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. Here, 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 biochemical 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 [3,7,18]. Recently, several research groups have proposed DNA computing methods for logical reasoning [6,10,16,17].

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 [5,9,15]. However, these parallel machines do not overcome the difficulties inherent to silicon-based technology.

Wasiewicz *et al.* [17] 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 [6,16], 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

M. Hagiya and A. Ohuchi (Eds.): DNA8, LNCS 2568, pp. 156–167, 2003.

[10]. But his suggestion is not physically feasible, because it requires too many experimental steps. Except for the inference system in [17], none of these was implemented in real bio-lab experiments.

In this paper, we describe resolution refutation theorem proving methods using DNA, which are 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 $\mathcal{C}$.

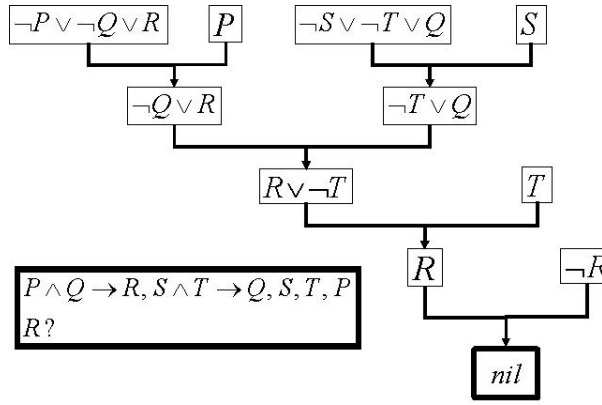

Fig. 1. Theorem proving using resolution refutation.

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, it is proved that ω is consistent with Δ. Stop.
5. If no new resolvent can be produced, ω is proved not to be consistent with Δ. 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 $\Delta = \{P \wedge Q \rightarrow R, S \wedge T \rightarrow Q, S, T, P\}$ is given. We want to prove R is consistent with Δ. 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. 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 symbol *nil* denotes an empty clause.

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 [5,9,15]. We took the second approach and chose to use the massive parallelism of DNA molecular reactions for implementing parallel theorem provers.

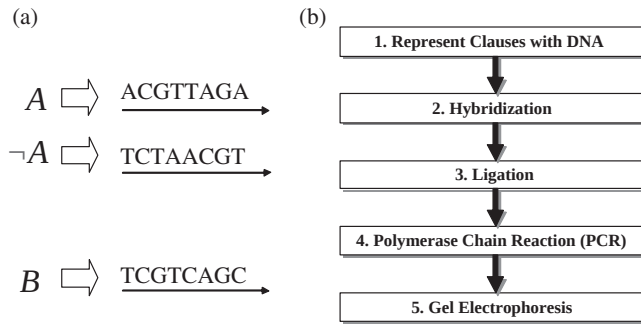

Fig. 2. (a) Encoding for a Boolean variable and its negation (The arrows are from 5' to 3'). (b) The general procedure for our experiments.

3 Linear Implementation of Resolution Refutation

We solved the theorem proving problem shown in Fig. 1 by biochemical experiments. We developed two versions of implementation and performed experiments separately. Each implementation consists of two steps.

In the first step, we represent formulas in clause form with DNA molecules. Because we restrict ourselves to propositional logic, we just need to encode each Boolean variable with different DNA sequences. Our implementations are different from each other in the way to make a clause from these variable sequences. In both implementations, we encoded each variable with different sequences of the same length. The negation of a variable is denoted as a Watson-Crick complementary sequence encoding 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 varies, depending on the representation method used in the previous step. But the general procedure is identical in our two implementations as follows (see Fig. 2-(b)).

1. Mix DNA molecules corresponding to clauses.
2. Hybridize DNA molecules to perform resolution.
3. Ligate hybridization products to make it easy to find a proof.
4. Perform PCR to amplify ligation products. In this step, only the ligation product which is a valid proof can be amplified.
5. Perform gel electrophoresis to see whether we found a proof or not.

3.1 Representation of Clauses

A clause is designed with single-stranded DNA that consists of each variables in the clause. We determined the order in which a literal in a clause appears so that a valid proof make a linear double-stranded molecule after 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).

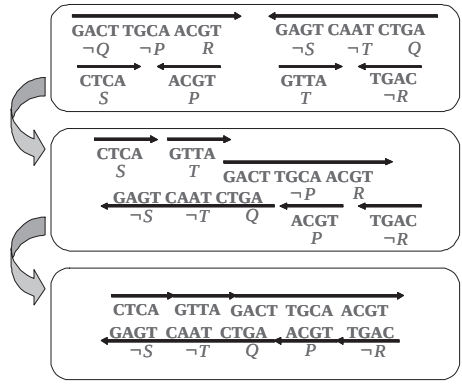

Fig. 3. Simplified process of linear implementation (The arrows are from 5' to 3').

3.2 Implementation of Resolution Refutation

Resolution of a variable between two clauses is represented with hybridization of two regions corresponding to that literal in each clause. For example, when a variable v is resolved from clauses $\mathcal{A}$ and $\mathcal{B}$, regions corresponding to v or $\neg v$ in each clause hybridize and other regions remain unchanged. Therefore, if the resolvent is an empty clause, no region will remain as single-stranded DNA. Therefore, to see whether an empty clause is produced, one needs to verify if there exists a molecule with no single-stranded region. We used ligation, PCR, and gel electrophoresis to find this molecule. During ligation, every clause sequence used to produce the empty clause will be ligated into a long double-stranded molecule. In the next step, we amplify 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 [14] 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 nM 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 mM 2-mercaptoethanol and 1 mM ATP (Sigma, St. Louis, MO, USA), in a volume of 100 μl. The T4 kinase was inactivated by heating to 95°C for 10 min.

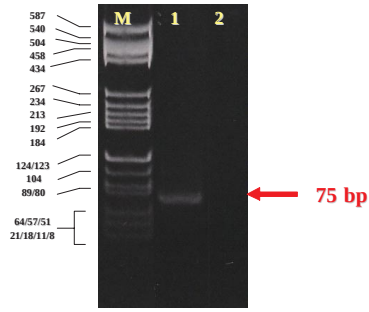

Fig. 4. Electrophoretogram of the PCR product in the linear implementation. Lane 1: PCR products with S and $\neg R$ as primers. Lane 2: PCR products with $\neg S$ and R as primers. Lane M is a size marker.

2. **Hybridization of oligomers:** 100 pM of each oligomer was 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 μl PCR amplification contained 100 pM 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 were run on a Bio-rad Model Power PAC 3000 electrophoresis unit at 60 W (6V/cm) with constant power.

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
$2 \sim 26$	0.5 min	0.5 min	0.5 min
27	0.5 min	0.5 min	10 min

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.

3.4 Discussion

There are some points to be considered to expand this implementation. First of all, we should rearrange variables in each clauses 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-stranded molecule no matter how we rearrange the variables. Also, it is impossible to know in advance which sequence to use as a primer. And there are possibilities of false positive. Even if S and T does not exist in Fig. 3, our implementation will make band with expected length. But we can reduce the possibilities of false positive by including exonuclease treatment step before PCR step.

4 Hairpin Implementation of Resolution Refutation

As mentioned in previous section, there are several limitations in linear implementation. To overcome these limitations, we introduce branched molecules and hairpin molecules to represent clauses. In this implementation, we can always use the negation of goal as a primer. Similar idea was introduced by Uejima and others[16] for Horn clause computation. But their work was intended to perform Horn clause computation not resolution refutation. Also in their work, molecular form of the empty clause is different from ours.

4.1 Representation of Clauses

Each clause with n literals is denoted by a branched molecule with n arms 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 which is not the goal, we represent it with a hairpin molecule having a sticky end. We encode the goal clause with a linear single-stranded molecule as in the linear implementation described in the

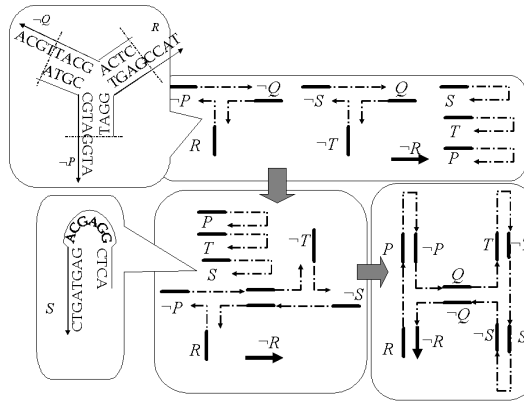

Fig. 5. The simplified process of hairpin implementation (The arrows are from 5' to 3').

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

clause	sequence
P	TATTAAGACTTCTTGTAGTCT
$\neg P \vee Q$	TAATAAGGAA
	TCATGTTCCT
$\neg Q$	CATGA

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-stranded molecule (see Fig. 5).

4.2 Implementation of Resolution Refutation

As in the linear implementation, resolution between two clauses is implemented 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 negation of goal variable only as a primer. To read the PCR product, we used gel electrophoresis. If a band is formed, we can conclude that the goal is consistent with the given clauses. If not, we say that the goal is not consistent with them. Because each band corresponds to a proof, we can find several different proofs at one time.

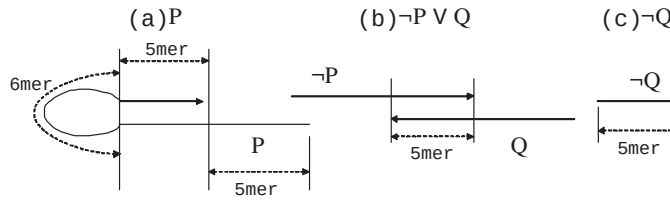

Fig. 6. The form of molecules used in experiments for hairpin implementation. (The arrows are from 5' to 3'.)

Table 4. The PCR condition for hairpin implementation

cycle	denaturation (98°C)	annealing (58°C)	polymerization (72°C)
1	5 min	1 min	1 min
2 ∼ 26	1 min	1 min	3 min
27	1 min	1 min	7 min

4.3 Experimental Results

To test our idea, we solved a very simple theorem proving problem: given P and $P \rightarrow Q$, is Q consistent with them? Putting it in clause form, we get the clauses $\{P, \neg P \vee Q, \neg Q\}$. The form of molecules representing these clauses are given in Fig. 6. As in the previous experiment, all sequences were designed by NACST [14] 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 should have used $\neg Q$, the negation of the goal variable, as a primer in principle. But in this case, the sequence corresponding to it is too short to use it as a primer. Therefore, we used lower part of the molecule in Fig. 6-(b) as a primer. We used 3% agarose gel at the gel electrophoresis step.

The electrophorestogram of hairpin implementation is given in Fig. 7. In Fig. 7, we can see bands of 23bp and 46bp as expected in Fig. 6. From this result, we can say our method can find a proof if it exist. If there is no such proof, we should fail to get band with that length. To verify this, we performed the same experiments without one or more molecules in Fig. 6 (see Fig. 8). Only lane 12 in Fig. 8 contains all of them. Therefore a band must appear in lane 12 only. But we can see short bands in lane 2∼7. We think that short bands in lane 3∼7 are formed by hybridization of two Ps(see Fig. 6-(a)). And the longer bands in lane 2 seems to be formed by hybridization of P and $\neg P \vee Q$ (see Fig. 6-(a) and Fig. 6-(b)). But this longer band must be shorter than the band in lane 12, for it does not contain molecule $\neg Q$. To compare both lanes, we draw dashed lines

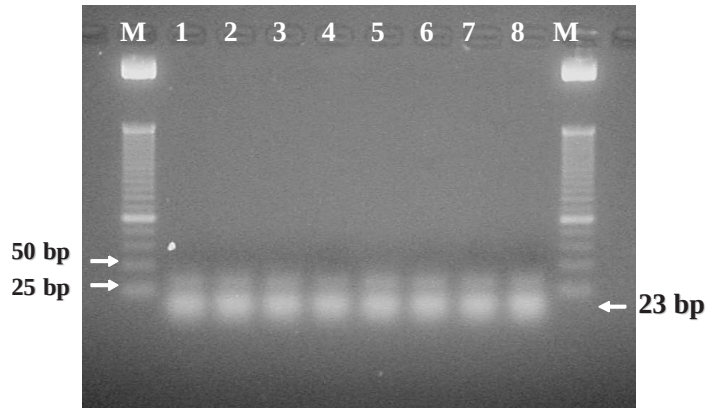

Fig. 7. The result of ligation mixture after PCR and electrophoresis. Lane 1-8: Ligation mixture under various conditions. Lane M: a 25 bp size marker. The arrow indicates the 23 bp PCR product.

in Fig. 8. We find that the band in lane 2 is shorter than that of lane 12 as we expected.

4.4 Discussion

In this experiment, because of self-complementary sequences such as hairpin and group of sequences with complementary subsequence such as branch, PCR step is extremely difficult. If some of these complementary sequences hybridize during PCR step due to their thermodynamical properties such as melting temperature, amplification will stop at that point and we will get false negative result. Also as you can see in Fig. 7, one proof can have two bands of different length. We think it is due to the hybridization of two hairpin molecules.

And there are some cases when our method can not tell whether a proof is found or not. For example, if the goal variable is resolved more than once, our method will tell there exists a proof regardless of the existence of empty clause. To solve this problem, we need a different detection method for empty clause or a different encoding scheme.

Also, for we did not restricted the form of clause, non-Horn clause can form a complicated self-loop structure. And even if we restrict ourselves to Horn-clauses, if there are two clauses which have two resolvable variables, they can form a self-loop structure, too. For example, if there are $\neg P \vee Q$ and $P \vee \neg Q \vee \neg R$, after resolving P or Q, the resolvent can form a self-loop structure. What is worse, as the number of variable grows, we should use longer sequences to encode each variable and the possibility of self-hybridization will grow. To solve this problem, we need to make procedure removing this self-loop or to design new representations for clauses.

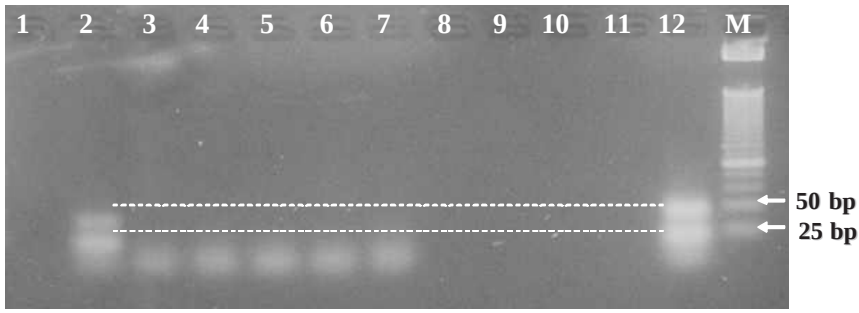

Fig. 8. The verification of detection method. Lane 1: without Fig. 6-(a). Lane 2: without Fig. 6-(c). Lane 3: without upper part of Fig. 6-(b). Lane 4: without lower part of Fig. 6-(b). Lane 5: Fig. 6-(a) and (c) only. Lane 6: Fig. 6-(a) and lower part of (b) only. Lane 7: Fig. 6-(a) and upper part of (b) only. Lane 8: Fig. 6-(b) only. Lane 9: lower part of Fig. 6-(b) and (c) only. Lane 10: upper part of Fig. 6-(b) and (c) only. Lane 11: the same as lane 8. Lane 12: with all of molecules. Lane M: a 25bp size marker.

When using branched molecules, branch migration may occur. But as suggested in [12], inserting T's to junction point can reduce the possibility of branch migration.

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).

As we discussed above, there are several things to consider in both implementations. And we are trying to improve our implementations.

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 Evo-

lutionary 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. Sa-Ardyen, P., Jonoska, N., and Seeman, N.C., Self-assembling DNA Graphs, *Preliminary Proceedings of the Eighth International Meeting on DNA Based Computers*, 20–28, 2002.
13. 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.
14. 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).
15. Suttner, C., SPTHEO - A parallel theorem prover, *Journal of Automated Reasoning*, **18**(2):253-258, 1997.
16. 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.
17. Wasiewicz, P., Janczak, T., Mulawka, J.J., and Pluciennicza k, A., The inference based on molecular computing, *International Journal of Cybernetics and Systems*, **31**(3):283–315, 2000.
18. Winfree, E., *Algorithmic self-assembly of DNA*, Ph.D. Thesis, California Institute of Technology, 1998.