Implementation of Turing Machine Using DNA Strand Displacement

Wataru Yahiro(✉) and Masami Hagiya

Department of Computer Science,
Graduate School of Information Science and Technology, University of Tokyo, 7-3-1,
Bunkyo-ku, Hongo, Tokyo 113-8656, Japan
{yahiro.wataru,hagiya}@is.s.u-tokyo.ac.jp

Abstract. The computational capability of biochemical systems is one of the major interest in the area of nanotechnology. Since Bennett proposed his thought experiment of chemical Turing machine using DNA-like molecules, many attempts for DNA Turing machine have been made. However, they are based on some hypothetical assumptions or require laboratory manipulations for each step. Here we propose an implementation of Turing machine by using DNA strand displacement cascades.

1 Introduction

The development of molecular biology revealed that activities in biological cells are carried out in a highly mechanical way. The research area of molecular computing was born inspired by the insight into such biochemical processes. Researchers of molecular computing have been studying for long years the way to perform computation by biomolecules.

The attempts to implement Turing machine by biomolecules have a long history. The first theoretical proposal was done by Bennett [2]. However, because it required hypothetical enzymes and polymers, it was only a theoretical consideration. Many other theoretical proposals used existing enzymes, but they required laboratory manipulations for each step of Turing machine [1,6].

Recently, DNA strand displacement (DSD) is widely known as a powerful framework for molecular computation. It was shown that multi-stack machines, which are Turing-universal model of computation, can be constructed within this framework [3,4]. In these works, a polymer of DNA molecules was used to resemble a stack and stack operations were implemented as polymer modification reactions which push and pop end monomers.

We take a step further and show a way to directly construct Turing machine using DSD. In our model, computation is driven by two types of formal chemical reactions implemented by DSD. The first type is for state transitions. They push forward the computation by consuming a character and a machine state, then generating a new character and a next machine state. The second type is for tape modification. A tape is implemented as a long DNA molecule, and it is designed to be modified easily, so tape modification reactions can be realized

C. Martín-Vide et al. (Eds.): TPNC 2016, LNCS 10071, pp. 161–172, 2016.
DOI: 10.1007/978-3-319-49001-4_13

by simple strand displacement reactions. In addition, tapes are autonomously extended by adding domains to which extension tapes can be attached, and it enables Turing-universal computation.

Our construction is different from works on DNA Turing machines such as those using restriction enzymes in that it does not require laboratory manipulations during the computation. Compared to the construction of multi-stack machines, our tapes can grow toward both sides if needed and it can be modified not only at the end but also at any part.

This paper mainly consists of two parts. First, in Sect. 2, we consider the case where the tape length is fixed, and show that an arbitrary space-bounded Turing machine can be simulated by DSD. Second, in Sect. 3, we introduce reactions for signal-driven tape extending, which enable Turing-universal computation.

In this paper, we assume that readers are familiar with the basic rules of DNA strand displacement [8].

2 Space-Bounded DSD Turing Machine

2.1 Definitions

We have to modify the definition of well-known Turing machine so that it is suitable to the framework of DSD. Our basic idea is roughly the same as that of stack machines [3,4]. The configuration of a machine is represented by a chemical solution that contains three types of special chemical species, which stand for states, characters and tapes, respectively, and the computation is driven by two types of chemical reactions, which are in charge of state transitions and tape operations respectively.

We clarify the basic notions of formal language theory used in this paper. Let Σ be a finite alphabet. The set of all finite-length strings over Σ is denoted by Σ^*. Let U, V be languages. The concatenation of languages is denoted by UV. For a character $a \in \Sigma$, a single-word language $\{a\}$ is often denoted by a itself. Next we define a labeled alphabet. Let D be a set of symbols called labels. A labeled alphabet is an alphabet $\Sigma^D = \{a^d \mid a \in \Sigma, d \in D\}$. We denote a subset of Σ^D that have a fixed label by $\Sigma^d = \{a^d \mid a \in \Sigma\}$ where $d \in D$.

Let us give the formal definition of a variant of Turing machine, which we call DSD Turing machine or DSDTM for its short hand.

Definition 1. *let $D = \{L, R\}$ be a set of labels. A DSD Turing machine M is defined as a 6-tuple $M = (Q, \Sigma, \delta, q_0, F, a_0^d)$ where*

- *Q is a finite set of states,*
- *Σ is a finite set of symbols called alphabet,*
- *$\delta \subseteq (Q \backslash F) \times \Sigma^D \times Q \times \Sigma^D$ is a set of 4-tuples called state transition reactions,*
- *$q_0 \in Q$ is the initial state,*
- *$F \subseteq Q$ is a set of final states,*
- *$a_0^d \in \Sigma^D$ is an initial character.*

We call elements of Σ^L and Σ^R, respectively, L-type characters and R-type characters.

Definition 2. *A tape of a DSD Turing machine M is defined as a string $T \in (\Sigma^L)^*(\Sigma^R)^*$. A head is defined as a partition (boundary line) between L-type characters and R-Type characters.*

In the following text, a tape is denoted as $T = [a_1 a_2 ... a_n | a_{n+1} ... a_{n+m}]$, where the vertical bar | represents the position of the head. Note that the "head" is defined as a partition while that of ordinary Turing machines is defined as a pointer to a character.

Definition 3. *A configuration of a DSD Turing machine M is defined as a 3-tuple $C = (q, a^d, T)$ where $q \in Q$ is a state, $a^d \in \Sigma^D$ is an extra character and $T \in (\Sigma^L)^*(\Sigma^R)^*$ is a tape.*

A configuration (q_0, a_0^d, T_0) is especially called an initial configuration, where T_0 is an initial tape that is given as an input. Likewise, when a configuration contains a state $q_f \in F$, it is called a terminal configuration. This definition is a formalization of a chemical solution in which q, x^d and T represent formal chemical species. Chemical solutions are often formalized as a multiset [3,4,7], but we use a tuple in order to clarify that the solution contains exactly one molecule of each species. This situation is exactly the same as in the previous works on stack machines [3,4].

A state transition rule of a DSDTM is represented as a formal reaction $(q, a^d, p, x^{d'}) \in \delta$. It is called a state transition reaction. A state transition reaction is a formal chemical reaction, as its name indicates, so we denote it by $q + a^d \to p + x^{d'}$ in the following text. According to the combination of labels, there are four types of state transition reactions as written below.

$$q + a^L \to p + x^L \tag{1}$$
$$q + a^L \to p + x^R \tag{2}$$
$$q + a^R \to p + x^L \tag{3}$$
$$q + a^R \to p + x^R \tag{4}$$

These are called an LL-type reaction (1), an LR-type reaction (2), an RL-type reaction (3) and an RR-type reaction (4), respectively. We often write a LD-type reaction or a DR-type reaction to partially fix labels.

The operations for a tape are also defined as formal chemical reactions. They are called tape modification reactions and written as follows.

$$a^L + [...c|bd...] \rightleftharpoons [...ca|d...] + b^R \tag{5}$$

Tape modification reactions edit the content of the tape and move the head simultaneously. The forward reaction pushes an L-type character into left of the head and pops an R-type character which was at right of the head. The backward reaction is a mere inverse of the forward reaction. The only requirement for tape

modification reactions is that the content of the opposite side of the pushed character is not empty. For example, tape modification reaction is not defined between an extra character a^L and a tape $[bc|]$ because there is no character to pop. Conversely, except for this corner case, tape modification reactions occur whenever an extra character and a tape exist. Note that this limitation implies that tape modification reactions cannot change the length of the tape.

Now we describe how the computational process of DSDTM proceeds. A DSDTM performs computation as rewriting its configuration by state transition reactions and tape modification reactions. The transition rules between configurations are defined as follows.

Definition 4. *Let C and C' be configurations of a DSD Turing machine M. The transition from C to C' is denoted by $C \Rightarrow C'$. A transition from $C = (q, a^d, [a_1...a_n|a_{n+1}...a_{n+m}])$ is enabled when the destination fulfills one of the following conditions.*

- $C \Rightarrow (p, x^{d'}, [a_1...a_n|a_{n+1}...a_{n+m}])$ *when* $q + a^d \to p + x^{d'} \in \delta$.
- $C \Rightarrow (q, a^R_{n+1}, [a_1...a_n a|a_{n+2}...a_{n+m}])$ *when* $a^d \in \Sigma^L$ *and* $m > 0$.
- $C \Rightarrow (q, a^L_n, [a_1...a_{n-1}|a a_{n+1}...a_{n+m}])$ *when* $a^d \in \Sigma^R$ *and* $n > 0$.

The first one is a transition by a state transition reaction and others are by tape modification reactions.

Transitions of a DSDTM are essentially nondeterministic, but we can make it "practically" deterministic by imposing two conditions on δ. First, δ must be (partially) functional. That is to say, for any q and a^d, there exists at most one state transition reaction $q + a^d \to p + x^{d'} \in \delta$. Second, any state q can react with characters of only one type. When both a state transition reaction and a tape modification reaction is simultaneously possible, q can react with different characters and it might make the computation nondeterministic. For example, let a configuration be $(q, x^R, [a|b])$ and δ include two state transition reactions $q+a^L \to p+c^R$ and $q+x^R \to p'+d^L$. If a tape modification reaction occurs first, the transition path will be $([a|b], q, x^R) \Leftrightarrow ([|xb], q, a^L) \Rightarrow ([|xb], p, c^R)$. On the other hand, if a state transition reaction occurs first, it will be $([a|b], q, x^R) \Rightarrow ([a|b], p', d^L) \Leftrightarrow ([ad|], p', b^R)$, so the path will be forked. However, any transition by tape modification reactions is reversible and a tape modification reaction always emits a character whose label is opposite to its reactant, so we can avoid undesired nondeterminism by limiting the type of characters to react with each state. Accordingly, we can guarantee that a computational path of DSDTM can be uniquely determined except for going back by tape modifications. This is why we said that "practically" deterministic at the beginning of this paragraph.

2.2 DSD Turing Machine vs. Turing Machine

Since DSDTMs have rules for state transition and tape modification, they seem equivalent to Turing machine in its computational power. So, we prove the theorem that DSDTM can simulate an arbitrary space-bounded Turing machine.

Theorem 5. *Let a space-bounded Turing machine be* $M_{TM} = (Q, \Sigma, \delta, q_0, F)$, *where* Q *is a set of state,* Σ *is a finite alphabet,* $\delta : (Q \backslash F) \times \Sigma \to Q \times \Sigma \times \{L, R\}$ *is a transition function,* $q_0 \in Q$ *is the initial state and* $F \subseteq Q$ *is the set of final states. There exists a space-bounded DSD Turing machine* M *that can simulate* M_{TM}.

Proof. Here we describe the main ideas of the proof. First, we regard a character at right of the head of a DSDTM as the head of a Turing machine. For example, in a tape of a DSDTM $[ab|cd]$, the character c is assumed to be pointed by Turing machine's head. In this proof, we refer to the head of a Turing machine simply by the head, and that of a DSDTM by the partition in order to avoid overlapping of the word "head". Second, we introduce additional character used for tape manipulation. Finally, we translate each transition rule into state transition reactions by subdividing each step of M_{TM} into some substeps.

Specifically, we can construct a desired DSDTM as follows. Let the DSDTM be $M = (Q', \Sigma \cup \{H\}, \delta', q_0, F, H^L)$ where $H \notin \Sigma$ is an additional character and $Q' \supseteq Q$ has some additional states, while q_0 and F are directly inherited from M_{TM}. So, all that is left is to translate transition rules.

Let $\delta(q, a) = (p, x, d)$ be a transition rule. Because our assumption about the head is asymmetric, the implementation varies according to direction d. They can be implemented as follows, respectively.
$\delta(q, a) = (p, x, L)$:

$$q + a^R \to q_a + x^R \tag{6}$$
$$q_a + H^L \to q'_a + H^R \tag{7}$$
$$q'_a + b^L \to q_{ab} + H^L \tag{8}$$
$$q_{ab} + H^R \to p + b^R \tag{9}$$

where (8) is defined for every $b^L \in \Sigma^L$.
$\delta(q, a) = (p, x, R)$:

$$q + a^R \to q_a + H^R \tag{10}$$
$$q_a + H^L \to q'_a + x^L \tag{11}$$
$$q'_a + H^R \to p + H^L \tag{12}$$

Figure 1 shows how each substep works. First, no matter what the direction is, each cycle starts with a configuration $C = (q, H^L, [...b|a...])$ where a tape $[...b|a...] \in (\Sigma^L)^*(\Sigma^R)^*$ does not include either H^L or H^R. The only possible transition from C is $C \Leftrightarrow (q, a^R, [...bH|...])$. This step corresponds to reading a character on the head of Turing machine.

From this point, the computational path branches according to the direction indicated by the original transition function. In the case of $\delta(q, a) = (p, x, L)$, the reaction (6) and succeeding tape modification reaction rewrite the character on the head. The reaction (7) is to read the character on left of the partition, because we have to switch its direction label in order to move the head toward left.

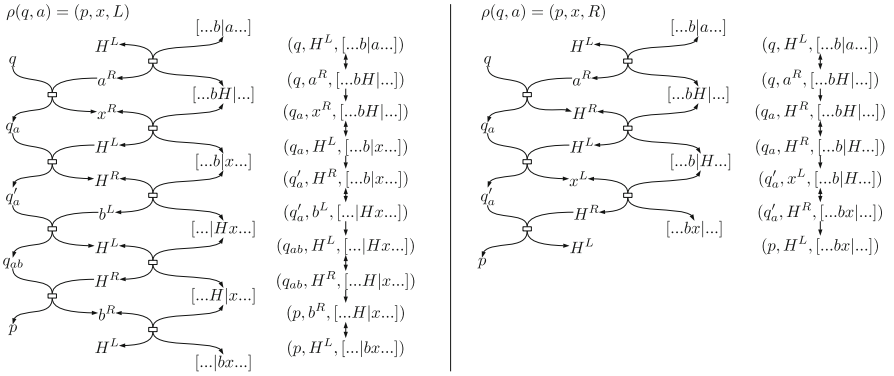

Fig. 1. A simulation of a Turing machine by a DSDTM

Fig. 2. The structure of formal species q, a^L, a^R and $[a...b|c...d]$

The reaction (8) recognizes the character and the next reaction (9) produces a switched character and the new state p. After the final substep, the configuration will be $(p, b^R, [...H|x...])$, and it can change to $(p, H^L, [...|bx...])$, which is the entry point of the next cycle.

The translation of $\delta(q, a) = (p, x, R)$ is slightly easier, because we only have to care about the character on the head. The first reaction (10) is to recognize the character on the tape. The second reaction (11) produces a new character x^L to be written. The final reaction (12) produces new basic state p, then the configuration reaches the entry point of next cycle.

Accordingly, it is proved that for any Turing machine M_{TM}, there exists an DSDTM M that can simulate M_{TM}. Moreover, M needs only a constant number of substeps to simulate each step of M_{TM}, so DSDTM can perform computation as fast as Turing machine.

2.3 Formal Species

Figure 2 illustrates the structure of formal species q, a^L, a^R and $[a...b|c...d]$. A state q and characters a^L and a^R are represented by short upper strands, which we call state strands and character strands. The structure of state strands is inherited from [4]. The two long domains ${}^-q$ and ${}^+q$ are unique to q, but all state strands have a toehold domain U in common. The structure of character strands a^L and a^R are like mirror images. The domains S and T are shared among all character strands, so each character strand has only one unique domain.

A tape $[a...b|c...d]$ is a long complex of DNA molecules. The bottom strand of it, which we name a substrate strand, is a single long strand. The substrate strand

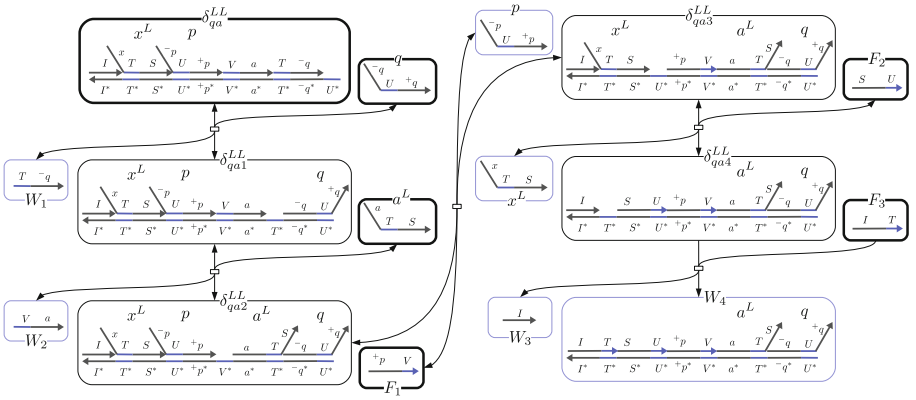

Fig. 3. The implementation of a state transition reaction $q + a^L \rightarrow p + x^L$. The bold frames indicate species required to be present initially. The blue frames indicate end products.

has a long domain S^* and a toehold domain T^* alternatively and repeatedly, and has T^* at the both ends. The content of the tape is represented by character strands attached to the substrate strand. Though almost all domains of the substrate strands are covered with character strands, only one toehold T^* is exposed and it works as a head. On the side of $3'$ end (left in Fig. 2) from the head, only L-type character strands are allowed. Conversely, on the side of $5'$ end (right in Fig. 2) from the head, only R-type character strands are allowed. Definition 2 reflects this structure. Needless to say, the structure of a substrate strand itself is independent of any character, so it is capable of representing arbitrary sequence of characters.

2.4 State Transition Reaction

Our implementation of state transition reactions is similar to that of irreversible formal chemical reactions in [4]. Figure 3 shows an LL-type state transition reaction $q + a^L \rightarrow p + x^L$ (1) in detail. We assume that δ_{qa}^{LL}, F_1, F_2 and F_3 exist in sufficiently large amount at any time so that the reaction can occur whenever reactants of the state transition reaction are present. In contrast, there are exactly one copy of q and x^L as we mentioned in the definition of a configuration. W_1, W_2, W_3 and W_4 are waste strands produced during the reaction.

The reaction is implemented by a chain of strand displacement proceeding from the $5'$ end to the $3'$ end. The existence of both q and a^L is confirmed by the first two steps. The next two steps is driven by fuel species and produce a new state and a character. The final step makes the entire process irreversible.

Other types of state transition reactions (2), (3) and (4) can be implemented similarly. Figure 4 shows the implementation of other types of state transition reactions. For the LR-type reaction (2), δ_{qa}^{LR} has almost the same structure as that of δ_{qa}^{LL}. On the other hand, those of RD-type reactions are quite different.

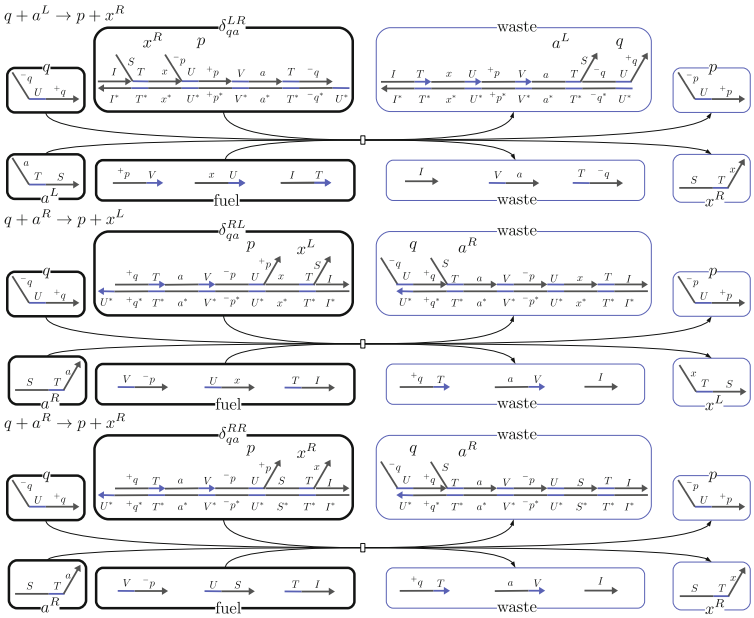

Fig. 4. The implementations of other types of state transition reactions

For example, δ_{qa}^{RL} for the RL-type reaction (3) has the first toehold U^* on the $3'$ end in order to recognize the unique domain of an R-type character. In other words, branch migration proceed to opposite directions between LD-type reactions and RD-type reactions. The difference between δ_{qa}^{RL} and δ_{qa}^{RR} is just as that of LD-type reactions.

2.5 Tape Modification Reaction

The implementation of tape modification reactions is simpler than that of state transition reactions. Figure 5 illustrates the tape modification reaction $a^L + [...c|bd...] \rightleftharpoons [...ca|d...] + b^R$ (5). As Fig. 5 shows, both forward and backward reaction are simple strand displacement reaction. By the forward reaction, a^L is pushed into the tape and b^R is popped, and the only exposed toehold T^*, which we decided as the head, moves right. The backward reaction is mere inverse of the forward reaction.

3 Space-Unbounded DSD Turing Machine

3.1 Definitions

The implementation we described above is not Turing-universal because of its limitation in the size of tape. Generally speaking, Turing-universality requires a model of computation to be able to use unbounded space. So, we implement in our architecture a function to extend the tape whenever more space is needed.

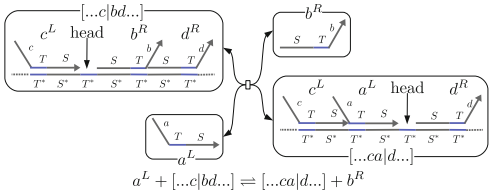

Fig. 5. The implementation of tape modification reaction

Definition 6. *Let* $D = \{L, R\}$ *and* $I = \{1, 2\}$ *be sets of labels. A space-unbounded DSD Turing machine is defined as a* 9*-tuple* $M = (Q, \Gamma, \oslash, \{[,]\}^I, \Sigma, \delta, q_0, F, a_0^d)$ *where*

- Q *is a finite set of states,*
- Γ *is a finite set of symbols called alphabet,*
- $\oslash \in \Gamma$ *is the blank character,*
- $\{[,]\}^I$ *is a set of the terminal characters,*
- $\Sigma = \Gamma \backslash \{\oslash\}$ *is a set of symbols called input alphabet,*
- $\delta \subseteq Q \times \Gamma^D \times Q \times \Gamma^D$ *is a set of 4-tuple called state transition reactions,*
- $q_0 \in Q$ *is the initial state,*
- $F \subseteq Q$ *is the set of final states,*
- $a_0^d \in \Sigma^D$ *is a initial character.*

Some definitions need to be modified in order to enable space-unbounded computation. A tape of a space-unbounded DSDTM is defined as a string $T \in [^I(\Gamma^L)^*(\Gamma^R)^*]^I$, and denoted by $\left[^i a_1 a_2 ... a_n | a_{n+1} ... a_{n+m}\right]^j$. It looks similar to that of space-bounded DSDTM, but $[^i$ and $]^j$ represent actual characters. Tape modification reactions are not changed essentially, but we clarify the following two special cases when the head reaches the end of the tape.

$$a^L + \left[^i ... b|\right]^j \rightleftharpoons \left[^i ... ba| + \right]^j \tag{13}$$

$$\left[^i |b...\right]^j + a^R \rightleftharpoons [^i + |ab...]^j \tag{14}$$

Strictly speaking, $\left[^i ... ba|\right.$ and $\left.|ab...\right]^j$ do not match the definition of tapes, but we do not distinguish them to avoid unnecessary redundancy.

Next, we introduce two kinds of reactions to extend tapes. The first type of reactions are tape generating reactions. They are written as follows.

$$[^i \rightarrow [^j \oslash ... \oslash + \oslash^L \tag{15}$$

$$]^i \rightarrow \oslash ... \oslash]^j + \oslash^R \tag{16}$$

where $[^j \oslash ... \oslash$ and $\oslash ... \oslash]^j$ are additional tapes of L-type and R-type, respectively, We have to impose the condition on the labels that $i \neq j$. This requirement seems unnatural but we will explain the reason later. There are four tape generating reactions according to the label of terminal characters. The second

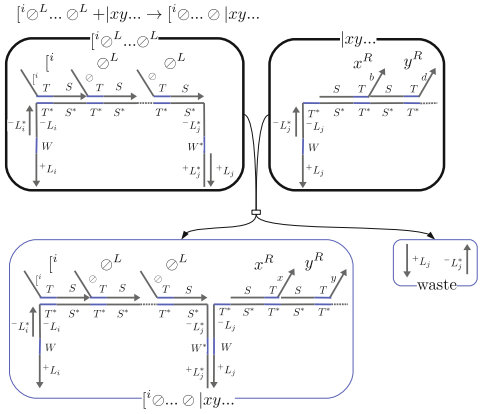

Fig. 6. The implementation of tape extending reaction

type of reactions are tape extending reactions. They attach an additional tape to an existing tape as follows.

$$\left[^{i} \oslash ... \oslash + |a_1 ... a_n\right]^{j} \rightarrow \left[^{i} \oslash ... \oslash |a_1 ... a_n\right]^{j} \tag{17}$$

$$\left[^{i} a_1 ... a_n \right| + \oslash ... \oslash]^{j} \rightarrow \left[^{i} a_1 ... a_n \right| \oslash ... \oslash\right]^{j} \tag{18}$$

Let us describe how a tape is extended by these reactions. Let the configuration be $C = (q, a^R, [^i|b...]^j)$. The transition $C \Rightarrow (q, [^i, |ab...]^j)$ is enabled by the tape modification reaction (14). Next, a new additional tape $[^k \oslash^L ... \oslash^L$ $(k \neq i)$ and a blank character $\oslash^L$ are generated by the reaction (15). Then the tape extending reaction (17) attaches the additional tape to the existing tape, and the tape is extended into $[^k \oslash ... \oslash |ab...]^j$. Finally, the configuration become $(q, \oslash^L, [^k \oslash ... \oslash |ab...]^j)$. Compared to C, it looks as if the blank character is popped out by moving the head to left. A tape can be extended to right similarly.

Needless to say, space-unbounded DSDTMs can simulate an arbitrary space-unbounded Turing machines, so they can perform Turing-universal computation.

3.2 Tape Extending Reaction

We have to change the structure of tapes in order to realize tape extending reactions. Figure 6 shows the implementation of an extendable tape and the reaction to extend the tape to left. Compared to Fig. 2, extendable tapes have additional domains to bind each other on both ends of substrate strands. We call them joint sections. The process of tape extending reaction itself is quite simple. As in Fig. 6, the existing tape $|xy..$ has a toehold domain W exposed and the additional tape has the complement toehold W^*, so they can bind each other, and two joint sections are bound irreversibly by branch migration.

Note that the joint sections of both ends of an additional tape have different labels (that is, $i \neq j$) because of the condition of labels in tape generation

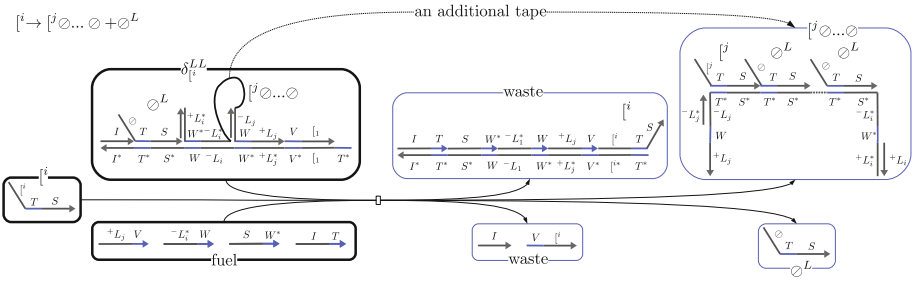

Fig. 7. The implementation of new tape generating reaction

reactions (15) and (16). This is necessary because if they have the same label, they can be bound by each other and the additional tape make a loop by itself. So, we have to use at least two different types of joint sections.

3.3 Tape Generating Reaction

Tape generating reactions can be realized in a way similar to the state transition reactions. Figure 7 illustrates the tape generating reactions (15). The complex $\delta^{LL}_{[i}$ corresponds to δ^{LL}_{qa} in Fig. 3. The additional tape $[^j \oslash \ldots \oslash +\oslash^L$, which is drawn as a black curved line on $\delta^{LL}_{[i}$ in Fig. 7, is attached to $\delta^{LL}_{[i}$ turned upside down. It is bound by joint sections of both ends and the content part of the tape is rounded. Other types of tape generating reactions is are similar.

4 Conclusions

In this paper, we proposed a way to implement (a variant of) Turing machine with DNA molecule using DNA strand displacement cascades. In the first half, we implemented the space-bounded version of Turing machine. It can be constructed by two types of reactions, state transition reactions and tape modification reactions. State transition reactions change the state of Turing machine and produce a new character. Basic approach to implementing them is to use mediating complexes as in the stack machine [4]. However, in our construction, since characters have directionality in order to designate the movement of the head in tape operations, we must design the mediating complex taking the directionality into consideration. On the other hand, tape modification reactions are for the tape operations. Our design of tapes made it possible to modify their content easily, so tape modification reactions themselves can be realized by simple toehold-mediated strand displacement reactions.

In the second half, we constructed the space-unbounded version of Turing machine by improving the structure of tapes. We introduced joint parts at which an additional tape can be attached and terminal characters behave as the signals for generating additional tapes. Once the head has reached the end of the tape, the terminal character is emitted and it triggers the tape generating reaction.

After that, the additional tape is bound to the existing tape by the joint part. This mechanism allows our construction to perform space-unbounded computation and achieves Turing-universality.

However, when it comes to the feasibility of laboratory experiment, we have some points to notice. First, it is technically difficult to directly synthesize a tape of arbitrary input because we cannot control exactly the position where a character strand is bound. So, we have to initially prepare an blank tape $[\oslash ... \oslash |]$, which can be synthesized by mixing a substrate strand and many blank characters $\oslash^L$, then give an input by program. Second, we use a long DNA molecule as a storage, but actual DNA molecules are easily breakable. So, although our framework theoretically enables space-unbounded computation, the size of memory is practically bounded by the durability of DNA molecules. Finally, since the basic idea of performing computation by interaction between free-floating molecules is inherited from the implementation of the stack machine [4], our constructions could not overcome problems deriving from it. Especially, our system will not run correctly if there are two or more copies of state strands in the solution, but it is difficult to prepare an exact number of molecules in laboratory. On the third problem, it is not altogether impossible to overcome by making a femtoliter droplets that contains single molecule using a special laboratory technique [5]. Another possible solution is to change the frameworks so that an arbitrary copy of machines in the same solution. For example, if we can link the state with the tape somehow and construct each machine by an independent single molecule as in Bennett's scheme. This can also improve the first problem because parallelism enhance the probability that tapes survive.

References

1. Beaver, D.: A universal molecular computer. DNA Based Comput. **27**, 29–36 (1996)
2. Bennett, C.H.: The thermodynamics of computationa review. Int. J. Theor. Phys. **21**(12), 905–940 (1982)
3. Lakin, M.R., Phillips, A.: Modelling, simulating and verifying turing-powerful strand displacement systems. In: Cardelli, L., Shih, W. (eds.) DNA 2011. LNCS, vol. 6937, pp. 130–144. Springer, Heidelberg (2011). doi:10.1007/978-3-642-23638-9_12
4. Qian, L., Soloveichik, D., Winfree, E.: Efficient turing-universal computation with DNA polymers. In: Sakakibara, Y., Mi, Y. (eds.) DNA 2010. LNCS, vol. 6518, pp. 123–140. Springer, Heidelberg (2011). doi:10.1007/978-3-642-18305-8_12
5. Rondelez, Y., Tresset, G., Tabata, K.V., Arata, H., Fujita, H., Takeuchi, S., Noji, H.: Microfabricated arrays of femtoliter chambers allow single molecule enzymology. Nature Biotechnol. **23**(3), 361–365 (2005)
6. Rothemund, P.W.: A DNA and restriction enzyme implementation of Turing machines. DNA Based Comput. **27**, 75–119 (1996)
7. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Natural Comput. **7**(4), 615–633 (2008)
8. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proc. Nat. Academy Sci. **107**(12), 5393–5398 (2010)