Towards Parallel Evaluation and Learning of Boolean μ-Formulas with Molecules

Masami Hagiya, Masanori Arita
Department of Information Science
Daisuke Kiga, Kensaku Sakamoto, Shigeyuki Yokoyama
Department of Biophysics and Biochemistry

Graduate School of Science, University of Tokyo
Bunkyo–ku Hongo 7–3–1, 113 Tokyo, Japan

ABSTRACT

A μ–formula is a Boolean formula in which each variable occurs at most once. The paper treats its molecular representation including its queries using techniques in molecular biology. The novelty is that this method can evaluate a Boolean formula in a single tube within a short time. The preliminary experimental result suggests the possibility of parallel evaluation and learning of general μ–formulas. This method is essentially a simulation of state transitions and can be used to simulate a decision tree or a state machine.

1 Introduction

From the beginning of molecular computation, its parallelism is what theoretical analyses are concerned with and what researchers have tried to fully utilize in lab experiments. Despite the initial optimism, however, researchers came to know pros and cons of molecular operations [11]. The most unfavorable is the instability of lab operations susceptible to temperature, concentration and enzyme activity. Another hurdle is the difficulty in realizing complex procedures and data structures in terms of primitive operations on molecules. Although DNA molecules can be concatenated or rotated by lab operations [2], the set of operations on DNA molecules is still far behind what can be called programmable. A molecular Random Access Machine (RAM) is only theoretically examined, and has not yet been realized [10]. Even with these obstacles, however, the parallelism inherent in molecular computation is hard to abandon. When fully utilized, this parallelism could be a great tool for a wide range of computation.

With standard molecular biological techniques, the best way to utilize the molecular parallelism is to execute what can be called *generate and search* as was initially performed by Adleman [1]. He first generated at random all the candidate solutions for an instance of Hamiltonian path problem, then chose the correct solution from them in standard lab operations. Lipton's method for solving a 3-SAT problem is also along the line of *generate and search* [5].

However, the power of *generate and search* with molecules is currently far from sufficient for solving actual problems because the available operations on molecules are severely limited. In order to make molecular computation practically feasible, it is vital to prepare molecular operations that allow complex procedures and data structures.

In this paper, a molecular method for parallel evaluation of multiple μ–formulas is introduced. A μ–formula is a Boolean formula in which each variable occurs at most once. Our ultimate goal is to learn μ–formulas from examples utilizing the parallelism of molecules. Pitt and Valiant showed that μ–formulas are not learnable from examples alone in polynomial time [9]. In our method, however, all the possible μ–formulas can be randomly generated, and then only formulas satisfying the given variable assignments can be selected in a single tube in polynomial time.

Unlike in other approaches towards evaluation of Boolean circuits with molecules [5, 7, 4], we represent both Boolean formulas and their variable assignments with DNA molecules. This is a significant progress from the existing paradigm for molecular computation. Since the DNA molecules representing Boolean formulas can be regarded as programs, and molecules for variable assignments can be regarded as inputs, our approach allows multiple programs with different inputs to run in parallel (MPMD or MIMD). On the other hand, in previous approaches, programs are implemented as a sequence of operations that are uniformly applied

on molecules in a tube (SIMD).

In our approach, a variable assignment (or an input), a Boolean formula (or a program), and the current state of evaluation (or a head in Turing machine's sense) are encoded in one single–stranded DNA in this order (Fig.1). Evaluation of the Boolean formula with its variable assignment is performed by state transitions implemented by molecular operations. In each transition, the head reads information from the Boolean formula as well as from the assingment to change the state. More precisely, single–stranded DNA forms a hairpin structure where the current state hybridizes to an appropriate position in the formula or the assignment, then the 3' end of the state is extended by one symbol to obtain the next state using polymerization.

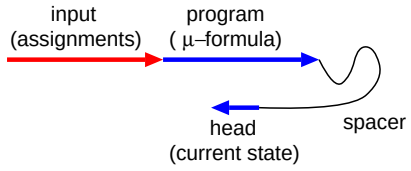

Figure 1: Our implementation

The contribution of this work is twofold: in theory and in experimental technique. In theory, we implement a successive state transitions of multiple μ-formulas in parallel as a single–tube reaction. This method is, to the authors' knowledge, the first molecular computation in which both programs and data are implemented and evaluated in parallel. The transitions are realized as a succession of thermal cycles, achieving parallel evaluation of Boolean formulas in a single tube.

As a new experimental technique, we also introduce a method to stop polymerization exactly after the head copies one symbol. For this purpose, we use a stopper sequence consisting of a deoxyribonucleotide whose complement is missing in the polymerization buffer. We call this technique *polymerization stop*.

This paper also reports the result of two lab experiments that have been done for showing the feasibility of the above method. One is that of a single transition and the other checked two successive transitions.

Utilizing the molecular operations for state transitions, we want to solve computationally hard combinatorial problems by generate and search. As is said before, we propose a method for learning Boolean μ–formulas from examples. Since we can randomly generate and then evaluate Boolean formulas in parallel, it is possible to select those formulas that behave as specified by examples.

The rest of this paper is organized as follows. Section 2 explains how μ–formulas are represented with molecules, and how they are generated and selected. Section 3 introduces experimental results for the basic operations required. The last section is for discussions, including that of implementing state machines with molecules in general.

2 Molecular representation

A μ–formula is inductively defined as follows.

- A Boolean variable is a μ–formula.
- If e is a μ–formula, then its negation $\neg e$ is also a μ–formula.
- If e_1 and e_2 are both μ–formulas, then so are their conjunction $e_1 \wedge e_2$ and disjunction $e_1 \vee e_2$, provided that e_1 and e_2 do not share a common variable.

In our method, the number of Boolean variables that can appear in μ–formulas is fixed, say n. They are denoted by n symbols, `var1`, `var2`, $\cdots$, `var`n. For short, we also write v_1, v_2, $\cdots$, v_n, respectively.

A Boolean vector whose length is the same as the number of Boolean variables is called an *input vector*, because it determines an assignment of true or false to each Boolean variable. The value of a μ–formula with respect to an input vector is defined as usual.

The symbols in Fig.2 are used to represent μ–formulas and input vectors. Each symbol will be encoded into a single–stranded DNA sequence.

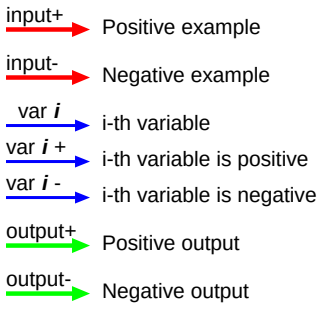

Figure 2: Symbols

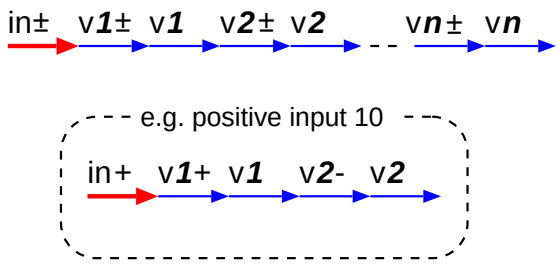

Figure 3: Input vectors

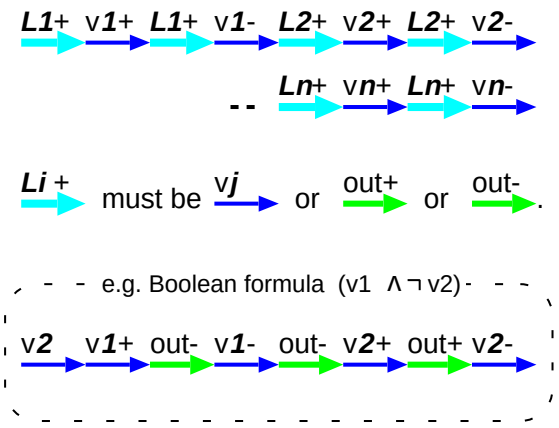

Figure 4: μ–formula

Without loss of generality, the first variable in each μ–formula can be fixed. In the following, we assume that the first variable in a μ–formula is always `var1`, or v_1 for short.

2.1 Input vectors

Each input vector is represented by the sequence of symbols as in Fig.3. The 5'–end symbol denotes whether the input is positive or negative; i.e., the sequence begins with `input+` (`in+` hereafter) if a μ–formula should take the value true for this input, and it begins with `input-` (`in-`) if the value is expected to be false. To the 3' downstream of this symbol, the assignment of true or false for each of the n Boolean variables v_1, v_2, $\cdots$, v_n is listed. If true is assigned to the variable v_i, it is preceded by v_i+. If false, it is preceded by v_i-. An example, the positive input vector `10`, is shown in Fig.3.

$$\texttt{trans}(v_i, x, y) = x \xrightarrow{\quad} v_i+ \xrightarrow{\quad} y \xrightarrow{\quad} v_i-$$

$$\texttt{trans}(\neg e, x, y) = \texttt{trans}(e, y, x)$$

$$\texttt{trans}(e_1 \wedge e_2, x, y) = \texttt{concatenate}(\texttt{trans}(e_1, \texttt{first}(e_2), y), \texttt{trans}(e_2, x, y))$$

$$\texttt{trans}(e_1 \vee e_2, x, y) = \texttt{concatenate}(\texttt{trans}(e_1, x, \texttt{first}(e_2)), \texttt{trans}(e_2, x, y))$$

$$\texttt{first}(v_i) = v_i$$

$$\texttt{first}(\neg e) = \texttt{first}(e)$$

$$\texttt{first}(e_1 \wedge e_2) = \texttt{first}(e_1)$$

$$\texttt{first}(e_1 \vee e_2) = \texttt{first}(e_1)$$

Table 1: Translation of μ-formulas

2.2 μ–formula

Each μ–formula is represented by a sequence of $4 \times n$ symbols of the form in Fig.4. A μ–formula is translated into the above form in such a way that its subsequence is interpreted as follows:

$v_j \xrightarrow{\quad} v_i+$: If the value of the i-th variable is true, then jump to the place to check the j-th variable.

$v_j \xrightarrow{\quad} v_i-$: If the value of the i-th variable is false, then jump to the place to check the j-th variable.

`out+` $\xrightarrow{\quad} v_i+$: If the value of the i-th variable is true, then terminate with the output `1` (true).

`out+` $\xrightarrow{\quad} v_i-$: If the value of the i-th variable is false, then terminate with the output `1` (true).

`out-` $\xrightarrow{\quad} v_i\pm$: Similarly, produce the output `0` (false).

The algorithm for translating μ–formulas to their representations is trivial. It is described by the function $\texttt{trans}(e, x, y)$ in Table.1. One can obtain the representation of μ–formula e by computing $\texttt{trans}(e, \texttt{out+}, \texttt{out-})$.

A μ–formula is equivalent to a decision tree in this translation. Note that a variable to be evaluated next can be uniquely determined in μ–formula. A general logical formula does not have this property.

2.3 Evaluation

Given an input vector and a μ–formula, its value can be evaluated using the concatenated sequence in Fig.5. The 3' end is a Watson–Crick complement

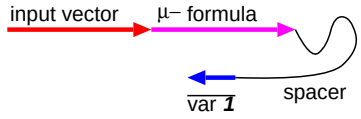

Figure 5: Evaluation

of the symbol v_1 bound by a spacer sequence to the input vector and the μ–formula. The entire sequence is assumed to be single–stranded.

The evaluation of the μ–formula consists of the repetition of the following three steps (Fig.6):

1. Anneal the 3' end symbol of the sequence to its complementary part of the same sequence.
2. Add exactly one symbol to the 3' end by extending it according to the result of annealing.
3. Denature the 3' end, and go to the first step.

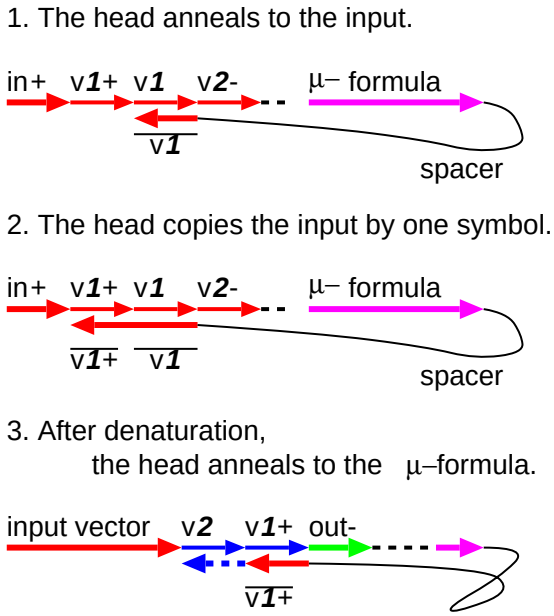

Figure 6: Evaluation steps

In order to stop the extension exactly after one symbol is added to the 3' end, both input vectors and μ–formulas contain a stopper sequence (denoted by `stop`) in between symbols. The actual implementation of the stopper sequence is described in Sec. 3.

2.4 Selection

After repeating the evaluation steps, the 3' end of a sequence eventually becomes `out+` or `out-`, meaning that the value of the μ–formula is true or false for the given input vector. Therefore, if the sign of `out`$\pm$ is identical to that of `in`$\pm$, the μ–formula has been evaluated as expected. This can be detected by PCR with `in`$\pm$ and `out`$\pm$ as primers.

Given a series of input vectors, it is possible to select only those μ–formulas that produce the expected result for all the input vectors from a pool of candidates in a test tube. By the process of evaluation and selection described in the last section, only valid μ–formulas are successively chosen for each input vector as follows:

1. Concatenate the copy of an input vector to the 5' end of each μ–formula in the tube. Also concatenate a spacer sequence and the symbol $\overline{v_1}$ to its 3' end.
2. Apply the process of evaluation and selection described in the last section.
3. Remove the copy of an input vector and the spacer with extended symbols from each μ–formula, so that the next input vector can be processed.

3 Experimental results

This section describes the experiments that have been done for showing the feasibility of the approach proposed in this paper. We have done two experiments: one is that of a single transition by extension of a hairpin structure, and the other is that of two successive transitions, where the DNA fragment produced in the first transition is denatured and the second transition is performed in the same tube.

In both experiments, symbols are assumed to be encoded with three out of the four common deoxyribonucleotides (dATP, dGTP, dCTP and dTTP). If a polymerization buffer contains only these three deoxyribonucleotides, a repetition of the missing deoxyribonucleotide works as a stopper sequence in polymerization. In the actual experiments, we used a triplet "`ggg`" or "`aaa`" as the stopper.

3.1 Single transition

Sequence design

The triplet "ggg" is used as a stopper sequence in Table 2. All the sequences are single–stranded if not specially mentioned. A sequence with a bar indicates its Watson–Crick complement. For example, $\overline{\texttt{s1}}$ represents the complementary sequence of s1. A hyphen between sequences indicates their concatenation, i.e., phosphodiester bonding between the sequences. For example, s1-s2 represents the concatenated sequence of s1 and s2 in this order.

The sequence ini-s1-s2-s3-$\overline{\texttt{s2}}$ is chemically synthesized in this experiment. Note that the subsequence ini contains the stopper ggg. Also synthesized are ini-s1-s2-s3-$\overline{\texttt{s2}}$-$\overline{\texttt{s1}}$ and ini-s1-s2-s3-$\overline{\texttt{s2}}$-$\overline{\texttt{s1}}$-$\overline{\texttt{ini}}$ as markers for electrophoresis.

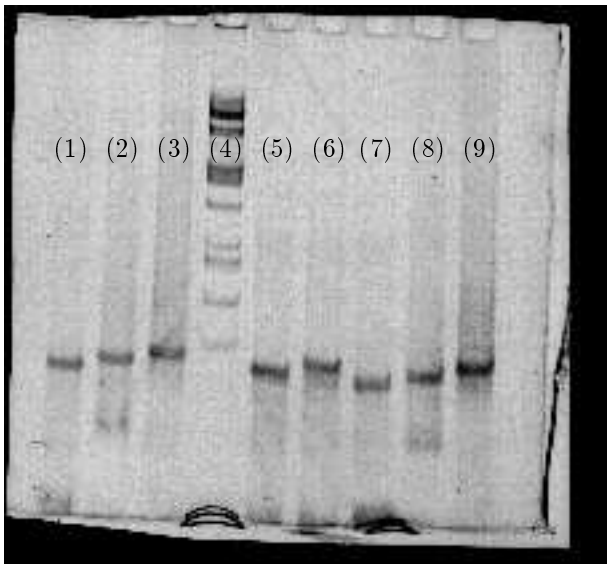

Figure 7: Electrophoresis without urea

Transition

The buffer in this experiment lacks dCTP. In 20 μl scale, the buffer contains 2 μg of DNA, 250 μM of dATP, dGTP and dTTP, and 2 units of AmpliTaq DNA polymerase (Perkin Elmer). Other conditions are the same as given by the supplier.

For comparison, we also prepared the buffer containing 250 μM of dCTP together with other three deoxyribonucleotides. In this case, it is expected

s1	tccattctaa
s2	ctcacccttatacat
s3	cgtttcag
ini	cgacaagcttggg

Table 2: DNA sequences 1

that polymerization does not stop and produces ini-s1-s2-s3-$\overline{\texttt{s2}}$-$\overline{\texttt{s1}}$-$\overline{\texttt{ini}}$.

Transition was performed by initial denaturaturing at 90 ℃ for 3 min, followed by an extension step, 68 ℃ for 10 min, using Perkin Elmer DNA Thermal Cycler 480. Finally, the reaction mixture was cooled down on ice.

Electrophoresis

Samples were analyzed by electrophoresis on 12% polyacrylamide gels with or without 8 M urea. The band of the transition product was compared with those of the two markers: one is ini-s1-s2-s3-$\overline{\texttt{s2}}$-$\overline{\texttt{s1}}$ and the other is ini-s1-s2-s3-$\overline{\texttt{s2}}$-$\overline{\texttt{s1}}$-$\overline{\texttt{ini}}$.

The result of electrophoresis without urea is shown in Fig.7. From the left, the lanes correspond to:

(1) the initial DNA ini-s1-s2-s3-$\overline{\texttt{s2}}$,
(2) the marker ini-s1-s2-s3-$\overline{\texttt{s2}}$-s1,
(3) the marker ini-s1-s2-s3-$\overline{\texttt{s2}}$-$\overline{\texttt{s1}}$-$\overline{\texttt{ini}}$,
(4) some known standard markers,
(5) the polymerization product with three deoxyribonucleotides,
(6) the polymerization product with four deoxyribonucleotides,
(7) the initial DNA ini-s1-s2-s3-$\overline{\texttt{s2}}$,
(8) the marker ini-s1-s2-s3-$\overline{\texttt{s2}}$-$\overline{\texttt{s1}}$, and
(9) the marker ini-s1-s2-s3-$\overline{\texttt{s2}}$-$\overline{\texttt{s1}}$-$\overline{\texttt{ini}}$.

In lanes (1), (2) and (3), DNA molecules were dissolved in water and renatured (cooled down on ice). Lanes (7), (8) and (9) contain the same molecules as in (1), (2) and (3), respectively, but they were dissolved in the polymerization buffer without AmpliTaq DNA polymerase; they were also renatured before electrophoresis.

The DNA molecules in these lanes each produced a single sharp band, indicating that they each formed a single stable structure, probably a hairpin structure. In contrast, the DNA molecules which were not renatured showed a smeared band

s1	gcacgatctaggaaa
s2	tcccgttctgggcct
s3	gtggtttgcgctcgt
ini	ccaaa

Table 3: DNA sequences 2

as well as another band unmoved during the electrophoresis. These bands probably indicate the formation of various aggregates (data not shown).

In lane (5), the band of the transition product corresponds to that of the marker ini-s1-s2-s3-$\overline{\text{s2}}$-$\overline{\text{s1}}$ in (8), while there is no band corresponding to that of ini-s1-s2-s3-$\overline{\text{s2}}$-$\overline{\text{s1}}$-$\overline{\text{ini}}$ in (9). This means that hairpin structures were properly formed and their extension was stopped after $\overline{\text{s1}}$ was copied. On the other hand, the extension with all four deoxyribonucleotides produced a band in lane (6), corresponding to the marker in lane (9).

Denaturing gel electrophoresis with 8 M urea exhibited a pattern similar in correspondence between the transition products and the markers to Fig.7 (data not shown).

3.2 Two successive transitions

Sequence design

In this experiment, we used "aaa" as a stopper sequence.

The sequence used is ini-s2-s1-s3-s2-$\overline{\text{s1}}$ as defined in Table 3. The expected extension is shown in Fig.8.

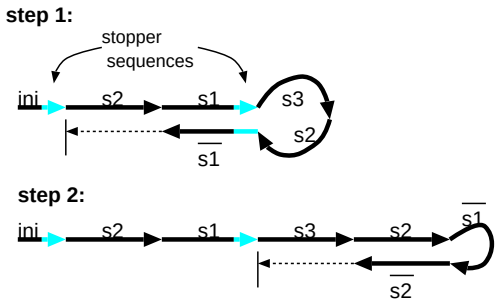

Figure 8: Expected extension

Transition

The buffer used in this experiment lacks dTTP. In 20 μl scale, the buffer contains 1 μg of DNA, 250 μM of dATP, dGTP and dCTP, and 2 units of AmpliTaq DNA polymerase (Perkin Elmer). Other conditions are the same as given by the supplier.

We programmed Perkin Elmer DNA Thermal Cycler 480 as follows: initial denaturation step, 90 ℃ for 1 min; extension step, 68 ℃ for 1 min; four cycles with denaturation step at 90 ℃ for 1 min, cooling down on ice for 1 min, incubation at 40 ℃ for 30 sec, and extension at 68 ℃ for 30 sec. The cooling step is for forming hairpin structures before extension.[1]

Electrophoresis

Samples were analyzed by electrophoresis on 12% polyacrylamide gels containing 8 M urea. The bands of the transition products were compared with those of the markers, which had been chemically synthesized.

The product of the first extension step exibiteqd two bands in electrophoresis. One is that of ini-s1-s2-s3-$\overline{\text{s2}}$. The other is considered that of ini-s1-s2-s3-$\overline{\text{s2}}$-$\overline{\text{s1}}$.

After the four cycles of denaturation, annealing, and cooling, the above two bands disappeared and the band corresponding to that of the marker ini-s2-s1-s3-s2-$\overline{\text{s1}}$-$\overline{\text{s2}}$-$\overline{\text{s3}}$ emerged.

Sequencing of Transition Product

The DNA molecules that have undergone the successive transitions was processed by terminal deoxynucleotidyl transferase, and thus a poly-T sequence was added to the tail of each DNA molecule. It was then amplified by PCR with the primers of the sequences shown in Table 4. Primer 1 includes s1 and Primer 2 has a poly-A sequence. The product of PCR was digested with EcoRI and SalI, and was ligated to the corresponding sites in vector pUC119. The sequences of the clones were determined, confirming that the successive transitions certainly occurred as we expected.

4 Discussions

In this section, we first discuss some crucial points for the implementation of state transitions. Then

[1] When cooled down, tubes were temporarily detatched from the thermal cycler and put on ice.

Primer 1	gctcgtcgac
	gcacgatctaggaaa
	gtgg
Primer 2	gacggaattc
	aaaaaaaaaaaaaaaaaaaa
	gtgg

Table 4: Primers for cloning

we introduce how to implement a finite automaton using our method, and compare its effectiveness with yet another implementation of state transition using PCR.

4.1 Experimental techniques

As crucial points, we consider another implementation of polymerization stop, and some conditions for more efficient state transitions.

Polymerization stop

In the previous section, we described the technique of polymerization stop using triplets as stopper sequences. This technique, however, requires encoding using only three deoxyribonucleotides.

We plan to use artificial nucleosides such as isoguanosine or iso-cytosine [8] for stopper sequences. As a result, all four common deoxyribonucleotides can be used for encoding. This allows more information to be encoded in sequences of the same length. Another advantage is that it is easier to adjust the gc-contents at an appropriate level.

Intra-molecular reaction

In our approach, since a program and its input reside on a single DNA strand, parallel execution of multiple programs requires that intra–molecular reaction occur exclusively.

In the above experiments, we cooled down single-stranded DNA molecules on ice to facilitate intra–molecular reaction, i.e., to prevent two or more molecules from interacting with one another.

Facilitating intra–molecular reaction with a sufficient certainty is an important issue to be settled down. There is a report of the stable formation of a DNA hairpin *in vitro* [3], and the adjustment in experimental conditions will greatly improve the efficiency and the accuracy of extension. So far, we propose the following two simple methods:

- Keep the concentration of DNA molecules sufficiently low.
- Quickly cool down DNA molecules during annealing.

The effectiveness of these methods should be verified in further experiments.

We also plan to utilize surface chemistry [2, 4]. By binding DNA molecules sparsely on a surface, it is possible to avoid inter–molecular interaction, thus facilitating intra–molecular reaction.

Successive hairpin formation

To perform two successive transitions, as in our last experiment, thermal cycles are repeated so that a hairpin structure once created and extended is denatured, and a new hairpin is formed for the second transition. However, it is highly probable that the first structure is formed again and no successive transition occurs (Fig.9).

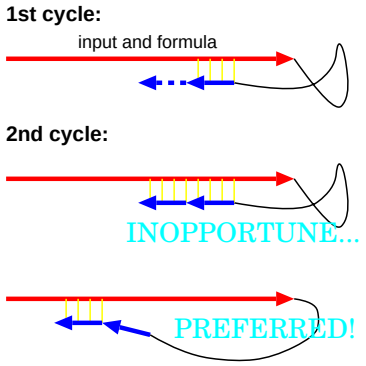

Figure 9: Forming the first structure again

This problem does not occur if the probability of the second structure is reasonably large compared with (not necessarily larger than) the first one, because by repeating thermal cycles, the second structure will be eventually formed. Note that the formation of the first structure produces no harmful results, because further extension from this structure is prohibited by the stopper sequence.

The stability of a hairpin structure is affected by the length of its loop. Sequences used in our experiment were designed so that the second structure has higher stability. More experiments and theoretical analyses are required to settle down this issue.

4.2 Finite automata

The method introduced in this paper is essentially that for simulating deterministic or non–deterministic state transitions. Therefore, it can be applied to implement state machines in general, and we have already considered how to implement finite automata with molecules.

The transition table of a finite (deterministic or non–deterministic) automaton is a sequence of triples of the form s'-c-s, where s denotes the current state and s' the next state corresponds to the input symbol c. For example, the table of the automaton M in Fig.10 is represented by the sequence `P-0-S-Q-1-S-Q-1-P-R-0-P`.

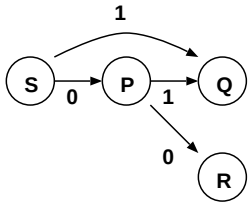

Figure 10: Automaton M

To realize state transitions, we concatenate the complement $\overline{s}$ of the current state s of the automaton as in Fig.11, where a spacer sequence with an appropriate length is inserted in between the table and the state.

The state transition corresponding to the input symbol c is performed as follows. First, $\overline{c}$ is attached to the end of the sequence using a method for concatenating DNA molecules as in [6]. The end consists of the subsequence $\overline{s}$-$\overline{c}$, which can hybridize to the triple s'-c-s in the transition table in the hairpin structure. By polymerization, the next state $\overline{s'}$ is copied to the end.

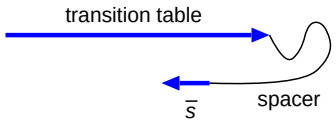

Figure 11: Current state of an automaton

To stop polymeration right after $\overline{s'}$ is copied, we again use the technique of polymerization stop.

By the method described above, it is possible to run finite automata in parallel with respect to an input sequence of symbols.

4.3 Yet another representation

State transitions can be realized still in other ways. We introduce another method in which one PCR step corresponds to one transition step, and compare its effetiveness with our original method.

We briefly explain the new method using the example automaton M in Fig.10.

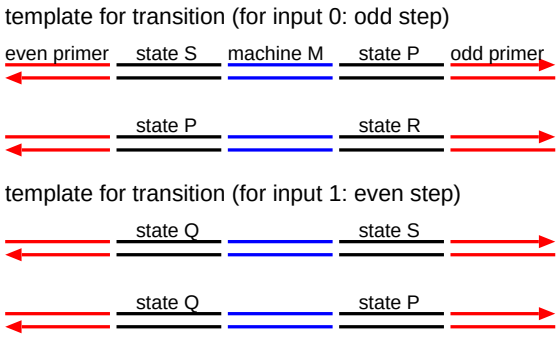

Figure 12: Another representation

As in Fig.12, each transition is represented by a double–stranded sequence of symbols used later as a PCR template. A different set of templates is prepared for each input (in this case `0` or `1`), i.e., the possible transitions for the input in M is represented by a set of PCR templates. The current state is represented by a primer so that only templates containing the next state is amplified in PCR. The primer is changed by turns (`odd` or `even`) to proceed the transition. A sample transition of the automaton M for the input `01` is shown in Fig.13. The corresponding molecular operation is shown in Fig.14.

This method can be easily parallelized by changing the name of an automaton in the middle of templates. The amount of each PCR template required for an amplification is very small, so many automata can be simulated in parallel.

This method (*PCR transition* for short) uses PCR, a standard operation when compared to our single–tube polymerization. Our method, however, is considered better for several reasons.

Random generation of data

In general, synthesis of molecular data is a costly process. In a generate and search approach, it is important to randomly generate all and only valid data. DNA for the single–tube polymerization can be randomly generated by a stepwise extension of

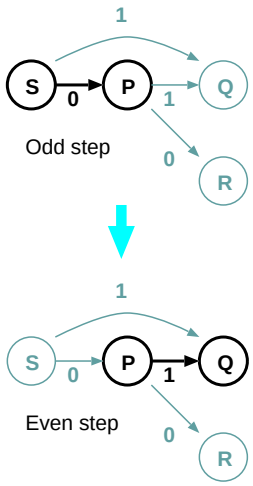

Figure 13: State transition

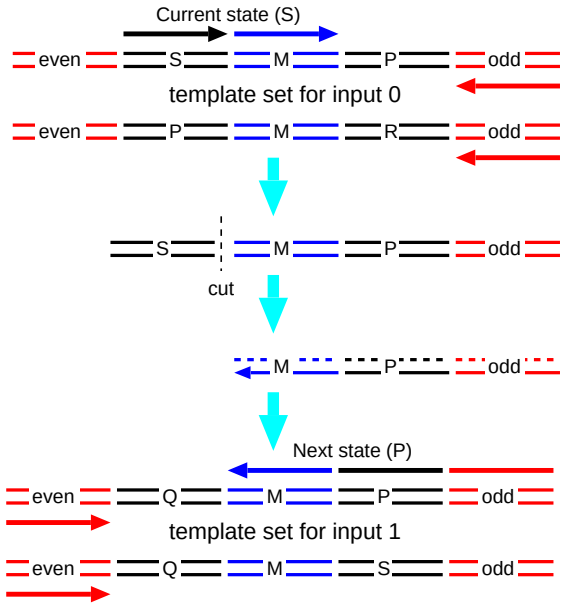

Figure 14: DNA for state transition

symbols as in [6], but DNA for the PCR transition is hard to randomly generate. The reason is that PCR templates representing the same automaton must have the same machine name at their middle part. With this restriction, it is almost impossible to randomly generate all the possible automaton of the given number of nodes.

Time in experiment

PCR or gel electrophoresis requires more than an hour to perform, while enzymatic modification (e.g. ligation or restriction) requires much shorter time. (e.g. 10 min. The time depends on temperature and concentration of enzymes.) The single–tube polymerization can repeat the transition steps using only these quicker operations without changing buffers, a great advantage which suggests the automation of the repetitive process like PCR in the future.

Amount of molecules

To preserve the intermediate variety of solutions, each solution should not be stored in a large population. PCR is certainly a reliable operation for amplification, but at the same time, it runs the risk of extermination of solutions of a small population. As long as the total population does not become too small, PCR should be avoided in a generate and search approach.

Acknowledgments

The second and the third authors are supported by Research Fellowships of the Japan Society for the Promotion of Science for Young Scientists. This work is also supported by the Japan Society for the Promotion of Science “Research for the Future” Program (JSPS-RFTF 96I00101).

References

[1] L. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.

[2] M. Arita, M. Hagiya and A Suyama. Joining and rotating data with molecules. In *Proc. ICEC'97 (to appear)*, 1997.

[3] R.R. Breaker and G.F. Joyce. Emergence of a replicating species from an *in vitro* RNA evolution reaction. *Proc. Natl. Acad. Sci. USA*, 91:6093–6097, 1994.

[4] W. Cai, A.E. Condon, and R.M. Corn *et al.* The power of surface-based DNA computation. In *Proc. RECOMB'97*, pages 67–74, 1997.

[5] R. Lipton. DNA solution of hard computational problems. *Science*, 268:542–544, 1995.

[6] N. Morimoto, M. Arita, and A. Suyama. Stepwise generation of hamiltonian path with molecules. In *BCEC'97 (under submission)*, 1997.

[7] M. Ogihara and A. Ray. Simulating boolean circuits on a DNA computer. Technical Report 631, University of Rochester, 1996.

[8] J.A. Piccirilli *et al.* Enzymatic incorporation of a new base pair into DNA and RNA extends the genetic alphabet. *Nature*, 343:33–37, 1990.

[9] L. Pitt and G.L. Valiant. Computational limitations on learning from examples. *J. Assoc. Comput. Mach.*, 35(4):965–984, 1988.

[10] E. Roweis, S. Winfree and B. Richard *et al.* A sticker based architecture for DNA computation. In *Proc. 2nd meeting on DNA based computers*, pages 1–27, 1996.

[11] H. Seeman, N. Wang and B. Liu *et al.* The perils of polynucleotides: the experimental gap between the design and assembly of unusual DNA structures. In *Proc. 2nd meeting on DNA based computers*, pages 191–205, 1996.