

# Biomolecular Implementation of Computing Devices with Unbounded Memory

Matteo Cavaliere<sup>1</sup>, Nataša Jonoska<sup>2</sup>, Sivan Yogev<sup>3</sup>,  
Ron Piran<sup>4</sup>, Ehud Keinan<sup>4,5</sup>, and Nadrian C. Seeman<sup>6</sup>

<sup>1</sup> Department of Computer Science and Artificial Intelligence,  
University of Sevilla, Sevilla, Spain

`martew@inwind.it`

<sup>2</sup> Department of Mathematics, University of South Florida,  
Tampa, FL 33620, USA

`jonoska@math.usf.edu`

<sup>3</sup> Department of Computer Science, Technion, Haifa 32000, Israel

<sup>4</sup> Department of Chemistry, Technion, Haifa 32000, Israel

`{sivan_y, ronppy}@techunix.technion.ac.il`

<sup>5</sup> The Scripps Research Institute, La Jolla, CA 92037, USA

`keinan@scripps.edu`

<sup>6</sup> Department of Chemistry, New York University

New York, NY 10003, USA

`ned.seeman@nyu.edu`

**Abstract.** We propose a new way to implement (general) computing devices with unbounded memory. In particular, we show a procedure to implement automata with unbounded stack memory, push-down automata, using circular DNA molecules and a class II restriction enzyme. The proposed ideas are inspired by the results from [1]. The same ideas are extended to show a way to implement push-down automata with two stacks (i.e, universal computing devices) using two circular molecules glued with a DX molecule and a class II restriction enzyme. In this case each computational molecule also contains a DX portion. These devices can potentially be incorporated in an array of TX molecules.

## 1 Introduction

A general idea for using successive restriction cuts on a double stranded DNA in order to simulate a Universal Turing machine was proposed by Rothmund [10]. This was experimentally achieved by Benenson et. al. [1, 2] who have implemented finite state automata with two states and two input symbols. In fact, several different automata were constructed by changing the computational molecules which indicate the state transitions. In [2] they show that the enzyme *FokI* can cleave even if the molecule is nicked which removes the necessity of a ligase.

On the other hand, circular DNA has been proposed for encoding information and using for computing theoretically [4, 9, 13] and implemented experimentally

[5, 8]. In this paper we combine these two ideas and propose using biomolecules (in particular circular DNA strands) and a class II restriction enzyme to implement computing devices, push-down automata, with unbounded memory. Since these automata are more powerful than finite state automata, this provides a feasible way for experimental increase of the computational power.

In Section 2 we present a procedure to implement a push-down automaton, that is a computing device with stack as an (unbounded) memory. In classical computation theory (see for ex. [6]), push-down automata are considered the simplest computing devices with unbounded memory and are strictly more powerful than finite state automata. The technique presented here is based on two main ideas: the use of circular DNA strands (which contains the information of the stack and the input symbol) and the use of a unique restriction enzyme, *PsrI*, which is able to cut a DNA strand in two places at the same time. Alternatively, the use of *PsrI* could be substituted with two back to back restriction sites for a class II enzyme similar to *FokI* (similar use was proposed in [10]). The theoretical analysis of splicing systems using this type of enzymes have not been developed and with this paper we hope that such investigations will be initiated.

The potential implementation described here can be considered as a generalization of what was done in [1] where finite state automata were implemented using linear DNA strands. The use of circular DNA molecules allows addition of an unbounded memory to the machine. The basic idea is that the circular DNA contains the instantaneous configuration of the push-down automaton, that means, in a single molecule, the contents of the stack and the content of the input yet to be read are encoded. The enzyme *PsrI* cuts the circular DNA in two places leaving overhangs (sticky ends) corresponding to the elements on the top of the stack-memory (on one side) and to the next input symbol to be read from the input tape (on the other side).

Once the circular DNA has been cut, a “computational” linear DNA strand encoding the transition of the machine that corresponds to the sticky ends is inserted into the circular DNA. In this way, a transition of the push-down automaton is implemented that consists of: an update of the stack memory, a move of the input-head over the input tape and a change of the state. Following the idea used in [1, 2] the state and the input symbols of the machine are encoded as a pair in the sticky end produced by the enzyme cut. To simplify the exposition, we show in detail how to implement a (simple) push-down automaton using two states, two input-symbols and two stack-symbols, accepting the (non regular) language  $\{a^n b^n \mid n \in \mathbb{N}\}$ .

In Section 3 we show how to implement push-down automata with two stacks using two circular molecules that are attached by two DX molecules and the restriction enzyme *FokI*. The idea is very similar to the one described for implementing push-down automata with one stack. This implementation is particularly interesting because push-down automata with two stacks are equivalent to Turing machines ([6]). We also describe a way how the DX molecule connecting the two “stacks” molecules can be incorporated in a two dimensional array and in that way potentially the whole computational process scaled up.

## 2 DNA Implementation of Push-Down Automata

### 2.1 Push-Down Automata with One Stack

In this section, we recall the definition of push-down automata, and some well known theoretical results. This section is mainly based on the material found in the classical automata theory book [6].

A push-down automaton (PDA) is a finite state automaton with a stack memory (called simply, stack). The class of languages recognized (accepted) by PDA's is the class of context-free languages that strictly includes the class of regular languages (recognized by finite state automata).

The PDA has control of both an input tape and a stack (see Figure 1). The stack is the memory of the machine and it works as a “first in - last out” list. That is, symbols may be entered or removed only at the top of the list such that a symbol that is entered (push) at the top pushes the rest of the symbols on the stack one step “down”. Similarly, when a symbol is removed (pop) from the top of the list, the remaining symbols on the stack move one step up.

Informally, a transition of a PDA is defined in the following way: at each step, an input-symbol and the stack-symbol at the top of the stack, are read. According to these symbols and the current state, the PDA changes its state and updates the stack, i.e., it either adds a stack-symbol at the top of the stack, removes one stack-symbol from the top of the stack, or leaves the stack unchanged. The computation stops when no transitions can be applied anymore. The input is accepted if (and only if) it has been entirely read and the PDA is in a final state (similarly as in the case of finite state automata).

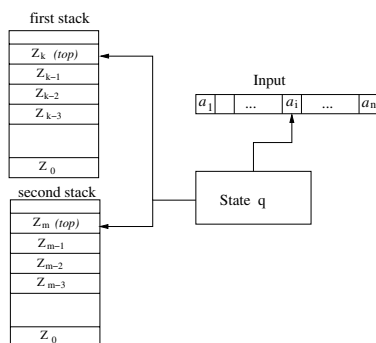


Fig. 1. A Push-down automaton

Well known examples of languages recognized by PDA are the language of *palindrome words* and  $\{a^n b^n \mid n \in \mathbb{N}\}$ . We give the formal definitions of PDA following [6].

**Definition 1.** A push-down automaton  $M$  is a system  $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  where :

$Q$  is a finite set of states;

$\Sigma$  is an input alphabet (its elements are called input-symbols);

$\Gamma$  is a stack alphabet (its elements are called stack-symbols);

$q_0$  in  $Q$  is the initial state;

$Z_0$  in  $\Gamma$  is a particular stack-symbol called the start symbol;

$F \subseteq Q$  is the set of final (terminal) states;

$\delta$  is the transition mapping from  $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$  to finite subsets of  $Q \times \Gamma^*$ .

The interpretation of the move (transition):

$$\delta(q, a, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_m, \gamma_m)\},$$

where  $q$  and  $p_i$ , ( $1 \leq i \leq m$ ), are states,  $a$  is in  $\Sigma$ ,  $Z$  is a stack-symbol, and  $\gamma_i$  in  $\Gamma^*$  is the following. The PDA in state  $q$ , reading an input-symbol  $a$  with  $Z$  as the top stack-symbol, for any  $i$ , can enter state  $p_i$ , replace symbol  $Z$  by string  $\gamma_i$ , and advance the input-head one symbol.

A PDA may also have empty moves

$$\delta(q, \epsilon, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_m, \gamma_m)\}$$

such that the PDA in state  $q$  with  $Z$  at the top stack-symbol, independently of the input-symbol being scanned, can enter state  $p_i$  and replace  $Z$  by  $\gamma_i$  for any  $i$ ,  $1 \leq i \leq m$ . We adopt the convention that the leftmost symbol of  $\gamma_i$  will be placed at the top of the stack and the rightmost symbol lowest on the stack.

The PDA stops if a transition from state  $q$ , reading input-symbol  $a_i$ , and stack-symbol  $Z$  is not defined.

If  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  is a PDA, we say  $(q, aw, Z\alpha) \rightarrow (p, w, \beta\alpha)$  if  $\delta(q, a, Z)$  contains  $(p, \beta)$ . We use  $\rightarrow^*$  to denote the reflexive and transitive closure of  $\rightarrow$ . The acceptance of a language by a PDA can be defined in two (equivalent) ways: by entering a final state or by emptying the stack. In this presentation we use the first manner.

**Accepted languages (by final states).** For a PDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  we define  $L(M)$  the language accepted by final state, to be:

$$\{w \mid (q_0, w, Z_0) \rightarrow^* (p, \epsilon, \gamma), \text{ for some } p \in F \text{ and } \gamma \in \Gamma^*\}.$$

We recall some classical results (see for ex. [6]).

**Theorem 1.** The class of languages accepted by PDA's is exactly the class of context-free languages (that strictly includes the class of regular languages).

**Theorem 2.** For every PDA there is an equivalent two state PDA that accepts the same language.

### 2.2 Implementing PDA's: An Example

To simplify the implementation of a PDA, without loss of generality, we suppose that every input word is inserted with a symbol indicating end of input denoted with  $\tau$ . Then a word is accepted by a PDA if, and only if, when reading the end-of-input symbol  $\tau$ , the PDA stops entering one of the final states. If the PDA stops before reading the end-of-input symbol (i.e. no transitions are possible), then the word is not accepted by the PDA.

In what follows we present a possible implementation of a PDA that accepts the non regular language  $L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$ . A PDA accepting the language  $L_1$  is the following:  $M_1 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ , with  $Q = \{0, 1\}$ ,  $\Sigma = \{a, b\}$ ,  $\Gamma = \{Z, \#\}$ ,  $Z_0 = \#, F = \{1\}$ .

$M_1$  has five transitions

- (i)  $\delta(0, a, \#) = (0, Z\#)$ : in state 0, reading an input-symbol  $a$  and stack-symbol  $\#$ , remain in state 0 and add (push) a  $Z$  at the top of the stack.
- (ii)  $\delta(0, a, Z) = (0, ZZ)$ : in state 0, reading an input-symbol  $a$  and stack-symbol  $Z$ , remain in state 0 and push a  $Z$  at the top of the stack.
- (iii)  $\delta(0, b, Z) = (1, \epsilon)$ : in state 0, reading an input-symbol  $b$  and stack-symbol  $Z$ , change to state 1 and remove (pop) a  $Z$  from the top of the stack.
- (iv)  $\delta(1, b, Z) = (1, \epsilon)$ : in state 1, reading an input-symbol  $b$  and stack-symbol  $Z$ , remain in state 1 and pop a  $Z$  from the top of the stack.
- (v)  $\delta(1, \tau, \#) = (1, \#)$ : in state 1, reading end-of-input  $\tau$ , and  $\#$  on the top of the stack, remain in state 1 (the computation halts).

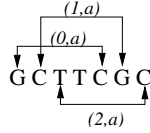
In the initial configuration the stack contains only  $\#$  and the initial state is  $q_0 = 0$ ; the input-head scans the first symbol of the input string. It is easy to see that the language accepted by  $M_1$  is  $L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$ . We show how to implement this PDA using the enzyme *PsrI* together with circular molecules containing the information for the stack and the tape, and linear DNA strands for the transitions. The enzyme cleaves as depicted in Figure 2 (for further details see [17]).



**Fig. 2.** The restriction mode of *PsrI*

**Encoding.** The input letters are encoded as  $a = TTC$  and  $b = AAC$ . Codes of the stack symbols, using strings of 5 letters can be chosen  $Z = TCCAG$  and  $\# = CAAAC$ .

The initial circular DNA strand corresponds to the initial configuration of the PDA: it contains the initial configuration of the stack and the input to be read. This is “codified” as follows. The input is written such that any pair of



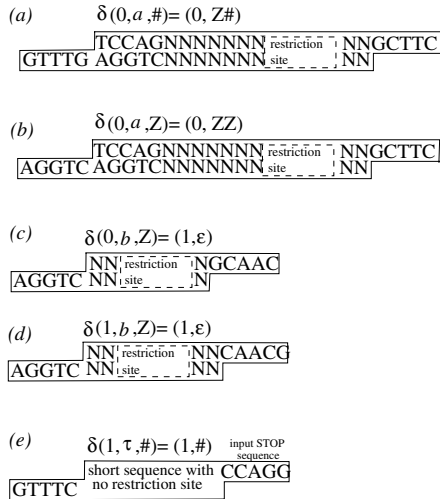
**Fig. 3.** Coding used to simulate three different states

input-symbols are separated with *GC*, which is also added in front of the first symbol and after the last symbol. A stop-sequence (the end-of-input) *CAGGC*, follows the input. For example, suppose the input is *aabb*, then it is codified with *GCTTCGCCTTCGCAACGCAACGCCAGGC*. (the separator *GC* is indicated in italics) The sequence *GC* allows “moving” between different states (following the idea used in [1]).

This coding allows for *three states* reading the same symbol to be encoded. For example, the symbol *a* surrounded by *GC*’s is encoded with *GCTTCGC*. We can assume that *GCTTC* encodes “state 0-reading *a*”, *CTTCG* encodes “state 1-reading *a*” and *TTCGC* encodes “state 2-reading *a*” (see Figure 3).

Following this idea, in our example we have (we only mention the codes that are used): *GCTTC* for *a* in state 0; *GCAAC* for *b* in state 0; and *CAACG* for *b* in state 1.

The circular DNA strand representing the initial configuration of the PDA is depicted in Figure 6. The first part *CAAAC* represents the initial configuration of the stack (containing only symbol #; virtually an empty stack). The middle



**Fig. 4.** Transitions of the PDA

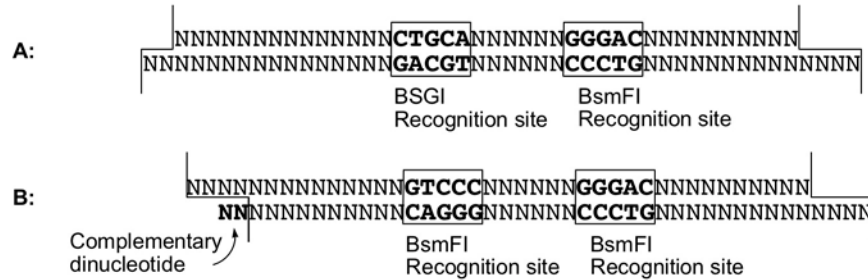
portion *GAACNNNNNTAC* is the restriction site for the enzyme *PsrI*. The final part is the input as described above.

**Transitions.** Together with the circular molecule that represents the initial configuration of the PDA five linear DNA strands that encode the transitions of the PDA are also needed: one strand for each transition. The linear strands corresponding to the transitions in our example are depicted in Figure 4. These molecules are added in the solution together with the circular molecules corresponding to the initial configuration of the PDA. The enzyme cuts the circular molecule and the transition molecules are allowed to connect to the circular molecule, after which, the enzyme cleaves again and the process is repeated.

Transitions (a) and (b) in Figure 4 add a symbol *Z* on the stack, but do not change the states. This is obtained by having *NN* between the restriction site and the sticky-end representing the input symbol to the right, and after a sequence of seven *N*'s having the sequence *TCCAG* representing *Z* on the left hand side. Since the enzyme cleaves seven nucleotides away from the restriction site on both sides, on the right hand side, the next cut will appear at the next input symbol, in same position (i.e. same state) for reading the input sequence, and on the left hand side the symbol '*Z*' will be added to the stack. The transitions (c) and (d) are similar, except they do not allow for writing *Z*'s on the stack, but for removing them (the number of *N*'s present between the restriction site and the sticky-end representing the input symbol depends in the way the state-symbol is encoded).

The strand in Figure 4(e) implements the transition 5 and stops in the final state if the end-of-input is read.

In each transition, two ligation reactions occur. Ligation on one side of the transition molecule corresponds to the input, while ligation on the other side corresponds to the stack. It is important to know which side is ligated first, since there is degeneracy in the stack side (it includes only *Z* or #), and therefore different transition molecules may be ligated at that end at any stage. For example, when the transition molecule in Figure 4(b) is the correct molecule for the next step, the molecules in Figure 4(c) and (d) may be ligated at the stack side, caus-



**Fig. 5.** Two alternative strategies to enhance the ligation rates at the input end relative to the ligation at the stack end

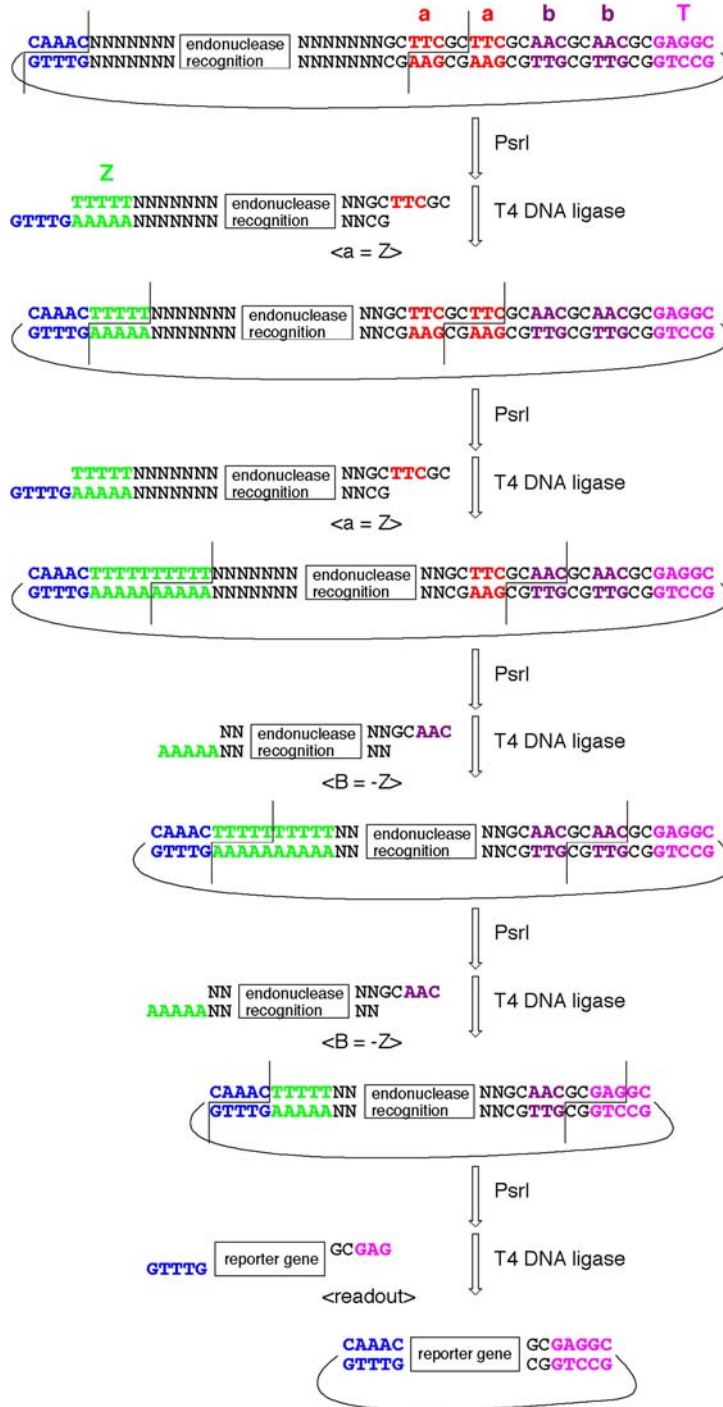


Fig. 6. Accepting of the string aabb

ing a halt of the computation process. One way to reduce this problem is to make sure that the first ligation occurs at the input side. Since intermolecular reactions are faster than their intramolecular counterparts, once the ligation occurred at the input side, the probability of correct ligation at the stack side would increase dramatically. This situation may be achieved by reduction of the length of the sticky end at the stack side. Kinetically, annealing processes are faster for longer sticky ends. This goal may be achieved by two alternative ways (see Figure 5):

- Use two different class II restriction enzymes, such as BsgI, which leaves a 2bp sticky end for the stack side, and BsmFI, which leaves a 4bp sticky end for the input side.
- Use two identical restriction sites, such as BsmFI and add a 2bp molecule to the mixture, which will correspond to the inner part of the stack sticky end, causing a need for a 2-step ligation at that end, which will further slow down the ligation rate at that end.

Figure 6 describes the steps of the PDA for accepting the word *aabb*. We start with the initial configuration of the PDA stored in the circular molecule as represented in Figure 6. The consecutive cuts and inserts of the transition molecules are presented in the steps that follow. The cuts end when the ‘stop’ transition molecule recognizes the end-of-input symbol and it ends in the terminal state. In this case, it has to contain the short sequence without the restriction site.

### 3 Push-Down Automata with Two Stacks

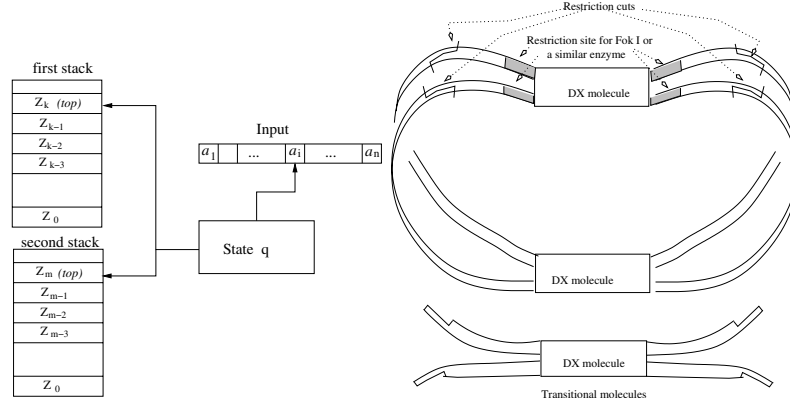
It is possible to consider push-down automata using two stacks instead of one.

Informally, a transition of a push-down automaton with two stacks (shortly, 2PDA) is defined in the following way: at each step, an input-symbol from the input tape, and the stack-symbols at the top of the first stack and at the top of the second stack, are read. According to the current state, the 2PDA updates both stacks; adds stack-symbols at the top of the stack(s), removes one stack-symbol from the top of the stack(s), or leaves the stack(s) unchanged. At the same time it changes its state and, reads the next input-symbol (as in the case of PDA’s, if there is an empty move, the input-head does not advance).

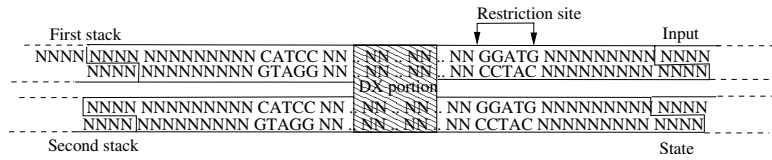
The 2PDA stops if for a given configuration, the next transition is not defined. At start, the 2PDA is in an initial state  $q_0$ , the input is placed on the input tape and the first and second stack contain the respective initial stack-symbol  $Z_0$  and  $Z_1$ . When the computation stops, the input is accepted if (and only if) it has been entirely read and the 2PDA is in a final state.

*The class of languages accepted by 2PDA’s is the class of recursively enumerable languages (i.e., 2PDA’s are equivalent to Turing machines).* The proof of this result and more formal details on 2PDA’s can be found in [6].

A graphical description of a 2PDA is described in Figure 7 to the left.



**Fig. 7.** A graphical description of 2PDA (left) and molecules used for implementation of a 2PDA (right). For simplicity, the middle portion of the body of the DX molecule is not presented



**Fig. 8.** An encoding of an instantaneous configuration of a 2PDA in a circular DX molecule

### 3.1 Implementing 2PDA's Using DX Molecules

In a similar way as presented in Section 2 we can implement 2PDA's i.e., push down automata with two stacks. In this case two circular molecules are “connected” with two DX molecules (called here *circular DX molecule*) and a class II restriction enzyme similar to *FokI* (as described in Figure 7, right). The content of the two stacks is stored in the two strands to the “left” of the “upper” DX molecule (the symbols of the two stacks are stored exactly as in the case of the PDA). On the other side of the DX molecule, the symbols of the input string and the current state of the 2PDA are stored, each on one of the strands.

*FokI* was used in [1] for implementation of a finite state automaton, and we use it in our description for implementing 2PDA. However, any similar enzyme can be used as well.

The restriction site for *FokI* is placed in four places of the strands, connected with the “upper” DX molecule, to be read away from the DX portion of the molecule (see Figure 7 to the right and in a more detail Figure 8 that also depicts the way *FokI* works). As described in Figure 8, the enzyme cleaves the molecule in these four places, and a new computational molecule with sticky ends corresponding to a particular transition of the automaton is inserted. This

new molecule has two linear strands with four sticky ends “connected” with a DX portion in the middle. In this way a transition of a 2PDA is simulated.

If the circular DX molecule cannot be cut anymore, or no DX molecule with sticky ends can be inserted in the circular DX molecule, then the computation halts and the circular DX molecule obtained is considered *final*. The final circular DX molecule contains a specific sequence which indicates whether the molecule has been accepted.

The circular DX molecule, containing the initial configuration of a given 2PDA is essentially the same for all 2PDA’s. The computation differs only in the set of molecules representing the transitions of the automaton.

**Example of one Transition**

We present the idea of implementing 2PDA on a simple example simulating a single transition of a 2PDA.

Consider the 2PDA  $M_2 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, Z_1, F)$  with states  $Q = \{0, 1\}$ , input alphabet  $\Sigma = \{a\}$ , initial state  $q_0 = 0$ , alphabet of the stack-symbols  $\Gamma = \{Z, \#\}$ , final state  $F = 1$  and  $Z_0 = Z_1 = \#$  are the stack-symbols present at the beginning of the computation in the first and second stack, respectively. Suppose that one of the transitions present in  $\delta$  is:  $\delta(0, a, \#, \#) = (1, Z\#, Z\#)$  meaning, when the 2PDA is in state 0, reads  $\#$  on both stacks, and  $a$  as input-symbol, then, the 2PDA push  $Z$  on top of both stacks and changes to state 1.

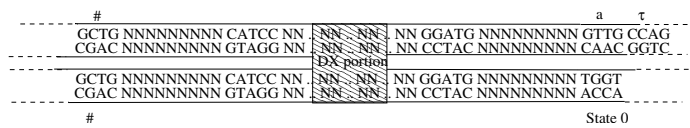
The symbols and the states used by  $M_2$  can be encoded similarly to the case of PDA  $M_1$  in Section 2. We fix a code composed of 4 letters for the symbols in  $\Sigma$  and  $\Gamma$ . Choose:  $a = GTTG$ ,  $Z = TCCA$ , and  $\# = GCTG$ .

Notice that in this implementation the coding of the states is in a “direct” way. There is no need to use the “shift” technique as described in the implementation of  $M_1$ . Choose:  $0 = TGGT$ , and  $1 = ACTC$ .

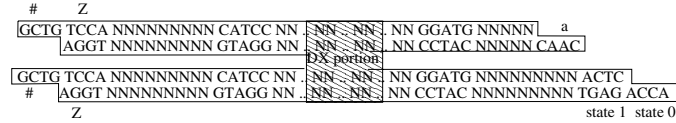
The implementation starts with a circular DX molecule corresponding to the initial configuration of the 2PDA  $M_1$ . The “upper” part of the circular DX molecule containing the initial configuration is described in Figure 9. Suppose that the input is  $a\tau$ , where  $\tau$  is the end-of-input symbol. Set the code of  $\tau$  to be  $CCAG$ .

The molecule that corresponds to the transition  $\delta(0, a, \#, \#) = (1, Z\#, Z\#)$  is depicted in Figure 10. The idea can be generalized to other transitions; it is very similar to the idea used in the implementation of the transitions of a PDA.

The enzyme *FokI* cuts the circular DX molecule in four places and this can be considered as reading the input-symbol, the current state and the symbols on the top of the two stacks. Once the circular DX molecule is cut, one of the



**Fig. 9.** Initial configuration of the circular DX molecule



**Fig. 10.** DX molecule for the transition:  $\delta(0, a, \#, \#) = (1, Z\#, Z\#)$

molecules representing the transitions is inserted into the circular DX molecule. This “insertion” simulates the application of a transition of the 2PDA.

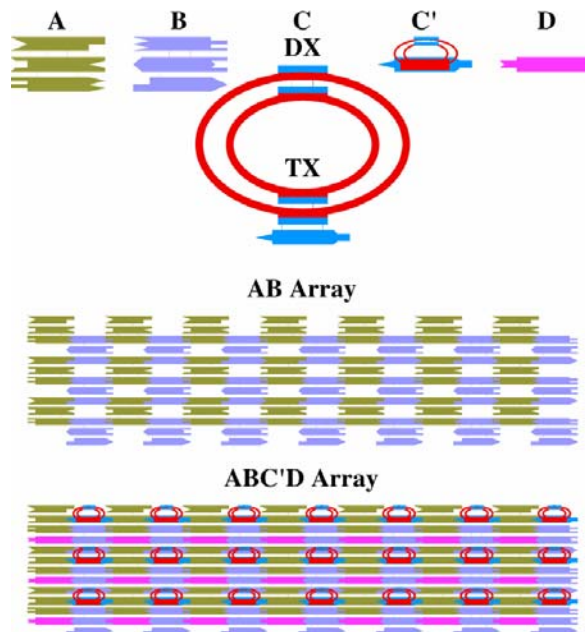
The transition  $\delta(0, a, \#, \#) = (1, Z\#, Z\#)$  is applied to the initial configuration of the 2PDA,  $M_2$ , by having all four sticky ends connected with a corresponding transition molecule. In our example, the DX molecule inserted is the one represented in Figure 10, and the new configuration obtained is such that the next four cuts will correspond to “reading” the end-of-input symbol  $\tau$  and stack-symbol  $Z$  on both stacks and “being” in state 1.

Different transitions can be implemented introducing different transition molecules (with different sticky ends). As in the case of PDA’s, the sticky ends can be adjusted to simulate both push and pop over the stacks, and also the empty moves. Similarly, when no cuts can be done on the circular DX molecule or no transition molecule with corresponding sticky ends can be inserted then the circular DX molecule obtained is final. Checking the presence of some special terminal sequence in the final DX molecule determines whether the input is accepted or not.

### 3.2 2PDA in an Array

Double and triple cross-over molecules have been used as tiles and building blocks for large nanoscale arrays [7, 14, 15]. The assembly of such two-dimensional array can be used to incorporate the circular DX molecules of a 2PDA. The body of the “bottom” DX portion of the circular DX molecule used for storing the instantaneous configuration of a 2PDA can be incorporated in a triple cross-over molecule, such that the sticky ends of the third duplex, from each side, diagonally opposite, are used for connecting the molecule in a TX based array. Schematically this is presented in Figure 11. The drawing aims to give an impression of the means by which the 2PDA units could be inserted into the array; we recognize that it will be much larger than shown, and the C tiles attached to the 2PDA units will need to be much less densely packed.

By including the 2PDA molecules in an array we can potentially (a) scale up the computational process and (b) avoid mutual interactions between the circular DX molecules and formation of dimers and faulty computation that could lead to a wrong result. This process, however, in order to determine precise positions of the 2PDA molecules will require careful coding of the two dimensional array similarly as was done in the bar-code design [16] or in the case of the design of the Sierpinski triangle [11].



**Fig. 11.** Insertion of circular DX molecule storing instantaneous configuration of a 2PDA in an array. C is drawn in the same plane as TX molecules A and B, but C' has been rotated nearly perpendicular to the AB array. D is linear duplex filler

#### 4 Concluding Remarks

We have presented a possible way to implement push-down automata (i.e., finite state automata with unbounded stack memory), using a class II restriction enzyme, circular and linear DNA strands. In the hierarchy of classical computing devices, push-down automata are the simplest computing devices using unbounded memory and are strictly more powerful than finite state automata.

In particular, an implementation of a small PDA, with two states, two input-symbols, and two stack-symbols, accepting the non regular language  $\{a^n b^n \mid n \in \mathbb{N}\}$  was shown. Using the same idea the implementation of a PDA that checks if a string is palindrome is straight forward. Palindromes are a standard example of non regular languages that may be of interest from biological point of view.

From the theoretical point of view, it is easy to describe implementation of a general (non deterministic) PDA with two states and two input-symbols, even using empty-moves in the same manner. Due to Theorem 2 this is enough to implement every PDA, but, the number of stack-symbols may increase. In the implementation presented here, the number of possible stack-symbols is limited to  $4^5$  as the code of each symbol uses 5 nucleotides. This bound is even smaller since other coding constrains may apply. There are methods to reduce the number of stack symbols ([3]) but in this case the number of states increases.

Therefore it is significant to determine how many stack-symbols can be really codified for a DNA implementation using one enzyme *PsrI* or *FokI*.

The paper also includes a way to implement push-down automata with two stacks. This implementation uses again a class II restriction enzyme and circular DX molecules. This result is of particular interest because such class of computing devices is equivalent to Turing machines. Although we used an identical coding for both stacks in our example, this should be avoided such that no cross annealing of the stacks and the transition molecule occurs. It can be adjusted similarly as in the case of PDA when three symbol alphabet is used for the symbols followed by a *GC* or *AT* indicating the two different stacks.

Incorporating circular DX molecules in an array may be a challenging task experimentally, in particular, encoding and assembling the two-dimensional array such that the sticky ends for annealing with the 2PDA molecule appear in the right positions.

## Acknowledgment

Parts of this research was performed while the first author was visiting University of South Florida during the Fall of 2003.

The first author is supported by a grant from the Spanish Ministry of Culture, Education and Sport under the Programa Nacional de Formación de Profesorado Universitario (FPU). Seeman has been supported by grants GM-29554 from the National Institute of General Medical Sciences, N00014-98-1-0093 from the Office of Naval Research, grants DMI-0210844, EIA-0086015 (also supporting Jonoska), DMR-01138790, and CTS-0103002 from the National Science Foundation. Keinan is supported by the German-Israeli Project Cooperation (DIP) and by the Skaggs Institute for Chemical Biology.

## References

1. Y. Benenson, T. Paz-Elizur, R. Adar, Eh. Keinan, Z. Livneh, Eh. Shapiro, *Programmable and autonomous computing machine made of biomolecules*, *Nature*, **414** (2001), 430-434.
2. Y. Benenson, R. Adar, T. Paz-Elizur, Z. Livneh, Eh. Shapiro, *DNA molecule provides a computing machine with both data and fuel*, *Proc. Nat. Acad. Sci. (PNAS)* **100** 5 (2003), 2191-2196.
3. J. Goldstine, J.K. Price, D. Wotschke, *On reducing the number of stack symbols in a PDA*, *Math. Systems Theory* **26** 4 (1993), 313-326.
4. T. Head, *Splicing schemes and DNA*, in *Lindenmayer Systems: Impact on Theoretical Computer Science and Developmental Biology* (G. Rozenberg, A. Salomaa, eds.), Springer, Berlin, 1992, 371-383.
5. T. Head et.al, *Computing with DNA by operating on plasmids*, *BioSystems* **57** (2000), 87-93.
6. J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

7. T. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J.H. Reif, N.C. Seeman, *The construction, analysis, ligation and self-assembly of DNA triple crossover complexes*, *J. Am. Chem. Soc.* **122** (2000), 1848-1860.
8. Liu Q. et al. *DNA computing on surfaces*, *Nature* **403** (2000), 175-179.
9. Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing - New Computing Paradigms*, Springer-Verlag, Berlin, 1998.
10. P.W.K. Rothemund, *A DNA and restriction enzyme implementation of Turing machines*, *DNA Based Computers* AMS DIMACS series **27** (1998), 75-119.
11. P. Rothemund, N. Papadakis, E. Winfree, *Algorithmic Self-assembly of DNA Sierpinski Triangles*, *Preproceedings of 9th DNA Based Computers*, Madison, Wisconsin June 1-4 (2003), 125.
12. J. Sakamoto et al., *State transitions by molecules*, *Biosystems* **52** (1999), 81-91.
13. Y. Sakakibara, H. Imai, *A DNA-based Computational Model Using a Specific Type of Restriction Enzyme*, *Proceedings of the 8th DNA Based Computer*, (M. Hagiya, A. Ohuchi eds.), LNCS 2568, 315-325.
14. E. Winfree, F. Liu, L.A. Wenzler, N.C. Seeman, *Design and self-assembly of two-dimensional DNA crystals*, *Nature* **394** (1998), 539-544.
15. E. Winfree, X. Yang, N.C. Seeman, *Universal computation via self-assembly of DNA: some theory and experiments*, *DNA computers II*, (L. Landweber, E. Baum eds.), AMS DIMACS series **44** (1998), 191-214.
16. H. Yan, T.H. LaBean, L. Feng, J.H. Reif, *Directed Nucleation Assembly of Barcode Patterned DNA Lattices*, *Proc. Nat. Acad. Sci. (PNAS)* **100** No. 14 (2003), 8103-8108.
17. <http://rebase.neb.com/rebase/rebase.html>.