposed into a sequence of available unitary parts, the algorithm repeatedly—in analogy to independent steps within an assembly stage—concatenates a subset of the remaining pairs of adjacent parts within the sequences until no pairs remain, meaning every target has been successfully reassembled. The sets of pairs concatenated at each stage of the execution constitute a viable assembly plan.

Three heuristics operate to influence the choice of assembly steps at each stage:

1. Each class of pairs is treated as being mutually exclusive to another class of pairs if at least one instance of each overlaps: share of a right or left part respectively such as B in the pairs AB

and BC, for the trio ABC. This makes it possible to safely concatenate all instances of any given pair, because instances of overlapping pairs will be prevented from also being assembled in the stage. For greedy efficiency, all non-excluded pairs are concatenated.

2. For efficacy, the choice of which pairs to concatenate is determined by the relative number of times each pair occurs within the nascent library data structure. Pairs are sorted by count (which establishes a ranking) but are then processed individually and either chosen as a step or excluded. The first few pairs to be processed are much more likely to be chosen because they are less likely to have been excluded by a previous pair. Those with lower or equal frequencies, assessed later, are likely to have already been excluded and cannot therefore be chosen at that stage.
3. To enable the broadest exploration of the space of assembly plans obtainable under the previous heuristics, we introduce an element of non-determinism to every stage. A random shuffle followed by a stable sort serves as a tiebreaker—reordering pairs with the same count while retaining the ranking, such that the most frequently occurring pairs are still selected.

This combats the potential for the introduction of bias towards the first encountered pairs, when breaking ties between pairs.

An explanatory example is assembling the single target $\{ABCABC\}$ given $\{A, B, C\}$. The algorithm will first produce either AB or BC, then always ABC and finally $ABCABC$, in the minimum three stages. Figure 8 gives another worked example, yielding the solution in Fig. 7b.

4.2 Formalized Problem and Previous Work

More formally, we will refer to strings over a given fixed finite alphabet (i.e., of nucleotides). The part concatenation operation applied to a set of strings $\boldsymbol{X}$ consists of selecting two strings A and B from $\boldsymbol{X}$, selecting a part (substring) A' of A and a part B' of B, and adding the string A' followed by B', denoted $A'B'$, back to $\boldsymbol{X}$.

An ***S-T** library assembly problem* is to produce the set of target strings $\boldsymbol{T}$ using a finite number of applications of the part concatenation operation on $\boldsymbol{S}$ (the set of initial sequences).

We note that the part concatenation operation is realizable in most cases using standard biochemistry. Part selection can be achieved via PCR amplification. Concatenation can be realized using various binary assembly protocols. The cost of a solution is defined by the number of concatenation operations. The depth of a solution is defined as the minimum number of parallel stages needed, where a parallel stage is a set of concatenation operations that can be performed in parallel since they do not depend on each other. Equivalently, the depth of a solution is the maximum number of concatenation steps on a directed path from a source string to

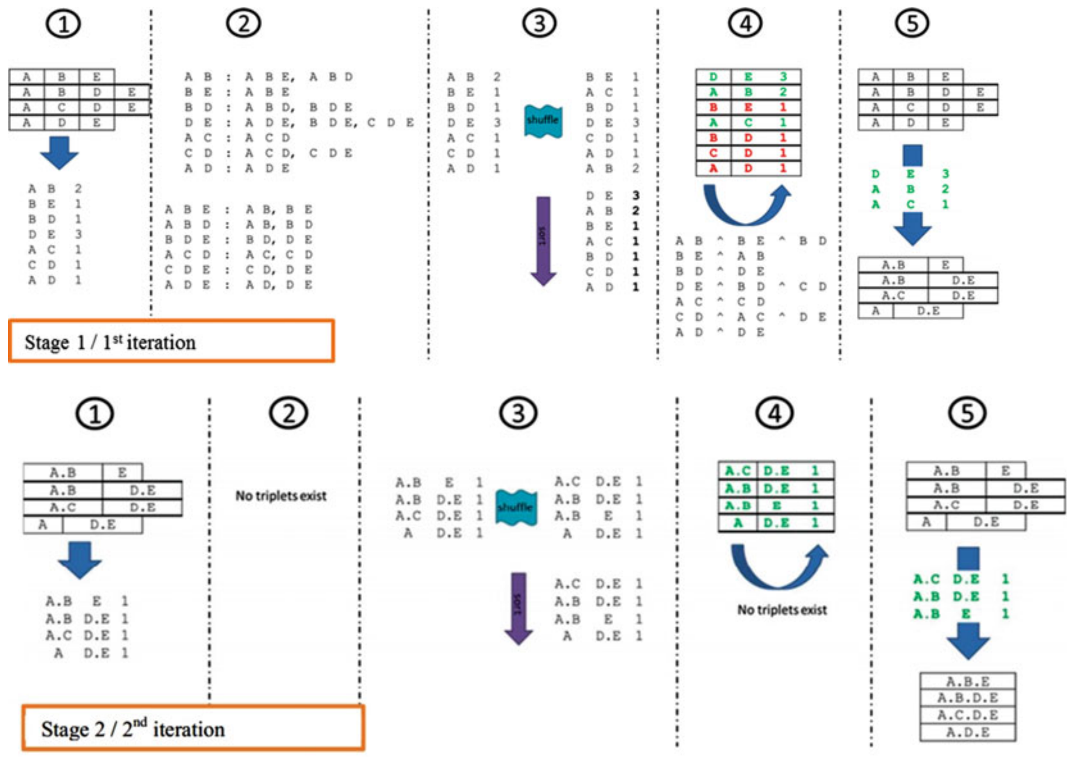


Fig. 8 Step-by-step walkthrough of a run with the example library {ABE, ABDE, ACDE, ADE}. While targets remain to be assembled, each iteration progresses in five phases: (*1*) occurrences of every pair are counted; (*2*) pairs are mapped to containing triplets and vice versa; (*3*) a list of pair-count tuples is shuffled (*blue wave*) and stably sorted by count descending (*purple arrow*); (*4*) for each yet-to-be-excluded pair, choose the pair to be a step (*green rows*) then exclude other pairs (*red rows*), which share the same containing triplets, by using mappings from the second phase (these mutually exclusive pairs are marked with ^); finally (*5*) concatenate all instances of each step-pair. Note that in this example, no triplets exist in the second iteration, as the structure of the library after the first iteration is exactly the set of pairs that will comprise the steps of this final stage

a target string. We also consider the problem of finding a solution of depth at most d and minimum cost. We call this problem *bounded-depth min-cost string production* (BDMSP).

When assembly steps are restricted to binary concatenations, the minimum number of stages can be calculated as *round up* $log_2(n_{\max})$. Determining the minimum number of steps is more difficult. It *could* be determined by exhaustively enumerating all possible assembly plans. However, even for modestly sized libraries this is an intractable task. Even for just a single target, the number of possible assembly graphs is the Catalan number, $(2p - 1)!/(p - 1)!p!$, that grows very rapidly with the target length in elementary parts (p), e.g., 1, 12, 360, 20160, 1814400. Furthermore, the bounded multistage DNA library assembly problem is either NP-hard or APX-hard [3], indicating that a superpolynomial number of operations is required to determine the minimum number of steps for minimal stages or even just to approximate it up to a constant error. In these circumstances, robust (fast and scalable) assembly algorithms, which do not sacrifice solution quality for performance, are required.

The previous state-of-the-art heuristic for BDMSP is the multiple goal-part assembly algorithm [4] that iteratively assembles individual targets by using the minimum number of stages, and produces a high-quality library assembly graphs by using three techniques:

1. Making accumulated subgraphs available to subsequent targets at no cost.
2. BIASING subgraph choice based on precomputed counts of all possible intermediates.
3. Permitting the depth of any target tree to grow up to the global minimum depth ("slack").

This algorithm has a runtime complexity $O(k^2 {n_{\max}}^3)$, where k is the number of targets and $n_{\max}$ the number of component parts in the largest target. The algorithm presented earlier in Subheading 4.1 has a runtime complexity $O(kn_{\max})$ and offered significant reduction in computation time, allowing for both instantaneous assembly planning as well as incorporation into an immediate feedback of construction costs to the library designer [3].

4.3 Problem Solved?

In our opinion, CAD software for the combinatorial design of DNA-based devices' libraries still lack an *accurate* measure of the optimal number of biochemical operations required to assemble any given library. An integrated decision-support mechanism leveraging this measure could guide the researcher in making the most effective use of limited funding and analytical resources. Moreover, knowing the value of the near optimal *stages* and *steps*, objective functions of solution quality would be an effective means by which to benchmark and differentiate algorithms that address this problem.

References

1. Linshiz G, Ben Yehezkel T et al (2008) Recursive construction of perfect DNA molecules from imperfect oligonucleotides. Mol Syst Biol 4:191
2. Shabi U, Kaplan S et al (2010) Processing DNA molecules as text. Syst Synth Biol 4:227–236
3. Blakes J, Raz O et al (2014) A heuristic for maximizing DNA reuse in synthetic DNA library assembly. ACS Synth Biol. doi:10.1021/sb400161v
4. Densmore D, Hsiau THC, Kittleson JT, DeLoache W, Batten C, Anderson JC (2010) Algorithms for automated DNA assembly. Nucleic Acids Res 38:2607–2616

Index