A Programmable DNA Automaton Model for Context-Free Grammars

W.-G. Li[1,4], Y.-S. Ding[1,2], D. Ruan[3], Z.-D. Huang[1], and S.-H. Shao[1,2]

[1] College of Information Sciences and Technology
[2] Engineering Research Center of Digitized Textile & Fashion Technology, Ministry of Education
Donghua University, Shanghai 200051, P. R. China
[3] The Belgian Nuclear Research Centre (SCK•CEN), Boeretang 200, B-2400 Mol, Belgium
[4] College of Mathematics and Computer Sciences, Anhui Normal University, Wuhu 241000, China
*Email: ysding@dhu.edu.cn

Abstract

Context-free grammars (CFGs) are widely applied in character string modeling. The languages accepted by CFGs are also accepted by pushdown automaton (PDA), and vice verse. Taking palindrome language as an example, we propose a novel method to construct a programmable DNA-based PDA, which is equivalent with the CFGs. We give out the nucleotides encoding for input alphabets, stack alphabets, the detectors to report the final result of PDA, and the transition rules. The major features of our method are demonstrated as follows: (1) to encode the components of PDA, we use general linear double-strands DNA (dsDNA) but not circular dsDNA; (2) we use only one restriction enzyme (*FokI*), that is employed for not only the transiting to state but also the operating to stack.

Keywords: DNA computing model, pushdown automaton, programmable, context-free grammars

1. Introduction

The formal grammars especially CFGs are widely applied in character string modeling. The languages accepted by CFGs are also accepted by PDA, and vice versa [1].

DNA is a potential computing tool because of its powerful parallel, huge storage, and lower energy [2]. The original work was proposed by Adleman, in which he showed how to use DNA at biological lab to solve directed Hamilton path problem in 1994 [2]. Since then, there have already been many exciting results emerged known as DNA-based computing [2-14]. Most work on DNA computing can be summarized to the following two directions. One is focused on how to use biological experiment to solve a kind of computational problems (such as Hamilton path problem). The other is focused on how to implement 1-tape Turing machine using DNA.

On the 1-tape Turing machine, there are two representative work: DNA universal Turing machine proposed by Rothemund [10], and the 2-states-2-symbols finite state automaton proposed by Shapiro's group [11]. In Rothemund's work [10], the instantaneous description of a Turing machine was encoded in a circular dsDNA. It mimicked the running of universal Turing machine by a series of cutting and binding operation. It needed as many as five kinds of restriction enzymes. In addition, it needed human's interfering while it was running. In Shapiro's work [11], it took enzymes as its hardware and used dsDNA to encode its input alphabets and transition table. It mimicked the processing of a finite state automaton (FSA) by digesting and ligating the input symbol sequentially with *FokI* and *T4* respectively. It could operate in an autonomous method without external human's interfering in principle.

In this paper, inspired by Rothemund's and Shapiro's work, taking palindrome languages as an example, we propose a DNA-based PDA. Unlike their work, we use linear dsDNA to encode the components of PDA. Also we use only one enzyme (*FokI*) to implement the operating to both state and stack. We encode input alphabets and stack alphabets in dsDNA. Its transition rules are encoded in dsDNA with double four-nucleotides sticky ends. We use only one restriction enzyme (*FokI*) to implement not only state transiting but also the operation of stack. The detectors designed in advance report the results of the PDA. Through a series of circles of digestion by *FokI* and ligation by *T4*, it completes the running of a PDA.

This paper is outlined as follows. In section 2, we give an example about the CFGs for palindrome languages. In section 3, we describe the design of autonomous DNA-based PDA. Finally, conclusions are given out in section 4.

2. The Context-Free Grammars for Palindrome Languages

CFGs are widely applied to model problems related to character string. As an example, we consider the following CFGs:

$$G=\{V_T, V_N, P, S\},$$

where V_T is the set of terminal, $V_T=\{a, b, c\}$; V_N is the set of non-terminal, $V_N=\{S\}$; P is the set of production rule, $P=\{S\rightarrow aSa,\ S\rightarrow bSb,\ S\rightarrow c\}$; and S is the start symbol.

The above grammars can accept the following palindrome languages: $L=\{wcw^T|\ w\in\{a,b\}^*\}$ [1]. We can construct a PDA, which is equivalent with G. To process an input string, the workflow of the PDA can be divided into two phases. On the first phase, it pushes an alphabet to the stack before reading out the input alphabet c. It will push A when the current input alphabet is a, and it will push B when the current input alphabet is b. On the second phase, it pops up an alphabet from the stack after reading out the input alphabet c. It will pop up A when the current input alphabet is a, and it will pop up B when the current input alphabet is b. The running of the PDA will stop when the top alphabet of the stack mismatches with the current input alphabet or the input tape is empty. Action table of the PDA are shown in Table 1. If and only if the tape and stack are all empty, the input string is a palindrome language.

As such, the following PDA are constructed:

$M=\{\{q_0, q_1, q_2, q_3\}, \{a, b, c, t\}, \{A, B, E\}, \delta, q_0, E\}$,

where $\{q_0, q_1, q_2, q_3\}$ is the set of possible state, q_0 is the initial state, and q_3 is the final state; $\{a, b, c, t\}$ is the set of input alphabet, and t is the terminal; $\{A, B, E\}$ is the set of stack alphabet, and E is the initial alphabet of the stack; δ is a finite set of action rule. According to an action rule which is related with the current state, the current top alphabet of stack, the current input alphabet, this PDA will get a new state and a new top of the stack. An action rule has the form $\delta(q_0, a_0, A_0)=\{(q_1, A_1)\}$, which denotes that the current state is q_0, the current input alphabet is a_0, the current top alphabet of stack is A_0, the new state is q_1 and the new top alphabet is A_1. To get the new top alphabet, there are three possible operations to the stack: pushing down, popping up and unchanging.

3. A DNA-Based Automaton Model for CFGs

In this section, we discuss how to construct an autonomous DNA-based PDA for M. It includes the nucleotides encoding for input alphabets, stack alphabets, instantaneous description at timestamp τ, action rules, and detectors to report the final results of M.

3.1. The nucleotide encoding for alphabets

There are two kinds of symbols in M. The set of input alphabet is $\{a, b, c, t\}$ and the set of stack alphabet is $\{A, B, E\}$. The encoding for all these alphabets must be distinct with each other. Here, the length of the encoding is 6 and all encoding are shown in Fig. 1.

The hardware of M consists of two enzymes: a restriction nuclease and a ligase. We use *FokI* as the restriction nuclease enzyme. It is a class II restriction enzyme and its recognition site is *5'-GGATG-3'*. It will generate a four-nucleotides sticky end digested by *FokI*. There are three kinds of sticky ends for each input alphabet. The different four-nucleotides sticky end digested in different position denotes the different combination of state and alphabet. State q_3 denotes that the input string is a palindrome language when M halts. We can make the same conclusion when the top alphabet of the stack is E and the current alphabet is T. So there are only nine kinds of combinations. All possible combinations and its four-nucleotides sticky end are shown in Table 2.

Table 1. All action rules of the PDA.

The top of stack and current state		***a***		***b***		***c***		***t***	
		state	**stack**	**state**	**stack**	**state**	**stack**	**state**	**stack**
A	q_1	q_1	Push *A*	q_1	Push *B*	q_2	Unchanged	Halt	Halt
	q_2	q_2	Pop *A*	Halt	Halt	Halt	Halt	Halt	Halt
B	q_1	q_1	Push *A*	q_1	Push *B*	q_2	Unchanged	Halt	Halt
	q_2	Halt	Halt	q_2	Pop *B*	Halt	Halt	Halt	Halt
E	q_0	q_1	Push *A*	q_1	Push *B*	q_2	Unchanged	Halt	Halt
	q_2	Halt	Halt	Halt	Halt	Halt	Halt	q_3	Pop *E*

As the same, it will produce the different four-nucleotides sticky ends digested by *FokI* in stack alphabet. If it is digested at the first four nucleotides, it means popping up of the stack. And if it is digested at the last four nucleotides, it means pushing down of the stack. All possible four-nucleotides sticky ends in stack alphabet are shown in Table 3.

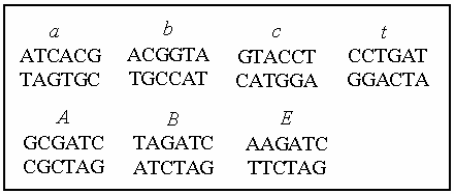

Fig. 1. The encoding of all alphabets for *M*.

Table 2. Encoding for combination of state and input alphabet.

Encoding& <state/alphabet>	***a***	***b***	***c***	***t***
q_0	*ATCA*	*ACGG*	*GTAC*	
q_1	*TCAC*	*CGGT*	*TACC*	
q_2	*CACG*	*GGTA*		*TGAT*

Table 3. All possible four-nucleotides sticky ends in stack alphabets.

Encoding& action/stack alphabet	***A***	***B***	***E***
Push-down	*CTAG*	*CTAG*	*CTAG*
Pop-up	*CGCT*	*ATCT*	*TTCT*

3.2. The instantaneous description at timestamp τ

The formal description of instantaneous description at timestamp τ is shown in Fig. 2. The two back-to-back *FokI* recognition site may be considered as the head of *M*. The left one will digest in the stack alphabet and the right one will read out the current input alphabet. *Stack* is the string of the encoding for stack alphabet, where the leftmost six is the bottom alphabet and the rightmost six is the top alphabet. *S2* is the fragment and its length is to make sure the cutting site by *FokI* is just among the encoding for the top alphabet. The sticky end on *Stack* formed by *FokI* distinguishes the operations to the stack according to Table 3. *Input* is the string of the encoding for input alphabet, where the leftmost six is the encoding for the current input alphabet. *S1* is another fragment and its length is to make sure the cutting site by *FokI* is just among the encoding for the current input alphabet. The sticky end on *Input* formed by *FokI* decides the current state and the current symbol according to Table 2.

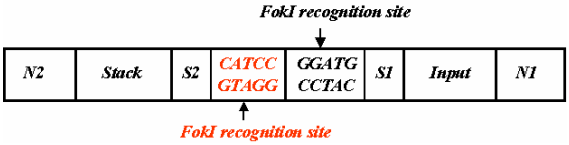

Fig. 2. The formal description of instantaneous description at timestamp τ .

3.3. The nucleotide encoding for action rule

It will produce correct instantaneous description at timestamp τ *+1* according to the instantaneous description at timestamp τ and action rules. At timestamp τ , the current symbol of *Input* and the top symbol of *Stack* will be digested by *FokI* , and it will produce two fragments. So the encoding for the transition function must have two four-nucleotides sticky ends, which are complementary with the two fragments. Under the action of *T4*, the two sticky ends of the proper action rule will ligate respectively with

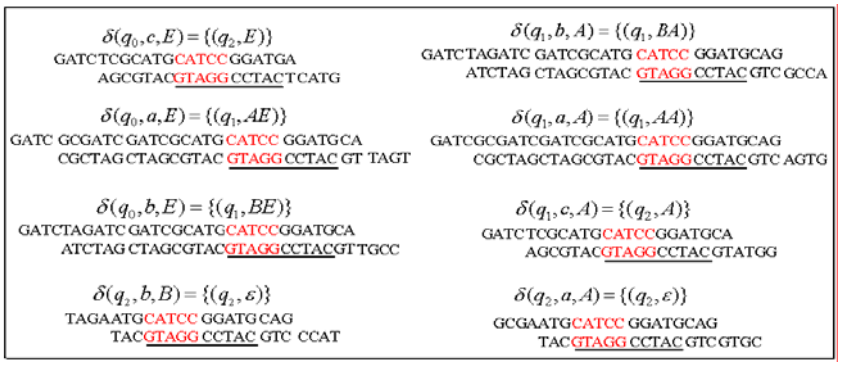

Fig. 3. The encoding for all possible action rules.

the two fragments and get the new instantaneous description at timestamp $\tau + 1$.

The encoding for all possible action rules are depicted in Fig. 3. There, the underlined part is the recognition site of *FokI*.

3.4. The real nucleotide encoding for detectors

The process will continue until there is not appropriate action rule matching with the two fragments or the machine meets the terminal alphabet when we mix all components and required enzyme together under appropriate temperature. According to Table 1, if the language is a palindrome language, the current top alphabet of stack would be *E* and the current alphabet of input would be *t*. So there is only one detector to be designed in advance to report the final result. The encoding for this detector is shown in Fig. 4. The encoding for this detector contains two four-nucleotides sticky ends and the length of 280 in the middle is for gel electrophoresis. Through PCR and gel electrophoresis, we can make the decision if the language is a palindrome language or not.

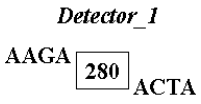

Fig. 4. The encoding for ***Detector_1***.

4. Conclusions

In this paper, we provide a method to construct a DNA-based PDA, which is equivalent with CFGs. In this DNA-based model, all components are encoded in general linear double-strands DNA. Furthermore, we employ only one restriction enzyme (*FokI*) for not only the state transiting but also the operation of the stack. When mixing all components of this PDA and the necessary reagent, it will process the input string autonomously until getting the final result. By gel electrophoresis, we can draw a conclusion whether the input string is a palindrome language or not.

Acknowledgements

This work was supported in part by the National Nature Science Foundation of China (No. 60474037 and 60004006), Program for New Century Excellent Talents in University, and Specialized Research Fund for the Doctoral Program of Higher Education from Educational Committee of China (No. 20030255009).

References

[1] Z. -L. Jiang, S. -X. Jiang, "Formal language and automaton," *Beijing: Tinghua university press*, 2003. (In Chinese).

[2] L. M. Adleman, "Molecular computation of solutions to combinatorial problems," *Science*, pp. 1021–1024, 1994(266).

[3] R. J. Lipton, "DNA solution of hard computational problem," *Science*, pp. 542–545, 1995(268).

[4] Y. -S. Ding, S. -H. Shao, and L. -H. Ren, "DNA Computing and Soft Computing," *Beijing: Science press*, 2002. (In Chinese).

[5] D. Faulhammer, A. R. Cukras, R. J. Lipton, and L. F. Landweber. "Molecular computation: RNA solutions to chess problems," *Proc. Natl. Acad. Sci*, pp. 1385–1389, 2000(97).

[6] E. Winfree, F. R. Liu, L. A. Wenzler, and N. C. Seeman, "Design and self-assembly of two-dimensional DNA crystals," *Nature*, pp. 539–544, 1998(394).

[7] K. Sakamoto, *et al*, "Molecular computation by DNA hairpin formation," *Science*, pp. 1223–1226, 2000(288).

[8] Q. Liu, *et al*, "DNA computing on surfaces," *Nature*, pp. 175–179, 2000(403).

[9] A. J. Ruben, L. F. Landweber, "The past, present and future of molecular computing," *Nature Rev. Mol. Cell Biol.*, pp. 69–72, 2000(1).

[10] P. W. K. Rothemund, in *DNA Based Computers: Proceedings of the DIMACS Workshop, April 4, 1995,Princeton University* (eds R. J. Lipton, E. B. Baum) pp.75-119 (American Mathematical Society, Providence, Rhode Island, 1996).

[11] Y. Benenson, *et al*, "Programmable and autonomous computing machine made of biomolecules," *Nature*, pp. 430–434, 2001(414).

[12] V. Balzani, A. Credi, and M. Venturi, "Molecular logic circuits," *Chemphyschem*, pp. 49–59 2003(4).

[13] K. Sakamoto, *et al*, "State transitions by molecules," *Biosystems*, pp. 81-91, 1999(52).

[14] C. Mao, T. H. LaBean, J. H. Reif, and N. C. Seeman, "Logical computation using algorithmic selfassembly of DNA triple-crossover molecules," *Nature*, pp. 493-496, 2000(407).