Non-Deterministic Finite State Automata Built on DNA

Robert Nowak[1] and Andrzej Płucienniczak [2]

[1] Warsaw University of Technology, Institute of Electronic Systems, Warsaw, Poland,
email: r.m.nowak@elka.pw.edu.pl
[2] Institute of Biotechnology and Antibiotics, Staroscinska 5, 02-516 Warsaw, Poland,
email: apl@iba.waw.pl

Abstract. This paper describes non-deterministic finite-state automaton based on DNA strands. The automaton uses massive parallel processing offered by molecular approach for computation and exhibits a number of advantages over traditional electronic implementations. This device is used to analyze DNA molecules, whether they are described by specified regular expression. Presented ideas are confirmed by experiment performed in genetic engineering laboratory.

1 Introduction

The method described in this paper uses DNA (deoxyribonucleic acid) molecules for performing computation. The double helix of DNA is formed from two separate DNA strands, connected together (head-to-toe) by hydrogen bonds. The DNA strands may be viewed as chain of nucleotides. There are four nucleotides: adenine, cytosine, guanine, and thymine, abbreviated to `A`, `C`, `G`, and `T` respectively. Each strand has a natural orientation, denoted (according to chemical convention) as 5' and 3' end. The hydrogen bond is selective, `A` bonds with `T`, and `G` bonds with `C`, the pairs `(A,T)` and `(G,C)` are complementary. The DNA strands are complementary if they are built from complementary nucleotides. More information about DNA and basic operations (i.e. hybridization, denaturation, ligation, cutting, PCR) from computer scientists point of view should be found in [1, 4, 5].

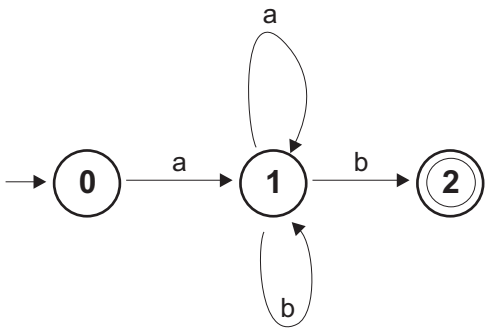
Symbol over some alphabet Σ, denoted in this paper by a small letter or digit, is represented by sequence of consecutive nucleotides of length n. For different symbols these sequences are different. In this work such sequences are written from `5'` to `3'` end. The $\bar{x}$ denotes the sequence complementary to sequence representing symbol x, for example, if x is represented by `ATCCCA`, the complementary sequence is `3'-TAGGGT-5'`, thus $\bar{x}$ is `TGGGAT`.

An ***alphabet*** is a finite nonempty set of symbols. A ***string*** over given alphabet is any finite sequence of symbols (for example ε, a, b, aa, ab, aab, are the strings over $\mathbf{\Sigma} = \{a, b\}$, ε represents empty string). The length of a string R (the number of symbol occurrences in R) is denoted by $|R|$, for example, $|aab| = 3$, $|\varepsilon| = 0$. The strings are represented by DNA strands, and denoted in this paper by capital letter. For example, consider the alphabet $\Sigma = \{a, b\}$, where symbols are represented by `ATCCCA`, `GGTCCT` respectively. The DNA strand for $R = abb$ has sequence `ATCCCAGGTCCTGGTCCT`.

Each subset of strings over given alphabet is called a ***language***. ***Regular expression*** is a useful way to describe some simple languages called ***regular languages***.

For any regular expression the equivalent right-linear grammar can be constructed. The ***right-linear grammar*** is quadruple $\mathbf{G} = (\mathbf{N}, \mathbf{T}, \mathbf{P}, q_0)$, where $\mathbf{N}$ is nonterminal alphabet, $\mathbf{T}$ is terminal alphabet $\mathbf{T} \subseteq \Sigma$, q_0 is the starting symbol (axiom) $q_0 \in \mathbf{N}$, and $\mathbf{P}$ is the set of production rules. Production rules conform the pattern $1 \rightarrow a2$ or $1 \rightarrow a$, where $1 \in \mathbf{N}$, $2 \in \mathbf{N}$ and $a \in \mathbf{T}$. The construction of right-linear grammar for given regular expressions is described in [3]. In this work numbers denotes nonterminal symbols, letters denotes terminal symbols.

The decision whether a given string belongs to a given regular language is undertaken by ***finite state automaton***. The finite state automaton can be constructed [3] for any regular language. If the length of regular expression describing given regular language is $|R|$, then simple algorithm (of linear time complexity for electronic computer) can construct non-deterministic finite state automaton. The number of states (memory complexity) is $O(|R|)$, and time complexity to analyze string X is $O(|R| * |X|)$. The deterministic finite state automaton has the number of states exponentially dependent on length of regular expression, so memory complexity is $O(2^{|R|})$, and the analysis for string X takes $O(|X|)$ steps. A sample finite state automaton is depicted in Fig. 1. The automaton recognizes strings belonging to language $a(a|b) * b$. The grammar $\mathbf{G} = (\mathbf{N}, \mathbf{T}, \mathbf{P}, q_0)$, where $\mathbf{N} = \{0, 1\}$, $\mathbf{T} = \{a, b\}$, $q_0 = 0$, and $\mathbf{P}$ showed in Fig. 1 generates strings belonging to language.



- $0 \rightarrow a1$
- $1 \rightarrow a1$
- $1 \rightarrow b1$
- $1 \rightarrow b$

Figure 1. Finite state automaton and production rules for language $a(a|b) * b$.

The idea described in this paper is to build non-deterministic finite state automaton in vitro. Such a device uses massive parallelism given by molecular approach, and has size complexity (understood as a number of different molecules) $O(|R|)$, where R is regular expression describing given language. The analysis for string X has $O(|X|)$ time complexity.

2 Molecular production

The ***molecular production*** is a biological system which conditionally creates designed DNA strand. It is the basic element in molecular automaton used to implement transition.

The molecular production, denoted $A \rightarrow B$, creates string XB if and only if the input is XA (A, B, X are sub-strings, $|A| > 0$). Such system checks if the sequence of nucleotides representing condition (here A) is presented at the 3' end of the input string, and then creates the output string: the DNA strand is copied from input, but the condition sequence is replaced by sequence representing result of production (here B). Therefore, for input XA, the XB is obtained (Fig. 2). It should be mentioned, that XA is also presented in the output, because input and output are not separated.

If the strand representing input string has not the condition sequence at the 3' end, the molecular production creates nothing. For example production $A \rightarrow B$ for input XC, where $C \neq A$, provides only XC (Fig. 2).

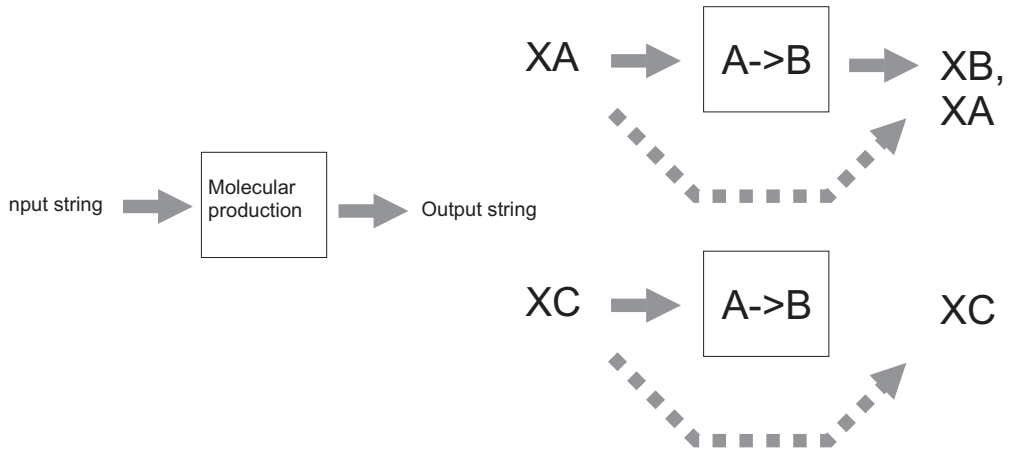


Figure 2. Molecular production $A \rightarrow B$; X, A, B, C are strings, and $|A| > 0$, $A \neq C$.

3 Molecular finite state automaton

Reductions for right-linear grammars

Theorem: If string S belongs to language generated by right-linear grammar $\mathbf{G} = (\mathbf{N}, \mathbf{T}, \mathbf{P}, q_0)$, then can be reduced to string q_0 (axiom). The following reduction rules are used for the last symbols in a string:

- $a \rightarrow 0$ when $0 \rightarrow a \in \mathbf{P}$,
- $a1 \rightarrow 0$ when $0 \rightarrow a1 \in \mathbf{P}$.

Lemma: When string is generated from axiom (q_0) by right-linear grammar it has at most one non-terminal symbol. This is the last symbol of the string.

Proof of lemma (mathematical induction): Assume, that $S_n = w_1 w_2 ... w_n A_n$, where $w_i \in T$, $A_n \in N$. S_{n+1} should be obtained from S_n by production $A_n \rightarrow w_{n+1} A_{n+1}$ (it retain the condition) or by production $A_n \rightarrow w_{n+1}$ (also retains it).

Generated strings are: $q_0 \rightarrow w_1 A_1 \rightarrow w_1 w_2 A_2 \rightarrow ... \rightarrow w_1 w_2 ... w_{n-1} A_{n-1} \rightarrow w_1 w_2 ... w_n$, where $w_i \in T$, $A_i \in N$.

Proof of theorem: String $S_n = w_1w_2...w_n$ can be generated by right-linear grammar $\mathbf{G} = (\mathbf{N}, \mathbf{T}, \mathbf{P}, q_0)$, only from string $S_{n-1} = w_1w_2...w_{n-1}A_{n-1}$, where $A_{n-1} \rightarrow w_n \in \mathbf{P}$. Therefore strings S_{n-1} can be constructed from S_n by reductions $w_n \rightarrow A_{n-1}$. If $\mathbf{P}$ does not contain production rule $A_{n-1} \rightarrow w_n$, then string $w_1w_2...w_n$ does not belong to language generated by $\mathbf{G}$.

String $S_n = w_1w_2...w_nA_n$ can be created only from strings $w_1w_2...w_{n-1}A_{n-1}$, where $A_{n-1} \rightarrow w_nA_n \in \mathbf{P}$. If production rule $A_{n-1} \rightarrow w_nA_n \notin \mathbf{P}$, then string $S_n = w_1w_2...w_nA_n$ does not belong to language generated by $\mathbf{G}$. Strings S_{n-1} are constructed from S_n by reductions $w_nA_n \rightarrow A_{n-1}$. Because $|S_{n-1}| = |S_n| - 1$, after $n-1$ steps, the set of strings of length equal 1 is obtained. If the axiom (q_0) is in this set, the string $w_1w_2...w_nA_n$ can be generated by $\mathbf{G}$.

Molecular automaton based on reductions

For given language the corresponding right-linear grammar is constructed [3]. To analyze the input string the reductions (described in theorem) are performed. If the axiom (starting symbol) is obtained the input string belongs to grammar (is accepted).

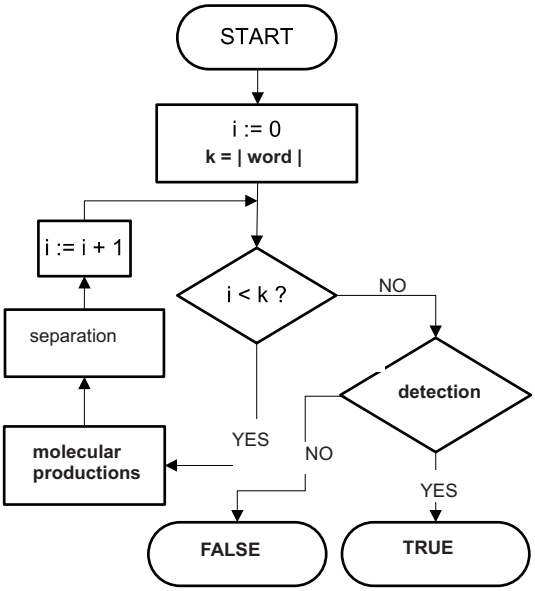


Figure 3. Molecular automaton - algorithm.

Such an idea is the core of molecular automation. This device takes advantage of molecular production to implement reductions. The algorithm is shown in Fig. 3.

Firstly, the automaton is prepared and DNA strand representing the input string is added to a vessel.

Then k steps (where $k = |S|$) of productions and separations are performed. After each production the last symbols from the string could be reduced. The separation removes input string (present in the output of molecular production), i.e. for input XA and molecular production $A \rightarrow B$ only XB is kept. It should be noted that each

molecular production could work independently of each other, so in a single step many different reductions should be performed.

Finally the axiom is detected. If such string is obtained, the answer is $true$, the input string is accepted by automaton. Otherwise, the answer is $false$.

Example

Consider regular language $a(a|b) * b$ and the right-linear grammar shown in Fig. 1. The reductions (molecular productions) for this language are: $b \rightarrow 1$, $b1 \rightarrow 1$, $a1 \rightarrow 1$ and $a1 \rightarrow 0$.

The molecular automaton performs 3 steps when the input string abb is analyzed: $abb \rightarrow ab1 \rightarrow a1 \rightarrow \{0, 1\}$. The axiom (symbol 0) is present, thus abb belongs to given language.

For bbb and the same automaton the reductions are: $bbb \rightarrow bb1 \rightarrow b1 \rightarrow 1$. The axiom is not present, so bbb is rejected.

4 Molecular production - realization

The molecular production is the basic element used to build the automaton considered in this paper. Below the details of this process are presented.

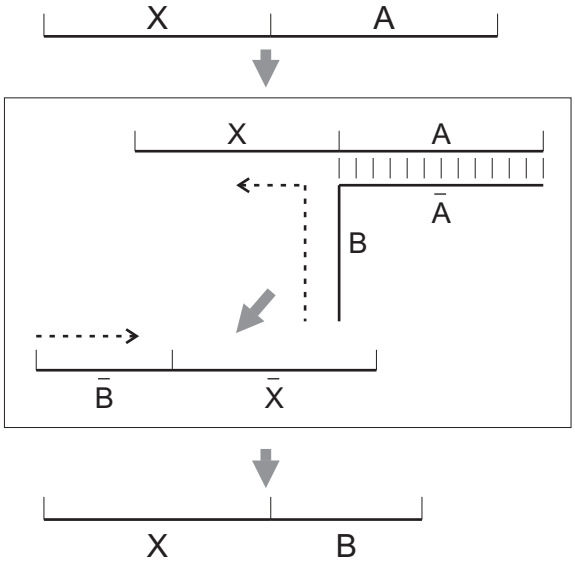


Figure 4. Molecular production $A \rightarrow B$. The production engine has sequence $\bar{A}B$.

The process of molecular production (illustrated in Fig. 4) needs the DNA strand called a production engine. This strand contains two parts: the first is complementary to the conditional part of molecular production (i.e. $\bar{A}$ for $A \rightarrow B$), the second has nucleotides representing the result of production (i.e. B for $A \rightarrow B$).

Firstly, the production engine partially hybridizes to the input string (only if it has the proper sequence on 3' end). Next, the special polymerization with DNA polymerase

"jump" is performed. A strand built by polymerase has the sequence complementary to the output string. Finally, the polymerization is performed, so the output string is produced. It should be noted that if the hybridization does not occur (the production engine and input string are not partially complementary), polymerase does produce only strand complementary to production engine, which can be easily removed.

The probability of DNA polymerase "jump" is very small, so in experiments the PCR is applied. PCR (and dilution) can also remove the strands representing input strings (separation in Fig. 3).

5 Molecular automaton example

The molecular automaton (described in section 3) is represented in vessel by a set of production engines. For language $a(a|b) * b$ these molecules are depicted in Fig. 5. For the considered language there are 4 productions, the sequences correspond to reductions: $a1 \rightarrow 0$, $a1 \rightarrow 1$, $b1 \rightarrow 1$ and $b \rightarrow 1$ respectively.

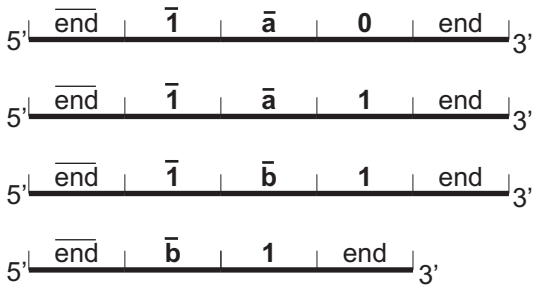


Figure 5. Molecular automaton (molecular engines) for language $a(a|b) * b$.

The DNA strand representing a model string aab is shown in Fig. 6. As denoted previously there are sequences representing symbols, and some temporary ones, denoted *start*, *end*, used as primers.

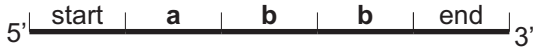


Figure 6. DNA strand representing a model word abb.

At the beginning of calculations the molecules implementing automaton (production engines) and input string are put into a vessel. Next the molecular productions are performed (firstly the hybridization, next polymerization with "jump", then denaturation and finally polymerization, like in section 4). Because of the length of string aab, three steps of molecular production and separation are performed.

The first step is schematically depicted in Fig. 7. The production engine representing reduction $b \rightarrow 1$ bonds to strand working as input string (the reduction $b \rightarrow 1$ is called to be active). The other engines are not bounded, so after molecular production and separation in the vessel the molecule corresponding to $aa1$ string were presented.

In the second step, depicted in Fig. 8, the reduction $a1 \rightarrow 1$ is active, and the string $a1$ is obtained.

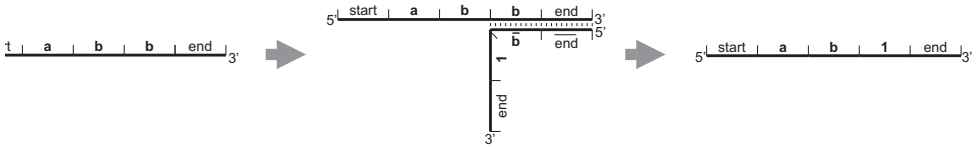


Figure 7. Examine string aab (for language $a(a|b) * b$). Step 1.

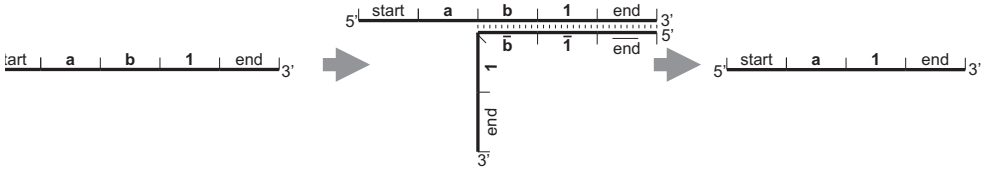


Figure 8. Examine string aab (for language $a(a|b) * b$). Step 2.

The third step (Fig. 9) shows parallelism of the described approach. The two reductions are active: $a1 \rightarrow 0$ and $a1 \rightarrow 1$. So the two different molecules were created: molecule representing string 0 and 1 respectively.

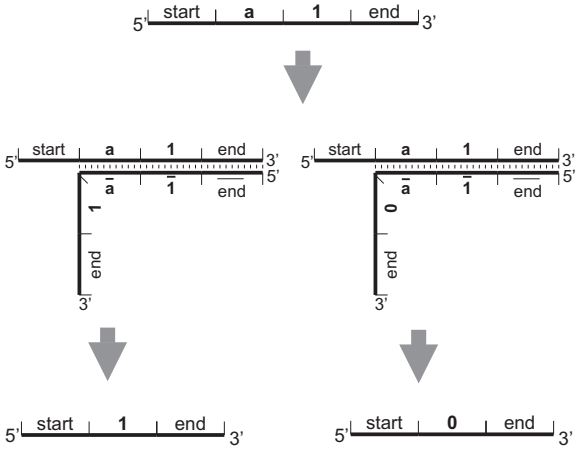


Figure 9. Examine string aab (for language $a(a|b) * b$). Step 3.

Finally, the detection is performed. It is done by checking if the molecule representing the axiom (string 0 here) is presented in the vessel. Because such DNA strand is produced in the third step, then answer is *true*. The string aab is accepted by the molecular automaton.

6 Conclusion

The short comparison between complexity of different finite state automata is presented in Tab. 1. The molecular approach has advantages over electronic implementations,

automaton	size	time
electronic nondeterministic	$O(\|\mathbf{r}\|)$	$O(\|\mathbf{r}\| * \|S\|)$
electronic deterministic	$O(2^{\|\mathbf{r}\|})$	$O(\|S\|)$
molecular	$O(\|\mathbf{r}\|)$	$O(\|S\|)$

Table 1. Complexity finite state automata described by regular expression r when the string S is analyzed. Size for molecular automaton is the number of different molecules used for calculation.

because each possible transitions from a given state are simultaneously considered (take advantage of massive parallel processing).

There are a few others works describing realization an automata by using the molecular approach. In [5] only propositions are given, in [6] the human assistance is needed. The described method uses one vessel to code many states (because implements nondeterministic automaton), and the person reads the current symbol from the input string, and decides to which vessel the molecules should be added (simulating transition). It complicates the experiments and makes the process slower, more expensive and much prone to errors. The interesting idea shown in [2], which uses *Fok*I enzyme, was experimentally proved. The main disadvantage of this method is small maximum number of states and transitions (256).

The presented non-deterministic finite state automaton can be treated as an alternative way of performing molecular computation. It is the step toward constructing molecular computer.

Practically, it might be used in biological and medical research, for searching DNA sequences described by regular expression. When a requested sequence is simple (can be described by regular expression), described non-deterministic automaton perform this task quickly and inexpensively (compare with currently used DNA sequencing), so for example the diagnosis of a genetic disease should be performed on a large scale.

Bibliography

[1] M. Amos. *Theoretical and experimental DNA computation*. Springer, 2005.

[2] Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, and E. Shapiro. Programmable and autonomous computing machine made of biomolecules. *Nature*, 414:430–434, 2001.

[3] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.

[4] K. Lila, R. Kitto, and G.Gloor. A computer scientist's guide to molecular biology. *Soft Computing*, 5:95–101, 2001.

[5] G. Păun, G. Rozenberg, and A. Salomaa. *DNA Computing: NewComputing Paradigms*. Springer, 1998.

[6] J.A. Rose, Y. Gao, M. Garzon, R. C. Murphy, R. Deaton, S. Franceschetti, and E. Stevens Jr. Dna implementation of finite-state machines. In *2nd Anneal Genetic Programming Conference, Morgen Kaufmann*, pages 160–165, 1997.