Addressable procedures for logic and arithmetic operations with DNA strands

Akihiro Fujiwara    Ken’ichi Matsumoto
Department of Computer Science and Electronics
Kyushu Institute of Technology
Fukuoka 820-8502, JAPAN
_fujiwara@cse.kyutech.ac.jp

Wei Chen
Department of Computer Science
Tennessee State University
Nashville, TN37209, U.S.A.

Abstract

In this paper, we consider addressable procedures with DNA strands for logic and arithmetic operations. Using a theoretical model for DNA computing, we first show a DNA representation of \( n \) binary numbers of \( m \) bits, and propose a procedure to assign values for the representation. The procedure is applicable to \( n \) binary numbers of \( m \) bits in \( O(1) \) lab steps in parallel. Next, we propose a procedure for logic operations. The procedure enables any boolean operation whose input and output are defined by a truth table, and executes different kinds of boolean operations simultaneously for any pair of \( n \) binary numbers of \( m \) bits in \( O(1) \) lab steps using \( O(mn) \) DNA strands. Finally, we propose a procedure for additions of pairs of binary numbers. The procedure works in \( O(1) \) lab steps using \( O(mn) \) DNA strands for \( O(n) \) additions of two \( m \)-bit binary numbers.

1. Introduction

In recent works for high performance computing, computation with DNA molecules, that is, DNA computing, has considerable attention as one of non-silicon based compute- mings. The massive parallelism of DNA strands enables us to solve combinatorial \( NP \)-complete problems, which need exponential computation time on a silicon based computer, in a polynomial number of steps. As the first work for DNA computing, Adleman[1] presented an idea of solving the Hamiltonian path problem of size \( n \) in \( O(n^2) \) lab steps using DNA molecules. His idea is successfully tested in a lab experiment for a small graph. Lipton[6] showed methods for solving the SAT problem of size \( n \) in \( O(n^2) \) lab steps using DNA molecules. There are a number of other works with DNA molecules for combinatorial \( NP \)-complete problems.

However, procedures for primitive operations, such as logic or arithmetic operations, are needed to apply DNA computing on a wide range of problems. There are some works for primitive operations in DNA computing[2, 3, 4, 5, 9]. Guarnieri et.al.[3] has proposed the first procedure for the addition of two binary numbers using DNA molecules. The procedure works in \( O(n) \) lab steps using \( O(n) \) different DNA strands for an addition of two \( n \)-bit binary numbers. Recently, Hug and Schuler[5] proposed a model for representing and manipulating binary numbers on the DNA chip, which allows parallel execution of a primitive operation. Their procedure computes an addition of two \( n \)-bit binary numbers in \( O(1) \) lab steps using \( O(n) \) different DNA strands. However, their procedure allows only one single operation in parallel execution for DNA strands.

In this paper, we consider addressable procedures for the primitive operations using DNA strands. The primitive operations are used on a silicon based computer with memory addressing, that is, each variable is stored in a memory location whose size is a constant number of bits, and an operation is executed for two memory locations indicated by the addresses. If the addressing feature is used in DNA computing, we can execute different operations for different variables stored in DNA strands.

Using a theoretical model for DNA computing, we first show a DNA representation of \( n \) binary numbers of \( m \) bits, and propose a procedure to assign values for the representation. The procedure is applicable to \( n \) binary numbers of \( m \) bits in \( O(1) \) lab steps in parallel. Next, we propose a procedure for logic operations. The procedure enables any boolean operation whose input and output are defined by a truth table, and executes different kinds of boolean operations simultaneously for any pair of \( n \) binary numbers of \( m \) bits in \( O(1) \) lab steps using \( O(mn) \) different kinds of DNA strands. Finally, we propose a procedure for additions of pairs of binary numbers. The procedure works in \( O(1) \) lab steps using \( O(mn) \) different kinds of DNA strands for \( O(n) \) additions of two \( m \)-bit binary numbers.

This paper is organized as follows. In Section 2, we give the brief description of the model for DNA computing. In Section 3, we show the DNA representation of binary numbers and propose a procedure for a basic operation. In Section 4 and Section 5, we propose procedures for logic and
arithmetic operations, respectively. Section 6 concludes the paper.

2. Preliminaries

2.1. DNA strands

A set of DNA strands is a key component for DNA computing, like a memory module used on a silicon based computer. A *single strand* of DNA is defined as a string of four different base nucleotides. Since any kind of single strands can be synthesized using biological methods[7], we assume that each single strand represent a symbol over a finite alphabet Σ.

A main concept used in DNA computing is Watson-Crick complementarity. We define the alphabet Σ = \{σ₀, σ₁, ..., σₘ₋₄, ⃗σ₀, ⃗σ₁, ..., ⃗σₘ₋₄\}, where the symbols \(σ_i, ⃗σ_i\) \((0 ≤ i ≤ m - 1)\) are complements. A single strand is a sequence of one or more symbols in Σ. Two single strands form a *double strand* if and only if the single strands are complements of each other. A double strand with \(σ_i, ⃗σ_i\) is denoted by \(\frac{σ_i}{⃗σ_i}\). Note that two single strands form a double strand if subsequences of the two single strands are complements. For example, let \(α, β\) be two single strands such that \(α = σ₀σ₁\) and \(β = ⃗σ₁σ₀\). Then, the following two double strands is obtained with \(α, β\).

\[
\left[\begin{array}{c}
σ₀σ₁ \\
⃗σ₁σ₀
\end{array}\right], \left[\begin{array}{c}
σ₀σ₁ \\
⃗σ₁σ₀
\end{array}\right]
\]

The strands are stored in a *test tube*. The following expression denotes a test tube \(T₁\) which stores two single strands \(σ₀σ₁, ⃗σ₁σ₀\).

\(T₁ = \{σ₀σ₁, ⃗σ₁σ₀\}\)

Also note that we let the set of strands stored in a test tube be a *k-multiset*. That is, each strand has \(k\) copies in the test tube, where \(k\) depends on the lab error in DNA manipulations. We define each strand in a set represented with a test tube as one unit. For example, the following test tube \(T₂\) contains two units of \(σ₀σ₁\), and three units of \(⃗σ₁σ₀\). (In this paper, we use one unit of a strand only.)

\(T₂ = \{σ₀σ₁, ⃗σ₁σ₀, ⃗σ₀σ₁, ⃗σ₁σ₀\}\)

2.2. Abstract model for DNA computing

A number of theoretical or practical computational models have been proposed for DNA computing[1, 4, 5, 6, 9, 10, 11]. In this paper, we assume a theoretical computation model based on the RDNA model[11], which is an abstract mathematical model for the performance of parallel DNA computation. The model allows the following eight DNA manipulations, which are widely used in DNA computing.

1. **Merge**: Given two test tubes \(T₁, T₂\), \(\text{Merge}(T₁, T₂)\) stores the union \(T₁ ∪ T₂\) in \(T₁\).

2. **Copy**: Given a test tube \(T₁\), \(\text{Copy}(T₁, T₂)\) produces a test tube \(T₂\) with the same contents as \(T₁\).

3. **Detect**: Given a test tube \(T\), \(\text{Detect}(T)\) outputs “yes” if \(T\) contains at least one strand, otherwise outputs “no”.

4. **Separation**: Given a test tube \(T₁\) and a set of strings \(X\), \(\text{Separation}(T₁, X)\) removes all single strands containing a string \(X\) from \(T₁\), and produces a test tube \(T₂\) with the removed strands.

5. **Selection**: Given a test tube \(T₁\) and an integer \(L\), \(\text{Selection}(T₁, L)\) removes all strands whose length is \(L\) from \(T₁\), and produces a test tube \(T₂\) with the removed strands. (The length of a strand is the number of symbols in the strand.)

6. **Cleavage**: Given a test tube \(T\) and a string of two symbols \(σ₀σ₁\), \(\text{Cleavage}(T, σ₀σ₁)\) cuts each double strand containing \(\frac{σ₀σ₁}{σ₀σ₁}\) in \(T\) into two double strands as follows.

\[
\left[\begin{array}{c}
σ₀σ₁ \\
⃗σ₁σ₀
\end{array}\right] \Rightarrow \left[\begin{array}{c}
σ₀σ₁ \\
⃗σ₁σ₀
\end{array}\right], \left[\begin{array}{c}
σ₁β₀ \\
α₁σ₀
\end{array}\right]
\]

It is assumed that Cleavage can only be applied to some specified symbols over the alphabet Σ.

7. **Annealing***: Given a test tube \(T\), \(\text{Annealing}(T)\) produces all feasible double strands from single strands in \(T\). (The produced double strands are still stored in \(T\) after Annealing.)

8. **Denaturation**: Given a test tube \(T\), \(\text{Denaturation}(T)\) dissociates each double strand in \(T\) into two single strands.

In this paper, we use all of the manipulations except for Selection. Figure 1 shows examples for Separation, Cleavage, Annealing and Denaturation.

The above eight manipulations are implemented with a constant number of biological steps for DNA strands[8]. In this paper, we assume that the complexity of each manipulation is \(O(1)\) lab steps, and consider asymptotic complexity of a sequence of the manipulations.

*In some papers, an operation *Ligation* is defined to concatenate single strands after *Annealing*. We assume that *Annealing* includes *Ligation*, that is, *Ligation* is automatically executed after *Annealing*. 
Assignment represents one bit. Then, we consider a basic operation of binary numbers, where we use one single DNA strand to represent one bit. In the above alphabet, V = “0” if a value of the bit is 0, otherwise V = “1”.

3. Bit representation and value assignment

In this section, we first describe a DNA representation of binary numbers, where we use one single DNA strand to represent one bit. Then, we consider a basic operation Value Assignment which assigns values to every bit.

3.1. Bit representation

We describe the representation of n binary numbers of m bits. In the representation, one single strand corresponds to one bit of a binary number. Therefore, we use O(mn) single strands to denote n binary numbers.

We first define the alphabet Σ used in the representation as follows.

\[ \Sigma = \{ A_0, A_1, \ldots, A_{n-1}, B_0, B_1, \ldots, B_{m-1}, C_0, C_1, D_0, D_1, 1, 0, \theta, A, B, C, D, \theta, T, \theta \} \]

In the above alphabet, \(A_0, A_1, \ldots, A_{n-1}\) denote addresses of binary numbers, and \(B_0, B_1, \ldots, B_{m-1}\) denote bit positions in a binary number. \(C_0, C_1\) and \(D_0, D_1\) are the specified symbols cut by Cleavage, that is, Cleavage(\(T, C_0C_1\)) and Cleavage(\(T, D_0D_1\)) cut all double strands containing \(\left[ \begin{array}{c} C_0C_1 \\ C_1C_0 \end{array} \right]\) and \(\left[ \begin{array}{c} D_0D_1 \\ D_1D_0 \end{array} \right]\) in a test tube \(T\), respectively.

Symbols “0” and “1” are used to denote values of bits, and “θ” is a special symbol for Separation.

Using the above alphabet, a value of a bit, whose address and bit position are \(i\) and \(j\), is represented by a single strand \(S_{i,j}\) such that

\[ S_{i,j} = D_1 A_i B_j C_0 C_1 V D_0, \]

where \(V = “0”\) if a value of the bit is 0, otherwise \(V = “1”\).

We call each \(S_{i,j}\) a memory strand, and use a set of \(O(mn)\) different memory strands to denote \(n\) binary numbers of \(m\) bits. (To overcome the lab error, we assume that each memory strand has \(k\) copies in the test tube.)

3.2. Value assignment

An input of Value Assignment is a test tube \(T_{\text{input}}\) which contains memory strands such that

\[ T_{\text{input}} = \{ D_1 A_i B_j C_0 C_1 V_{i,j} D_0 | 0 \leq i \leq n-1, 0 \leq j \leq m-1 \}, \]

where \(V_{i,j} \in \{0,1\}\). An output is also a test tube \(T_{\text{output}}\) such that

\[ T_{\text{output}} = \{ D_1 A_i B_j C_0 C_1 V' D_0 | 0 \leq i \leq n-1, 0 \leq j \leq m-1 \}, \]

where \(V' \in \{0,1\}\). (Note that all memory strands are set to the same value \(V'\).)

The procedure for Value Assignment consists of two steps. The first step is deletion of values from memory strands, and the second step is assignment of values to the strands. In the first and second steps, we use auxiliary test tubes \(T_{\theta}\) and \(T_{V'}\) such that

\[ T_{\theta} = \{ C_0 C_1 \}, \quad T_{V'} = \{ C_1 V' D_0, \overline{C_0 C_1} \}. \]

Since DNA strands in tubes \(T_{\theta}\) and \(T_{V'}\) are independent of addresses or values of memory strands, we can prepare them in advance. In addition, we use a test tube \(T_{\text{tmp}}\) to store the strands temporarily. A detailed description of the procedure is presented below.

Procedure ValueAssignment(\(T_{\text{input}}, T_{\text{output}}\))

Step 1: Delete values from memory strands.

\[ \text{Merge}(T_{\text{input}}, T_{\theta}) \]

\[ \text{Annealing}(T_{\text{input}}) \]

\[ \text{Cleavage}(T_{\text{input}}, C_0 C_1) \]

\[ \text{Denaturation}(T_{\text{input}}) \]

\[ \text{Separation}(T_{\text{input}}, \{ C_1, \overline{C_0}, \overline{C_1} \}, T_{\text{tmp}}) \]

Step 2: Assign values to memory strands.

\[ \text{Merge}(T_{\text{input}}, T_{V'}) \]

\[ \text{Annealing}(T_{\text{input}}) \]

\[ \text{Denaturation}(T_{\text{input}}) \]

\[ \text{Separation}(T_{\text{input}}, \{ C_0 C_1 \}, T_{\text{output}}) \]

We illustrate an execution of the above procedure. To simplify the illustration, we assume that an input tube contains only one memory strand, that is, \(T_{\text{input}} = \)
\[ \begin{array}{c|cc|cc}
V_{\text{in1}} & V_{\text{in2}} & V_{\text{out1}} & V_{\text{out2}} \\
0 & 0 & \alpha_{00} & \beta_{00} \\
0 & 1 & \alpha_{01} & \beta_{01} \\
1 & 0 & \alpha_{10} & \beta_{10} \\
1 & 1 & \alpha_{11} & \beta_{11} \\
\end{array} \]

Figure 2. A truth table (where \( \alpha_{ij}, \beta_{ij} \in \{0, 1\} \)).

\[ \{ D_1 A_i B_j C_0 C_1 V D_0 \} \]. After Merge and Annealing operations in Step 1, a test tube \( T_{\text{input}} \) is given by

\[ T_{\text{input}} = \left\{ D_1 A_i B_j C_0 C_1 \frac{1}{V} D_0 \right\}, \]

and after Step 1, a test tube \( T_{\text{input}} \) becomes as follows.

\[ T_{\text{input}} = \{ D_1 A_i B_j C_0 \} \]

In Step 2, after Merge and Annealing operations, a test tube \( T_{\text{input}} \) is given by

\[ T_{\text{input}} = \left\{ D_1 A_i B_j C_0 C_1 V^2 D_0 \right\}, \]

then, an output tube of the procedure is as follows.

\[ T_{\text{output}} = \{ D_1 A_i B_j C_0 C_1 V^2 D_0 \} \]

The complexity of the above procedure is \( O(1) \) lab steps because only a constant number of DNA manipulations are executed for memory strands in parallel. The DNA strands used in the procedure is \( O(nm) \) memory strands and \( O(1) \) kinds of \( O(mn) \) auxiliary strands in \( T_\text{w} \) and \( T_\text{v} \). Then, we obtain the following lemma.

**Lemma 1** Value Assignment for \( O(mn) \) memory strands can be executed in \( O(1) \) lab steps using \( O(1) \) kinds of \( O(mn) \) additional DNA strands. \( \Box \)

### 4. Procedure for logic operations

In this section, we show a procedure which computes logic operations for pairs of two memory strands in parallel. Let us consider a logic operation whose inputs and outputs are Boolean values \( V_{\text{in1}}, V_{\text{in2}} \) and \( V_{\text{out1}}, V_{\text{out2}} \), respectively, and the values are defined by the truth table in Figure 2.

Also let the following test tube \( T_{\text{input}} \) contain two memory strands whose values are \( V_{i,j} \) and \( V_{g,h} \) as an input.

\[ T_{\text{input}} = \{ D_1 A_i B_j C_0 C_1 V_{i,j} D_0, D_1 A_g B_h C_0 C_1 V_{g,h} D_0 \} \]

Then, an output of the procedure, for the logic operation defined in Figure 2, is a test tube \( T_{\text{output}} \) given below.

\[ T_{\text{output}} = \left\{ \begin{array}{ll}
D_1 A_i B_j C_0 C_1 \alpha_{00} D_0, & D_1 A_g B_h C_0 C_1 \beta_{00} D_0 \\
D_1 A_i B_j C_0 C_1 \alpha_{01} D_0, & D_1 A_g B_h C_0 C_1 \beta_{01} D_0 \\
D_1 A_i B_j C_0 C_1 \alpha_{10} D_0, & D_1 A_g B_h C_0 C_1 \beta_{10} D_0 \\
D_1 A_i B_j C_0 C_1 \alpha_{11} D_0, & D_1 A_g B_h C_0 C_1 \beta_{11} D_0 \\
\end{array} \right. \]

In the following subsections, we first describe single strands which define a logic operation, and next show an overview and details of the procedure.

#### 4.1. Single strands for logic operations

Four single strands are used to define a logic operation. We call the four strands *logic strands*, and each of them corresponds to each row of a truth table. We assume that \( S_{i,j}(V) \) is a memory strand whose value is \( V (\in \{0, 1\}) \), that is, \( S_{i,j}(V) = D_1 A_i B_j C_0 C_1 V D_0 \). Then, we consider a logic operation given by the truth table in Figure 2. We define the set of logic strands \( L_{i,j,g,h} \) for a pair of two memory strands \( S_{i,j} = D_1 A_i B_j C_0 C_1 V_{i,j} D_0, S_{g,h} = D_1 A_g B_h C_0 C_1 V_{g,h} D_0 \) as follows.

\[ L_{i,j,g,h} = \left\{ \begin{array}{ll}
\alpha_{00} D_0 S_{i,j}(0) S_{g,h}(0) D_1 \beta_{00}, & \\
\alpha_{01} D_0 S_{i,j}(0) S_{g,h}(1) D_1 \beta_{01}, & \\
\alpha_{10} D_0 S_{i,j}(1) S_{g,h}(0) D_1 \beta_{10}, & \\
\alpha_{11} D_0 S_{i,j}(1) S_{g,h}(1) D_1 \beta_{11}, & \\
\end{array} \right. \]

In the above definition, each single strand in \( L_{i,j,g,h} \) consists of a complemental chain of two input memory strands, and two strings in a set of single strands \( \{ \alpha_{00} D_0, \alpha_{01} D_0, \alpha_{10} D_0, \alpha_{11} D_0, D_1 \beta_{00}, D_1 \beta_{01}, D_1 \beta_{10}, D_1 \beta_{11} \} \), which is a set of auxiliary strands to separate memory strands according to their values.

Figure 3 shows an example of logic strands which denote the AND operation such that \( V_{\text{out1}} = V_{\text{in1}} \land V_{\text{in2}} \) and \( V_{\text{out2}} = V_{\text{in2}} \).

#### 4.2. Overview of the procedure

The procedure consists of the following 3 steps.

1. Divide memory strands into two test tubes \( T_0 \) and \( T_1 \) according to outputs of the operation. Memory strands whose output values are 0 are stored in \( T_0 \), and the others are stored in \( T_1 \).
2. Assign values to memory strands in each of test tubes $T_0, T_1$.

3. Merge two test tubes.

The second and third steps are easily executed using Value assignment and Merge, respectively. We describe outline of the first step in the following.

To simplify the description, we assume that input tube $T_{input}$ contains two memory strands $S_{i,j}, S_{g,h}$, whose values are $X, Y (\in \{0, 1\})$, and their corresponding logic strands, that is, $T_{input} = \{S_{i,j}(X), S_{g,h}(Y)\} \cup L_{i,j,g,h}$. The first step consists of three substeps. In the first substep, two memory strands are connected according to one of complementary logic strands. Then, the connected memory strands are separated to a temporal test tube. The substep is executed using mainly Annealing, Denaturation and Separation as follows. (Single or double strands which do not include a memory strand are omitted in the description.)

$$\{S_{i,j}(X), S_{g,h}(Y)\} \cup L_{i,j,g,h}$$

$\Rightarrow$ (Annealing) $\Rightarrow$

$$\left\{ \frac{S_{i,j}(X)S_{g,h}(Y)}{\alpha_{xy}D_{0}S_{i,j}(X)S_{g,h}(Y)D_{1}\beta_{xy}} \right\}$$

$\Rightarrow$ (Denaturation & Separation) $\Rightarrow$

$$\{S_{i,j}(X)S_{g,h}(Y)\} \rightarrow T_{tmp}$$

In the second substep, two memory strands are separated according to their output of the operation. Each connected memory strand is annealed with a complementary logic strand again. Each auxiliary string in the complementary logic strand denotes an output value of the annealed memory strand. The annealed double strand is cut into two double strand so that each double strand contain one memory strand. Then, an auxiliary string are added to each memory strand, and separated into one of two test tubes according to its auxiliary string. After this substep, a memory strand whose output value is $\alpha \in \{0, 1\}$ is stored in a test tube $T_{a}$ with its auxiliary string. The substep is executed using mainly Annealing, Cleavage, Denaturation and Separation as follows.

$$\{S_{i,j}(X)S_{g,h}(Y)\} \cup L_{i,j,g,h}$$

$\Rightarrow$ (Annealing) $\Rightarrow$

$$\left\{ \frac{S_{i,j}(X)S_{g,h}(Y)}{\alpha_{xy}D_{0}S_{i,j}(X)S_{g,h}(Y)D_{1}\beta_{xy}} \right\}$$

$\Rightarrow$ (Cleavage) $\Rightarrow$

$$\left\{ \frac{S_{i,j}(X)}{\alpha_{xy}D_{0}S_{i,j}(X)} \right\} \left\{ \frac{S_{g,h}(Y)}{\alpha_{xy}D_{0}S_{g,h}(Y)} \right\}$$

$\Rightarrow$ (Denaturation) $\Rightarrow$

$$\{\alpha_{xy}D_{0}S_{i,j}(X), S_{g,h}(Y)D_{1}\beta_{xy}\}$$

$\Rightarrow$ (Separation) $\Rightarrow$

$$\alpha_{xy}D_{0}S_{i,j}(X) \rightarrow T_{0} \quad (\text{if } \alpha_{xy} = 0)$$

$$T_{1} \quad (\text{otherwise})$$

$$S_{g,h}(Y)D_{1}\beta_{xy} \rightarrow T_{0} \quad (\text{if } \beta_{xy} = 0)$$

$$T_{1} \quad (\text{otherwise})$$

In the third substep, an auxiliary string is cut after annealing, and removed from each test tube. The substep is executed using mainly Annealing, Cleavage and Denaturation as follows. (In case of $\alpha_{xy} = \beta_{xy}$)

$$T_{0} = \{ \frac{\alpha_{xy}D_{0}S_{i,j}(X)}{S_{g,h}(Y)D_{1}\beta_{xy}} \}$$

$\Rightarrow$ (Annealing) $\Rightarrow$

$$\left\{ \frac{\alpha_{xy}D_{0}S_{i,j}(X)}{\alpha_{xy}D_{0}S_{i,j}(X)} \right\} \left\{ \frac{S_{g,h}(X)D_{1}\beta_{xy}}{S_{g,h}(Y)D_{1}\beta_{xy}} \right\}$$

$\Rightarrow$ (Cleavage) $\Rightarrow$
\[
\left\{ \frac{S_{i,j}(X)}{S_{i,j}(X)} \right\}, \left\{ \frac{S_{g,h}(Y)}{S_{g,h}(Y)} \right\}
\]
\Rightarrow (Denaturation) \Rightarrow
\]
\[T_{\alpha \gamma} = \{ S_{i,j}(X), S_{g,h}(Y) \} \]

(In case of \( \alpha \gamma \neq \beta \gamma \))
\[
\begin{align*}
T_{\alpha \gamma} &= \{ \alpha \gamma \frac{D_{0}S_{i,j}(X)}{D_{0}S_{i,j}(X)} \}, \\
T_{\beta \gamma} &= \{ S_{g,h}(Y)D_{1}\frac{\beta \gamma}{\gamma \beta} \}
\end{align*}
\Rightarrow (Annealing) \Rightarrow
\]
\[
\left\{ \frac{\alpha \gamma D_{0}S_{i,j}(X)}{\alpha \gamma D_{0}S_{i,j}(X)} \right\}, \left\{ \frac{S_{g,h}(Y)D_{1}\beta \gamma}{S_{g,h}(Y)D_{1}\beta \gamma} \right\}
\]
\Rightarrow (Cleaveage) \Rightarrow
\]
\[
\left\{ \frac{S_{i,j}(X)}{S_{i,j}(X)} \right\}, \left\{ \frac{S_{g,h}(Y)}{S_{g,h}(Y)} \right\}
\]
\Rightarrow (Denaturation) \Rightarrow
\]
\[T_{\alpha \gamma} = \{ S_{i,j}(X) \}, T_{\beta \gamma} = \{ S_{g,h}(Y) \} \]

### 4.3. Details of the procedure

We summarize details of the procedure for a logic operation in the following. \( T_{\text{input}} \) and \( T_{\text{output}} \) are test tubes which contain input and output memory strands, respectively. We assume \( O(mn) \) memory strands are stored in \( T_{\text{input}} \). In addition, \( T_{L} \) is a test tube which contains \( O(mn) \) logic strands. Test tubes \( T_{0}, T_{1}, T_{0}', T_{1}' \) and \( T_{\text{tmp}} \) are used as temporal storage.

**Procedure** Logic Operation (Test Tube \( T_{\text{input}} \), Test Tube \( T_{L} \), Test Tube \( T_{\text{output}} \))

**Step 1:** Divide memory strands into two test tubes \( T_{0} \) and \( T_{1} \) according to outputs of the operation for a pair of two memory strands.

1.1) Connect and select each pair of memory strands according to their logic strands.

\begin{align*}
\text{Merge}(T_{\text{input}}, T_{L}) \\
\text{Annealing}(T_{\text{input}}) \\
\text{Denaturation}(T_{\text{input}}) \\
\text{Separation}(T_{\text{input}}, \{ D_{0}D_{1}, D_{0}D_{1}' \}, T_{\text{tmp}})
\end{align*}

(1-2) Separate memory strands according to outputs of operations.

Annealing(\( T_{\text{tmp}} \))
Cleaveage(\( T_{\text{tmp}}, D_{0}D_{1} \))
Merge(\( T_{\text{tmp}}, \{ 0 \} D_{0}D_{1}, D_{1} \) \( \overrightarrow{0} \))

Denaturation(\( T_{\text{tmp}} \))
Separation(\( T_{\text{tmp}}, \{ 0 \}, T_{0} \))
Separation(\( T_{\text{tmp}}, \{ \overrightarrow{0} \}, T_{1} \))

**Step 2:** Assign values to memory strands in each of test tubes \( T_{0}, T_{1} \).

\begin{align*}
\text{Value Assignment}_{\downarrow}(T_{0}, T_{0}') \\
\text{Value Assignment}_{\downarrow}(T_{1}, T_{1}')
\end{align*}

**Step 3:** Merge output test tubes.

\begin{align*}
\text{Copy}(T_{0}', T_{\text{output}}) \\
\text{Merge}(T_{\text{output}}, T_{1}')
\end{align*}

(1-3) Cut and remove auxiliary strings from each strand.

Annealing(\( T_{0} \), Annealing(\( T_{1} \))
Cleaveage(\( T_{0}, D_{0}D_{1} \), Cleaveage(\( T_{1}, D_{0}D_{1} \)
Denaturation(\( T_{0} \), Denaturation(\( T_{1} \))
Separation(\( T_{0}, \{ \overrightarrow{0}, C_{0}C_{1} \}, T_{\text{tmp}} \),
Separation(\( T_{1}, \{ \overrightarrow{0}, C_{0}C_{1} \}, T_{\text{tmp}} \))

The procedure consists of a constant number of DNA manipulations except for Step 2, and Step 2 can be executed in \( O(1) \) lab steps from Lemma 1. Since the number of kinds of DNA strands used as memory or logic strands is \( O(mn) \), we obtain the following theorem.

**Theorem 1** Logic operations for \( O(mn) \) memory strands can be executed in \( O(1) \) lab steps using \( O(mn) \) different additional DNA strands.

Since the above procedure is applicable to any pair of memory strands using a logic strand, we can execute different operations for pairs of memory strands in parallel. In addition, the other simple operations, such as NOT, SHIFT or COPY, are also executed with the same complexity.

### 5 Procedure for arithmetic operations

In this section, we show a procedure to add two binary numbers represented by memory strands. Since a basic idea of the procedure is similar to [5], we give a brief explanation only.

We consider addition of two binary numbers, \( a_{m-1}a_{m-2}\ldots a_{0} \) and \( b_{m-1}b_{m-2}\ldots b_{0} \), which represent two numbers \( a \) and \( b \) such that \( a = \sum_{j=0}^{m-1} a_{j} \cdot 2^{j} \) and \( b = \sum_{j=0}^{m-1} b_{j} \cdot 2^{j} \). We assume the two numbers \( a \) and \( b \) are stored in two sets of memory strands \( \{ S_{i_{m-1}, m-1}, S_{i_{m-2}, m-2}, \ldots, S_{i_{0}, 0} \} \) and \( \{ S_{i_{m-1}, m-1}, S_{i_{m-2}, m-2}, \ldots, S_{i_{0}, 0} \} \), respectively. We also assume \( a_{m-1} = b_{m-1} = 0 \) to simplify the description.
Then, the sum $s_{m-1}s_{m-2}\ldots s_0$ of two numbers $a, b$ is obtained using a procedure consisting of the following four steps. (Binary operators $\oplus$ and $\land$ are XOR and AND operations, respectively.)

1. For each $j \ (0 \leq j \leq m-1)$, compute $x_j = a_j \oplus b_j$, and $y_j = a_j \land b_j$.

   (This step denotes behavior of a half adder. $x_j$ and $y_j$ denote a sum and a carry bit obtained from $j$-th bits, respectively.)

2. For each $j \ (0 \leq j \leq m-1)$, compute $p_j = x_j \land \neg y_j$.

   ($p_j = 1$ in case that the $j$-th bit propagates a carry in the $(j-1)$-th bit to the $(j+1)$-th bit.)

3. For each $j \ (1 \leq j \leq m-1)$, set $c_j = 1$ if $y_{j-1} = 1$ or there exists $k \ (< j)$ such that $p_{j-1} = p_{j-2} = \ldots = p_{k+1} = 1$ and $y_k = 1$, otherwise set $c_j = 0$.

4. For each $j \ (1 \leq j \leq m-1)$, set $s_j = x_j \oplus c_j$.

The first, second and fourth steps of the above procedure are easily implemented using a constant number of DNA manipulations and logic procedures described in Section 4. Thus, we discuss implementation of the third step in the followings.

Let $y_j$ and $p_j$ be binary values stored in memory strands $S_{y,j}$ and $S_{p,j}$ for each $j \ (0 \leq j \leq m-1)$. We first generate the following single strand $C_j$ for each $j \ (1 \leq j \leq m-1)$. ($C_j$ includes a memory strand for $c_j$ whose value is 0, that is, $S_{c,j}(0)$.)

$$C_j = D_1A_cB_jC_10D_0D_1A_yB_j\not= S_{c,j}(0)D_1A_yB_j\not= $$

Next, we generate the following two single strands $Y_j(1)$ and $P_j(1)$ if $y_j = 1$ and $p_j = 1$ for each $j \ (0 \leq j \leq m-1)$, respectively.

$$Y_j(1) = \not= A_yB_j\not= 1, \quad P_j(1) = \not= A_yB_jA_yB_j\not= $$

Finally, we generate a single strand $Q_{j\leftarrow j-1}$ for each $j \ (1 \leq j \leq m-1)$ to propagate a carry.

$$Q_{j\leftarrow j-1} = A_yB_j\not= A_yB_{j-1}$$

The above single strands are stored in a test tube and annealed. We first consider the case that $y_{j-1} = 1$. We obtain the following double strand for $c_j$.

$$\begin{bmatrix} C_j & Y_{j-1}(1) \\ Q_{j\leftarrow j-1} & \end{bmatrix}$$

We next consider the case that $p_{j-1} = p_{j-2} = \ldots = p_{k+1} = 1$ and $y_k = 1$. Single strands $Q_{j\leftarrow j-1}, Q_{j-1\leftarrow j-2}, \ldots, Q_{k+1\leftarrow k}$ are used to propagate a carry stored in $Y_k(1)$, and we obtain the following double strand for $c_j$.

$$\begin{bmatrix} C_j & P_{j-1}(1) & P_{j-2}(1) & \ldots \\ Q_{j\leftarrow j-1} & Q_{j-1\leftarrow j-2} & \ldots & \end{bmatrix}$$

After Denaturation for the test tube, a single strand containing a string "#1" is separated into another test tube $T_1$ using Separation. (In the above generated single strands, only $Y_k(1)$ contains the string.) The other single strands containing a memory strand is also separated in test tube $T_0$.

Memory strands in separated single strands are cut and extracted by a similar method to Value assignment. Therefore, we obtain two test tubes: one contains memory strands set to 0, and the other contains memory strands set to 1. Using Value assignment and Merge, the third step is completed.

Since the whole procedure consists of a constant number of DNA manipulations and uses $O(mn)$ kinds of DNA strands, we obtain the following theorem.

**Theorem 2** Additions for $O(n)$ pairs of $m$-bit binary numbers can be executed in $O(1)$ lab steps using $O(mn)$ different additional DNA strands. 

The procedure for additions are easily modified to be applied to subtractions. Thus, we can compute subtractions with the same complexity using DNA strands.

6. Conclusions

In this paper, we proposed two procedures for logic and arithmetic operations using DNA strands. Both procedure works in $O(1)$ lab steps using $O(mn)$ different additional DNA strands. Features of our procedures are that different logic or arithmetic operations are executed in parallel because addresses and operations are denoted with single strands. Thus, our procedure archives general parallel processing for DNA strands.

Although our results are based on a theoretical model, they can be implemented practically since every DNA manipulation used in the model has been already realized in lab level. In addition, arithmetic operations are so primitive that we believe that our results will play an important role in the future DNA computing.
References


