DNA Implementation of Theorem Proving with Resolution Refutation in Propositional Logic

In-Hee Lee[1], Ji-Yoon Park[2], Hae-Man Jang[2]
Young-Gyu Chai[2], and Byoung-Tak Zhang[1]

[1] Biointelligence Laboratory
School of Computer Science and Engineering
Seoul National University, Seoul 151-742, Korea

[2] Department of Biochemistry and Molecular Biology
Hanyang University, Ansan, Kyongki-do 425-791, Korea
{ihlee,jypark,hmjang,ygchai,btzhang}@bi.snu.ac.kr

Abstract. Theorem proving is a classical AI problem having a broad range of applications. Since its complexity grows exponentially with the size of the problem, many researchers have proposed methods to parallelize the theorem proving process. In this paper, we use the massive parallelism of molecular reactions to implement parallel theorem provers. In particular, we show that the resolution refutation proof procedure can be naturally and efficiently implemented by DNA hybridization. Novel DNA encoding schemes, *i.e.* linear encoding and hairpin encoding, are presented and their effectiveness is verified by bio-lab experiments.

1 Introduction

DNA computing is famous for its power of massive parallelism. Since Adleman's first experiment [1], many researchers utilized parallel reactions of DNA molecules to solve hard computational problems [3, 7, 17]. Recently, several research groups have proposed DNA computing methods for logical reasoning [6, 10, 16, 15].

Theorem proving is a method for logical reasoning and has a variety of applications, including diagnosis and decision making [8, 11]. Resolution refutation is a general technique to prove a theorem given a set of axioms and rules. But theorem proving by resolution refutation has a difficulty in practice. If the goal becomes complex or the number of axioms gets large, the time for theorem proving grows exponentially. To overcome this drawback, parallel theorem provers have been proposed [5, 9, 14]. However, these parallel machines do not overcome the difficulties inherent to silicon-based technology.

Wasiewicz *et al.* [16] describe an inference system using molecular computing. Their inference system is different from ours in that theirs does not use a resolution refutation technique. A resolution method for Horn clause computation was suggested in [6, 15], but was not used for theorem proving. Mihalache proposed an implementation of Prolog interpreter with DNA molecules which

is an important practical application of resolution refutation theorem proving [10]. But his suggestion is not physically feasible, because it requires too many experimental steps. Except for the inference system in [16], none of these was implemented in real bio-lab experiments.

In this paper, we describe resolution refutation theorem proving methods using DNA, which is verified by experiments. We develop two different encoding methods for logical formulas, *i.e.* linear and hairpin encodings. These make use of the DNA hybridization reaction in a natural way to perform resolution refutation. Our implementation requires only a constant number of lab steps. The feasibility of the methods is confirmed by lab experiments.

The rest of the paper is organized as follows. A brief introduction to theorem proving and resolution refutation is given in Section 2. Sections 3 and 4 describe the linear and hairpin implementation methods and their experimental results, respectively. Conclusions are drawn in Section 5.

2 Theorem Proving with Resolution Refutation

Theorem proving is a method for automated reasoning [8, 11]. In theorem proving, one must decide methods for representing information and inference rules for drawing conclusions [8]. Here, we confine ourselves to propositional logic. We use the resolution as inference rule. In this section, we will briefly describe propositional logic, resolution principle, and resolution refutation.

Propositional logic formula consists of Boolean variables and logical connectives. Boolean variable is a variable which can take only $\mathcal{T}$ (*true*) or $\mathcal{F}$ (*false*) as its value. Among basic logical connectives, there are $\wedge$ (*and*, logical product), $\vee$ (*or*, logical sum), $\neg$ (*not*, negation), and $\rightarrow$ (*implication*). A Boolean variable or its negation is called a *literal*. More exactly, a Boolean variable with $\neg$ connective is called a negative literal and one without it is called a positive literal. It is proven [2] that any n-ary connective can be defined using only $\wedge$, $\vee$, and $\neg$. For example, $A \rightarrow B$ can be defined as $\neg A \vee B$ for any Boolean variables A, B.

To prove theorems using resolution refutation, every formula must be expressed in clause form. A clause form in propositional logic is defined as follows:

$$(clauseform) := (clause) \wedge (clause) \wedge \cdots \wedge (clause)$$
$$(clause) := (literal) \vee (literal) \vee \cdots \vee (literal)$$

A clause with no literal is called an empty clause.

Now the resolution principle can be defined. Let $\mathcal{A}$ and $\mathcal{B}$ be clauses, and v be a literal such that $v \in \mathcal{A}$ and $\neg v \in \mathcal{B}$. Then, from both $\mathcal{A}$ and $\mathcal{B}$ we can draw $(\mathcal{A} - \{v\}) \vee (\mathcal{B} - \{\neg v\})$. We say that we resolved $\mathcal{A}$ and $\mathcal{B}$ on v and the product of resolution is called a *resolvent*.

In general, a resolution refutation for proving an arbitrary formula ω from a set of formulas Δ proceeds as follows [11]:

1. Put the formulas in Δ into the clause form.
2. Add the negation of ω, in clause form, to the set of clauses and call it $\mathcal{C}$.

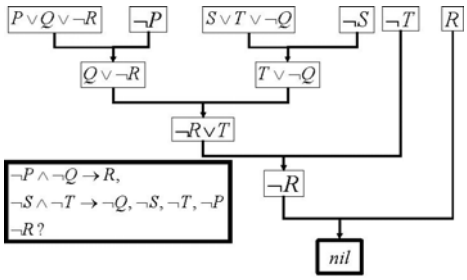

Fig. 1. Theorem proving using resolution refutation.

3. Resolve these clauses together, producing a resolvent that logically follows from them.
4. If an empty clause is produced, a contradiction is occurred. Thus, ω is proved to be true. Stop.
5. If no new resolvent can be produced, ω is proved to be false. Stop.
6. Else, add the resolvent to $\mathcal{C}$ and go to step 3.

When an empty clause is drawn, $\mathcal{C}$ is called the proof of the goal ω. Fig. 1 shows an example of theorem proving process. The set of formulas $\{ P \wedge Q \rightarrow R, S \wedge T \rightarrow Q, S, T, P \}$ is given. We want to prove R is true. After converting the formulas into clause form, we get a set of clauses $\{\neg P \vee \neg Q \vee R, \neg S \vee \neg T \vee Q, S, T, P\}$. Then we add the negation of R, *i.e.* $\neg R$, to this set. Each box in Fig. 1 contains one clause. The clauses and their resolvents are connected with arrows. The symbol *nil* denotes an empty clause. Resolving on P from $\neg P \vee \neg Q \vee R$ and P results in $\neg Q \vee R$. Similar steps can be continued until an empty clause is produced. The above theorem proving process becomes complex as the number of formulas grows. In the case of the propositional calculus, one must decide the literal to resolve on. Therefore, if there are n different literals, there are $n!$ different theorem proving processes with $n! = O(n^n)$ by Stirling's approximation. Thus, the complexity of proof grows exponentially. So it is impossible for large n to test one by one with digital computer which one is a logically correct proof. To speed up finding proofs, two approaches were developed. One approach is to use heuristics such as the breadth-first strategy, the set of support strategy, or the unit preference strategy [8]. The other approach is to parallelize the theorem proving process [5, 9, 14]. We took the second approach and chose to use the massive parallelism of DNA molecular reactions for implementing parallel theorem provers.

3 Linear Implementation of Resolution Refutation

We solved the theorem proving problem in Fig. 1 by bio-lab experiments. We developed two versions of implementation and performed experiments separately. Two steps are involved in each implementation.

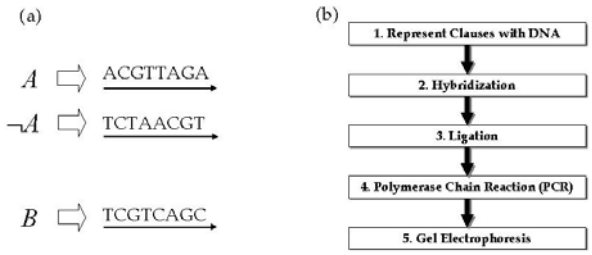

Fig. 2. (a) Encoding for a Boolean variable and its negation (The arrows are from 5' to 3'). (b) The general procedure for our experiments.

In the first step, we represent formulas in clause form with DNA. Because we restrict ourselves to propositional logic, we need to encode each Boolean variable with DNA sequence. Our implementations are different from each other in the way to make a clause from these variable sequences. In both implementations below, we encoded each variable with a different sequence of the same length. The negation of a variable is denoted as a Watson-Crick complementary sequence of the variable. Encoding for a Boolean variable and its negation is shown in Fig. 2. In the second step, we implemented resolution refutation steps with molecular reactions. This step depends on the result of the previous step. But in our two implementations, the general procedure is the same (see Fig. 2).

1. Mix molecules corresponding to clauses.
2. Hybridize molecules to perform resolution.
3. Ligase hybridization products to make it easy to find a proof.
4. Perform PCR to amplify ligation products. In this step, ligation products which are not a proof are not amplified.
5. Perform gel electrophoresis to see whether we found a proof or not.

3.1 Representation of Clauses

Each sequence for a variable in the clause is concatenated into a single long sequence. We arranged the variables to make linear double-strand after the hybridization step. As an example, sequence for a clause $\neg Q \vee \neg P \vee R$ is a concatenation of sequences for $\neg Q$, $\neg P$, and R (see the topmost box in Fig. 3).

3.2 Implementation of Resolution Refutation

Resolution between two clauses is represented with the annealing of two sequences corresponding to each clause. Thus, the resolvent may be either a partially double-stranded molecule or a fully double-stranded molecule, if the resolvent is an empty clause. Therefore, to see whether an empty clause is produced,

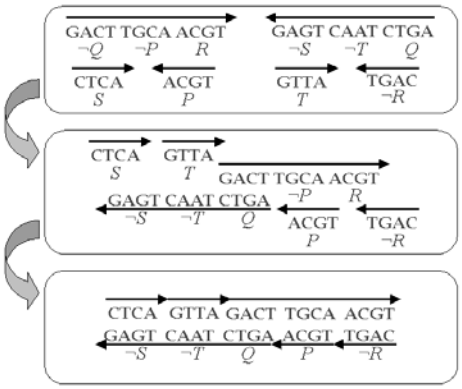

Fig. 3. Simplified process of linear implementation (The arrows are from 5' to 3').

one needs to verify if there exists a fully double-stranded molecule. We used ligation, PCR, and gel electrophoresis to detect this molecule. During ligation, every clause sequence used to produce an empty clause will be ligated into a long double strand. In the next step, we amplified this ligation product by PCR with the goal sequence as a primer. Finally, the result is examined by gel electrophoresis. All of these experimental steps are summarized in Fig. 3.

3.3 Experimental Results

The sequences used in the experiment were designed by NACST using evolutionary optimization [13] and synthesized by Bioneer Co. (Tae-Jeon, Korea). The oligomer sequences are given in Table 1. The experiment consists of the following steps:

1. **Purification of oligomers:** Each oligomer was 5'-phosphorylated and purified by PAGE. Briefly, 1 nmol of each oligomer was mixed and incubated for 1 h at 37°C with 10 U T4 polynucleotide kinase (Life Technologies) in 70 mM Tris-HCl, pH 7.6, buffer containing 10 mM $MgCl_2$, 100 mM KCl, 1

Table 1. Sequences for linear implementation (in order of from 5' to 3').

clause	sequence
$\neg Q \vee \neg P \vee R$	cgtacgtacgctgaa ctgccttgcgttgac tgcgttcattgtatg
$Q \vee \neg T \vee \neg S$	ttcagcgtacgtacg tcaatttgcgtcaat tggtcgctactgctt
S	aagcagtagcgacca
T	attgacgcaaattga
P	gtcaacgcaaggcag
$\neg R$	catacaatgaacgca

Table 2. The PCR condition for linear implementation.

cycle	denaturation (94°C)	annealing (58°C)	polymerization (72°C)
1	4 min	0.5 min	0.5 min
2 ∼ 26	0.5 min	0.5 min	0.5 min
27	0.5 min	0.5 min	10 min

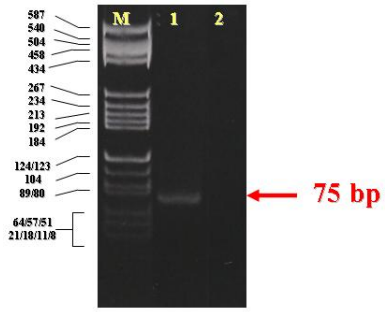

Fig. 4. Electrophoretogram of the PCR product in the linear implementation. Lane 1: PCR products with S and $\neg R$ as primers. Lane 2: PCR products with $\neg S$ and R as primers. Lane M is a size marker.

mM 2-mercaptoethanol and 1 mM ATP (Sigma, St. Louis, MO, USA), in a volume of 100 ul. The T4 kinase was inactivated by heating to 95°C for 10 min.

2. **Hybridization of oligomers:** 100 pmol of each oligomers were mixed. Initial denaturation was achieved at 95°C for 10 min. During hybridization we lowered temperature from 95°C to 16°C (1°C / min) using iCycler thermal cycler (Bio-rad, USA).
3. **Ligation of hybrid molecules:** Ligation was achieved with T4 DNA ligase at 16°C for overnight using iCycler thermal cycler. The reaction buffer contains 50 mM Tris-HCl (pH 7.8), 10 mM $MgCl_2$, 5 mM DTT, 1 mM ATP, and 2.5 μg/ml BSA.
4. **PCR amplification for 'readout':** 50 ul PCR amplification contained 100 pmol of primer and template. The reaction buffer consists of 10mM Tris-HCl, pH 7.8, containing 50 mM KCl, 1.5 mM $MgCl_2$, 0.1% Triton X-100, 0.2 mM dNTP, and 1 U DNA Taq polymerase (Korea Bio-technology, Korea). PCR condition is given in Table 2.
5. **Gel electrophoresis:** Amplified PCR products were purified by electrophoresis on a 15% polyacrylamide gel (30% acrylamide [29:1 acrylamide / bis (acrylamide)]). The running buffer consists of 100 mM Tris-HCl, pH 8.3, 89 mM boric acid, and 2 mM EDTA. The sample buffer is Xylene Cyanol FF tacking dye. Gels are run on a Bio-rad Model Power PAC 3000 electrophoresis unit at 60 W (6V/cm) with constant power.

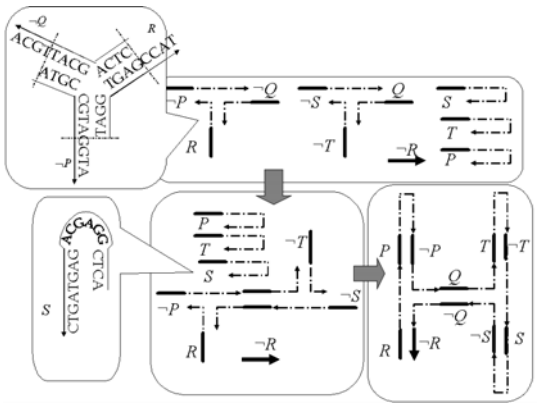

Fig. 5. The simplified process of hairpin implementation (The arrows are from 5' to 3').

The electrophoretogram is given in Fig. 4. To make an empty clause, all of the 5 variables must be resolved. Every time one variable is resolved, double-stranded region with 15 bp is produced. Therefore, we can expect a 75 bp band will appear after gel electrophoresis. As can be seen in Fig. 4, an empty clause is produced. Thus, we found a proof for the given problem.

4 Hairpin Implementation of Resolution Refutation

In linear implementation, variables were needed to be rearranged to make an empty clause form a linear double-stranded molecule. This requirement poses a limitation on the type of clauses we can use. That is, some kind of clauses does not make a linear double strand no matter how we rearrange the variables. Also, it is impossible to know in advance which sequence to use as a primer. To overcome these limitations, we introduce branched molecules and hairpin molecules to represent clauses. As a result of this improvement, we can decide the primer sequence once the goal is given. Similar idea was introduced by Uejima and others[15] for Horn clause computation. But their work was intended to perform Horn clause computation not resolution refutation. And in their work, clause with one literal is represented in a different way from ours.

4.1 Representation of Clauses

Each clause with n literals is denoted by an n-armed branched molecule except for the clause with one literal. Each n-arm has a sticky end corresponding to each literal. Sticky ends for the positive and negative literals of one variable are complementary. For a clause with one literal except the goal, we represent it with a hairpin molecule having a sticky end. We encode the goal clause with

Table 3. Sequences for hairpin implementation (in order of from 5' to 3').

clause	sequence
$\neg Q \vee \neg P \vee R$	gtctgatgcc ggcattacac tcctccagta
	gtgtaatgcc ttccggtacc agcacacgaa
	tcgtgtgctt ggcatcagac accttaccgg
$Q \vee \neg T \vee \neg S$	tacgtgtagc ctgactacaa tactggagga
	ttgtagtcag tgaggaagac gcgttatcac
	tacgtgtagc ctgactacaa tactggagga
S	agttggtctg tcgcaa cagaccaact aatcacaagg
T	gttcttccgt atgcga acggaagaac gtgataacgc
P	ctcttgtctg agcgtt cagacaagag aaggccatgg
$\neg R$	ccggtaaggt

a linear single strand as in the linear implementation described in the previous section. For example, if there were $\neg Q \vee \neg P \vee R$, S, and $\neg R$ (the goal), the clause $\neg Q \vee \neg P \vee R$ is represented by a 3-armed branched molecule, S by a hairpin molecule, and $\neg R$ by a single strand (see Fig. 5).

4.2 Implementation of Resolution Refutation

As in the linear implementation, resolution between two clauses is denoted as hybridization between two molecules. When an empty clause is drawn, it will start with a goal sequence and end with its negation, since clauses are either branched molecules or hairpin molecules except for the goal. Therefore, at the PCR step, we used the goal variable and its negation as a primer. To read the PCR product, we used gel electrophoresis. If a band is formed, we can conclude that the goal is true. If not, we say that the goal is not true. Because each band corresponds to a proof, we can find several different proofs at one time.

4.3 Experimental Results

As in the previous experiment, all sequences were designed by NACST [13] and were synthesized by Bioneer Co. The sequences we used are given in Table 3. Experimental steps are the same as in the previous experiment. The main differences between two experiments were the PCR condition and the type of gel used in gel electrophoresis. The PCR condition for this experiment is given in Table 4. We used 2% agarose gel at the gel electrophoresis step. Due to the properties of our sequences, we could not get an exact result. Another experiment is in progress to find conditions to get a clearer result. The electrophoretograms of hairpin implementation after hybridization and PCR are given in Fig. 6. In Fig. 6-(a), a hazy band appeared around $25 \sim 40$ bp which is the range of sequences we used. In Fig. 6-(b), we can see a partial proof is produced. From this fact, we conclude that some parts of sequences prevented PCR. We are trying to find a clear result with new sequences.

Table 4. The PCR condition for hairpin implementation

cycle	denaturation (95°C)	annealing (65°C)	polymerization (72°C)
1	5 min	1 min	1 min
$2 \sim 26$	1 min	1 min	1 min
27	1 min	1 min	10 min

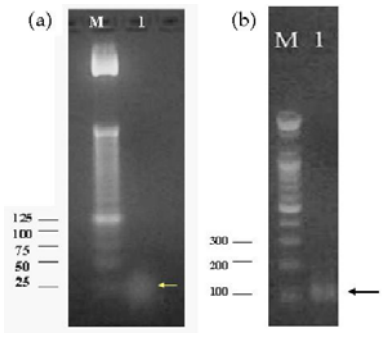

Fig. 6. (a) Electrophoresis of products after hybridization. Lane 1: Hybridization unit of oligomer. Lane M is a 25 bp size marker. (b) Electrophoresis of products after PCR. Lane 1: The PCR product in the presence of primer. PCR amplification of ligation was performed with primer $\neg R$. Lane M is DNA molecular weight marker V (Roche, Germany). The arrow indicates the 110 bp PCR product.

5 Conclusions

Using molecular reactions of DNA, we proved theorems in the propositional calculus. We presented methods for encoding clauses with DNA molecules and solved theorem proving problems with lab experiments. Our methods are distinguished from other work in several points. First, it does not need additional operations except hybridization. Only simple operations such as ligation and PCR are needed to verify the results. Second, the number of experimental steps does not vary with the problem size. Our implementation methods require only hybridization, ligation, PCR, and gel electrophoresis, and these operations are all $O(1)$ operations. Finally, taking the limit of PCR operation (10,000bp) into consideration, our methods can solve theorem proving problems with up to 660 literals (15mer per literal).

There are, however, some cases when our method can not tell whether a proof is found or not. For example, in the hairpin implementation, if the goal variable is resolved more than once, our method will tell there exists a proof regardless of the existence of empty clause. We are trying to develop new methods without this drawback.

Acknowledgements

This research was supported in part by the Ministry of Education under the BK21-IT program and the Ministry of Commerce through the Molecular Evolutionary Computing (MEC) project. The RIACT at Seoul National University provided research facilities for this study.

References

1. Adleman, L., Molecular computation of solutions to combinatorial problems, *Science*, **266**:1021–1024, 1994.
2. Fitting, M., *First-Order Logic and Automated Theorem Proving*, Springer-Verlag New York Inc., 1942.
3. Hagiya, M., Arita, M., Kiga, D., Sakamoto, K., and Yokoyama, S., Towards parallel evaluation and learning of Boolean μ-formulas with molecules, *Preliminary Proceedings of the Third DIMACS Workshop on DNA Based Computers*, 105–114, 1997.
4. Hagiya, M., From molecular computing to molecular programming, *Lecture Notes in Computer Science*, 2001.
5. Hasegawa, R., Parallel theorem-proving system: MGTP, *Proceedings of Fifth Generation Computer System*, 1994.
6. Kobayashi, S., Horn clause computation with DNA molecules, *Journal of Combinatorial Optimization*, **3**:277-299, 1999.
7. Lipton, R.J., DNA solution of hard computational problem, *Science*, **268**:542–545, 1995.
8. Luger, G.F. and Stubblefield, W.A., *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, 2nd Ed., Benjamin/Cummings, 1993.
9. Lusk, E.L. and McCune, W.W., High-performance parallel theorem proving for shared-memory multiprocessors, http://www-fp.mcs.anl.gov/~lusk/papers/roo/paper.html, 1998.
10. Mihalache, V., Prolog approach to DNA computing, *Proceedings of the IEEE International Conference on Evolutionary Computation*, IEEE Press, 249–254, 1997.
11. Nilsson, N.J., *Aritificial Intelligence: A New Systhesis*, Morgan Kaufman Publishers Inc., 1998.
12. Sakamoto, K., Gouzu, H., Komiya, K., Kiga, D., Yokoyama, S., Yokomori, T., and Hagiya, M., Molecular computation by DNA hairpin formation, *Science*, **288**:1223-1226, 2000.
13. Shin, S.-Y., Kim, D., Lee, I.-H., and Zhang, B.-T., Evolutionary sequence generation for reliable DNA computing, *2002 IEEE World Congress on Evolutionary Computation*, 2002 (accepted).
14. Suttner, C., SPTHEO - A parallel theorem prover, *Journal of Automated Reasoning*, **18**(2):253-258, 1997.
15. Uejima, H., Hagiya, M., and Kobayashi, S., Horn clause computation by self-assembly of DNA molecules, *Preliminary Proceedings of the Seventh International Meeting on DNA Based Computers*, 63–71, 2001.
16. Wasiewicz, P., Janczak, T., Mulawka, J.J., and Plucienniczak, A., The inference based on molecular computing, *International Journal of Cybernetics and Systems*, **31**(3):283–315, 2000.
17. Winfree, E., *Algorithmic self-assembly of DNA*, Ph.D. Thesis, California Institute of Technology, 1998.