Autonomous Push-down Automaton Built on DNA*

Tadeusz Krasiński, Sebastian Sakowski
Faculty of Mathematics and Computer Science, University of Łódź
Banacha 22, 90-238 Łódź, Poland
E-mail: krasinsk@uni.lodz.pl, sakowski@math.uni.lodz.pl

Tomasz Popławski
Department of Molecular Genetics, University of Łódź
Pomorska 141/143, 90-236 Łódź, Poland
E-mail: tplas@biol.uni.lodz.pl

Keywords: push-down automaton, DNA computing

Received: July 19, 2012

In the paper we introduce a biomolecular implementation of the push-down automaton (one of the theoretical models of computing devices with unbounded memory) using DNA molecules. The idea of this improved implementation was inspired by Cavaliere et al. (2005).

Povzetek: Predstavljen je avtonomni avtomat na osnovi DNK po vzoru Cavaliereja.

1 Introduction

In the paper written by Cavaliere, Janoska, Yogev, Piran, Keinan, Seeman [4] the authors propose a theoretical model (*i.e*. not tested in laboratory) of implementation of the push-down automaton built on DNA. The idea was inspired by two papers: the first one by Rothemund [7] who proposed a simulation of the Turing machine - the basic theoretical model of computation - and the second one by Benenson, Paz-Elizur, Adar, Keinan, Livneh, Shapiro [1] who proposed a simulation of the finite automaton – the simplest model of computation. The above three implementations represent all the basic theoretical models of computers in the Chomsky hierarchy. But all these simulations have weak points in different places.

The Rothemund model is not autonomous. A person must interfere in the process to obtain the required sequences of actions through many restriction enzymes. This is likely a reason why nobody tested it experimentally.

Next, Benenson *et al*. [1] model is autonomous, programmable and was tested in laboratory but it represents the simplest computational model - a finite automaton (in fact it was only 2-states 2-symbols finite automata). The next propositions along the same idea (Soreni *et al.* [10], Unold *et al.* [11], Krasiński and Sakowski [6]) essentially did not improve the situation.

The last model, Cavaliere *et al.* [4] is more powerful (a push-down automaton), autonomous, programmable (although the action of it was illustrated only on one simple example) but the problem lies in obtaining the right sequence of ligations of transition molecules to the input and to the stack (represented by the same circular DNA). The authors themselves indicate this problem "It is first important to know which side is ligated first, since there is degeneracy in the stack side … and therefore different transition molecules may be ligated at that end at any stage" and propose two ways to reduce (not eliminate) the problem. Moreover, another problem in their model is that it is not clear biochemically whether the used enzyme *Psr*I could not accidentally cut transition molecules of the first kind (which add the symbol Z to the stack) before ligating it to the input and to the stack.

In this paper we suggest an improvement of the last model of push-down automata to avoid these problems. However, it is a theoretical model not tested yet in laboratory. We propose a new shape of transition molecules and another kind of restriction enzymes, which cut only when the ligation of a transition molecule to the circular molecule of the input will be accomplished on both sides.

2 Push-down Automaton

In this section we recall shortly the definition of the push-down automaton (PDA). More information can be found in any textbook (Hopcroft and Ullman [5]; Sipser [8]).

A push-down automaton is a finite automaton (nondeterministic) which has a stack, a kind of simple memory in which it can store information in a last-in-first-out fashion.

* This project is supported by the National Science Centre of Poland (NCN). Grant number: DEC-2011/01/B/NZ2/03022.

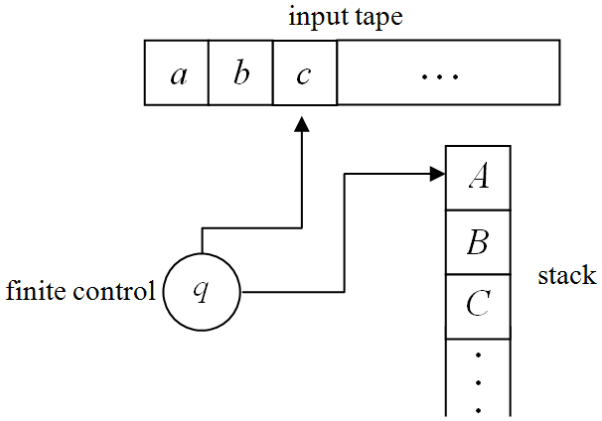


Figure 1: A scheme of the PDA.

In each step the machine, based on its current state (q), the input symbol which is being currently read (c) and the top symbol on the stack (A) performs a move according to a transition rule (from a list of transition rules associated to a given PDA): pops the top symbol from the stack, pushes a symbol (or a sequence of symbols) onto the stack, moves its read head one cell to the right and enters a new state. We also allow ε - transitions in which a PDA can pop and push without reading the next input symbol. The PDA is nondeterministic, so there may be several transitions that are possible in a given configuration. We will denote transition rules in the following way

$$(q, c, A) \to (q', A')$$

where: q' - a new state, A' - a new symbol or a sequence of symbols (may be an empty sequence) which replaces A on the top of the stack.

There are two (equivalent) alternative definitions of acceptance of an input word w: by empty stack and by final state. Since in the presented implementation we use the second one we will recall only that one. A PDA accepts an input word w if it enters a final state (from a distinguished subset of all states) after scanning the entire word w, starting from the initial configuration with w on the input tape and with the special initial symbol $\perp$ on the stack.

The class of languages accepted by PDA is the class of context-free languages which strictly includes the class of regular languages (accepted by finite state automata) and is strictly contained in the class of recursive enumerable languages (accepted by Turing machines).

We will illustrate the above definition by giving an example of PDA which adds integers. It will be our main example in the implementation.

Example 1. A PDA accepting the language

$$ADD = \{a^n b^m c^{n+m} : n, m \in N\}$$

has four states: q_0 - initial state, q_1, q_2, q_3 - final state. The PDA has the following transitions:

1. $(q_0, a, \perp) \to (q_0, A\perp)$
2. $(q_0, a, A) \to (q_0, AA)$
3. $(q_0, b, A) \to (q_1, AA)$
4. $(q_1, b, A) \to (q_1, AA)$
5. $(q_1, c, A) \to (q_2, \varepsilon)$
6. $(q_2, c, A) \to (q_2, \varepsilon)$
7. $(q_2, \varepsilon, \perp) \to (q_3, \varepsilon)$

A sequence of configurations (state, remaining input word, stack) of this PDA on the input word $aabccc \in ADD$ is as follows.

$$(q_0, aabccc, \perp) \overset{1}{\to} (q_0, abccc, A\perp) \overset{2}{\to} (q_0, bccc, AA\perp)$$
$$\overset{3}{\to} (q_1, ccc, AAA\perp) \overset{5}{\to} (q_2, cc, AA\perp) \overset{6}{\to} (q_2, c, A\perp) \overset{6}{\to}$$
$$(q_2, \varepsilon, \perp) \overset{7}{\to} (q_3, \varepsilon, \varepsilon) \text{ - acceptation,}$$

and on the input word $abc \notin ADD$ is as follows.

$$(q_0, abc, \perp) \overset{1}{\to} (q_0, bc, A\perp) \overset{3}{\to} (q_1, c, AA\perp) \overset{5}{\to}$$
$$(q_2, \varepsilon, A\perp) \text{ - stop the action.}$$

3 The Structure of DNA

DNA (deoxyribonucleic acid) is the storage medium for genetic information in all living things. It is a single-stranded (ss) or a double-stranded (ds) chain made of four nucleotides A, C, T, G. In a dsDNA two ssDNA (with the inverse orientations) are linked by hydrogen bonds in such a way that A can only pair together with T and C with G. To manipulate DNA we take various enzymes from a variety of organisms for catenating, splitting, cutting and copying DNA. In our consideration we will use restriction enzymes (restrictases) which recognize fixed sites in a DNA and cut it, leaving sticky ends on both sides of the cutting place. For instance the restrictase *Fok*I cuts in the following way (Fig. 2).

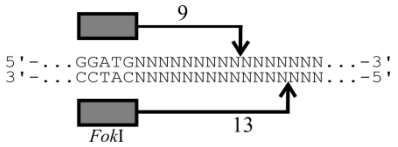


Figure 2: The action of the enzymes *Fok*I.

4 The implementation of PDA

The implementation of a PDA is similar to that of Cavaliere *et al.* [4] with changes which eliminate their obstacles. The main idea of the implementation is as follows.

The basic elements of a PDA i.e. the input tape and the stack are represented in the same circular dsDNA molecule of which one end represents the stack and the second one the input word (Fig. 3).

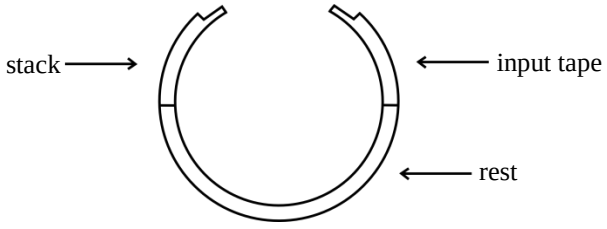


Figure 3: The basic elements of implementation of a PDA.

The sticky end of the stack represents the top symbol on the stack and the sticky end of the input tape represents the first symbol of the input word (to be read) and simultaneously the state of the PDA.

The transition rules of a PDA are suitable DNA molecules which hybridize to both ends of the circular DNA representing this PDA (Fig. 4).

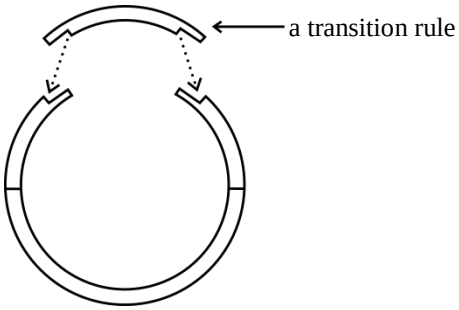


Figure 4: Process of hybridizing a transition rule to both ends of DNA.

After ligation, appropriate restriction enzymes cut this circular molecule. Their actions cause changes in the stack and in the input word according to the move which is represented by this transition molecule. A new idea is that the action of restriction enzymes will take place only when the transition molecule ligate to both ends of the circular molecule. It happens because the chosen restriction enzyme (*Bgl*I) has two separated recognition sites (Fig. 5), which appear both only when a transition molecules ligates to both ends of the circular molecule. After the cut additional molecules and restriction enzymes make adequate changes in the stack and in the input word. Then the next transition rule may act. When a sequence of such transitions leads to reading out the input word and the last sticky end would represent the final state of the PDA, then a long additional DNA molecule ligates to the molecule. It can be detected in the solution by gel electrophoresis. The word is accepted.

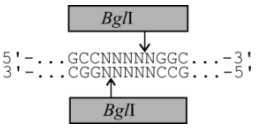


Figure 5: The action of the enzyme *Bgl*I.

5 The Practical Implementation

The idea of the practical implementation will be illustrated on the PDA given in Example 1 i.e. on a PDA performing the addition of integers. The graph of it is represented in Fig. 6.

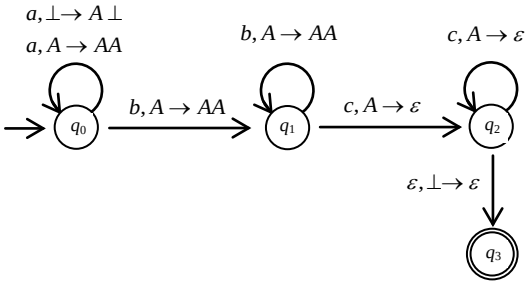


Figure 6: The graph of a PDA which adds integers.

It has seven moves. Each of them is represented by a transition molecule, additional molecules and suitable restriction enzymes (see Appendix 1).

The action of the enzyme *Bgl*I is presented in Fig. 5. The remaining enzymes act as follows (Fig. 7).

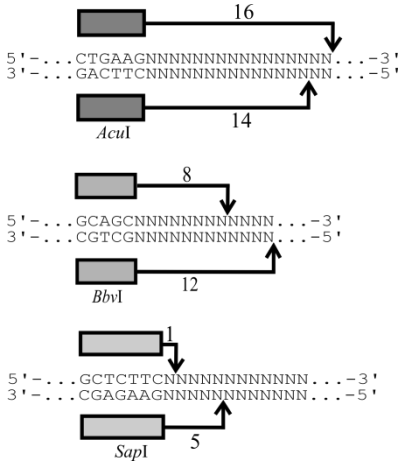


Figure 7: The action of the enzymes *Acu*I, *Bbv*I, *Sap*I.

The sticky end of an input word represents both a symbol and a state of the PDA according to the rules (Fig. 8).

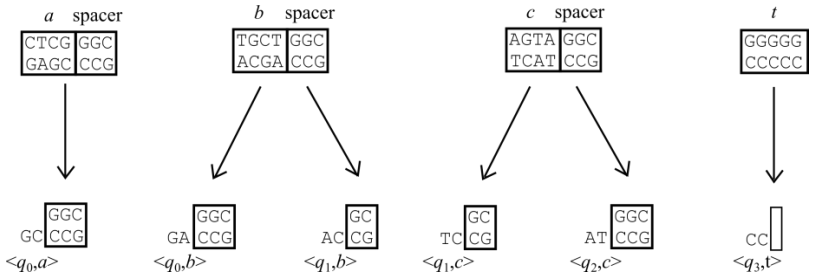


Figure 8: DNA codes of the symbols and pairs <state, symbol>.

The symbols $\{A, \perp\}$ on the stack and their representations on the top of the stack are presented in Fig. 9.

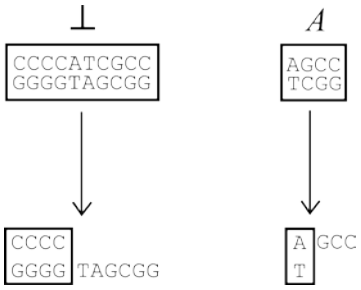


Figure 9: The representations of the stack symbols.

The representation of the considered PDA with the input word *aabccc* in the initial state q_0 and the symbol $\perp$ on the stack is shown in Fig. 10.

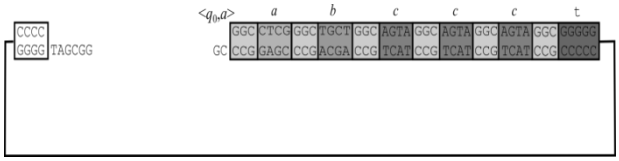


Figure 10: The PDA with the input word *aabccc*.

The action of the PDA will be illustrated on two moves, the first of which pushes a symbol on the stack (Appendix 2) and the second one of which pops the symbol from the stack (Appendix 3).

The main idea of the first move $(q_0, a, \perp) \rightarrow (q_0, A\perp)$ which pushes the symbol A on the stack is to use the restriction enzyme *Bgl*I, which cuts the DNA strand only when the transition molecule merges the stack and the input tape. It is caused by the fact that the enzyme *Bgl*I has two separated recognition sites 5'...GCC(5nt)GGC...3' which appear when the transition molecule ligates to the stack and to the input word. An important fact is to use spacers GGC between symbols of the input word. After the cut the second restriction enzyme *Acu*I together with an additional molecule make a change in the input word.

A second move $(q_1, c, A) \rightarrow (q_2, \varepsilon)$ which pops the symbol from the stack acts by using also the restriction enzyme *Bgl*I (Appendix 3). After cutting with the enzyme *Bgl*I we have to remove actual symbols from the input word and from the stack. The operation of removing from the input word is the same as in the first move (using the restriction enzyme *Acu*I).

Since we could not find a commercial enzyme which cuts a DNA molecule in a long distance from the recognition site and leaves a 3-nt sticky end we have to apply two restriction enzymes (*Bbv*I and *Sap*I)

The remaining moves act similarly. The whole process on the word *aabccc* is presented in Appendix 4.

6 Conclusions

We have presented a new method to implement a push-down automaton based on DNA molecules and restriction enzymes. It is an improved version of the idea presented in [4]. Other attempts (not fully matured and functioning) are in [9], [13], [14]. A new idea is to use a restriction enzyme which has two separate recognition sites. It allows to cut DNA molecules representing elements of a PDA after ligating of transition molecules to both sides of circular DNA. It avoids problems that appeared in Cavaliere *et al.* [4]. This will enable us in the future to construct more powerful automata than PDA, which provides the possibility to solve more complicated problems. Actually we implemented our theoretical model of finite automata (more powerful than the one presented in Benenson *et al.* [1] in a laboratory in the cooperation with a research group from the Department of Molecular Genetics of the Łódź University. This attempt of a laboratory implementation of our research groups is described by Błasiak, Krasiński, Sakowski, Popławski [3]. We tested in the laboratory simultaneous action of two restriction enzymes *Acu*I and *Bbv*I which is a crucial step in the experiment presented in this paper. The next step could be laboratory implementation of the PDA presented in this article.

The circular molecule dsDNA used in our model opens a new possibility to insert and apply our automaton to the bacteria cell. Such a type of DNA molecules are plasmids - heritable DNA molecules that are transmissible between bacterial cells and bacterial genomes. Bacteria controls DNA replication process via origin replication elements. These genetic elements are built with blocks of repeated sequences and replication is initiated when special proteins (e. g. DnaA in *E. coli*) binds to series of repeats. Regulations of bacterial genome and plasmid propagation is possible with use of our automaton by controlling the number of repeat motifs presented in origin (by inserting to the stack or removing from the stack). In a similar way it is possible to control in bacteria not only DNA replication but also transcription of some bacterial genes. Transcription starts when RNA polymerase binds to special genetic elements called promoter. The bacterial promoter is built with some genetic elements essential for efficient initiation of transcription (e.g. -10 and -30 blocks), thus we can switch on and off gene transcription by inserting or deleting some sequence blocks within promoter or even changing the distance between them. This method of DNA replication or transcription control with the use of an automaton has one major advantage in comparison of natural scheme of control – it allows to make some logical calculations before cell take the final decision.

Acknowledgement

We thank prof. Jacek Hejduk for an improvement of the text in the paper.

References

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

[2] Benenson, Y., Adar, R., Paz-Elizur, T., Livneh, Z., Shapiro, E. (2003). DNA molecule provides a computing machine with both data and fuel. PNAS 100, 2191-2196.

[3] Błasiak, J., Krasiński, T., Popławski, T., Sakowski S. (2011). More powerful biomolecular automaton. ArXiv:109.5893v1 [Cs.ET]. Cornel University Library.

[4] Cavaliere, M., Jonoska, N., Yogev, S., Piran, R., Keinan, E., Seeman, N. (2005). Biomolecular implementation of computing devices with unbounded memory. Lecture Notes in Computer Science 3384, 35-49.

[5] Hopcroft, J., Ullman, J. (1979). Introduction to Automata Theory, Languages and Computation. Addison-Wesley.

[6] Krasiński, T., Sakowski S. (2008). Extended Shapiro Finite State Automaton. Foundations of Computing and Decision Science 33, 241-256.

[7] Rothemund, P. (1995). A DNA and restriction enzyme implementation of Turing machines. In DNA Based Computers, American Mathematical Society, Providence, RI , 75-119.

[8] Sipser., M. (2006). Introduction to the Theory of Computation. Thomson Course Technology.

[9] Shi, X., Xin, L., Zhang, Z., Xu, J. (2005). Improve Capability of DNA Automaton: DNA Automaton with Three Internal States and Tape Head Move in Two Directions. Lecture Notes in Computer Science 3645, 71-79.

[10] Soreni, M., Yogev, S., Kossoy, E., Shoham Y., Keinan, E. (2005). Parallel biomolecular computation on surfaces with advanced finite automata. Journal of the American Chemical Society 127, 3935-3943.

[11] Unold O., Troć M., Dobosz T., Trusiewicz A. (2004): Extended molecular computing model. WSEAS Transactions on Biology and Biomedicine 1, 15-19.

[12] Yin, P., Turberfield, A., Reif, J. (2004). Designs of Autonomous Unidirectional Walking DNA Devices. Tenth International Meeting on DNA Computing (DNA10), Milano, Italy. Lecture Notes in Computer Science 3384, 7-10.

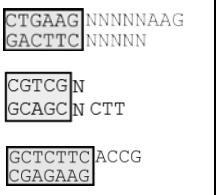
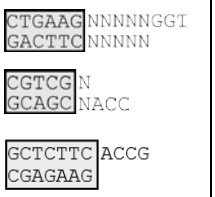
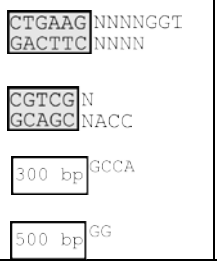
[13] Zhang, Z., Xu, J., Liu, J., Pan, L. (2006). Programmable pushdown store base on DNA computing. Lecture Notes in Computer Science 4115, 286-293.

[14] Zhang, Z., Jie, L., Xiao-Long, S. (2008). Biomolecular Pushdown Automaton Based on the DNA Computing. Chinese Journal of Computers 31, 2168-2172.

Appendix 1

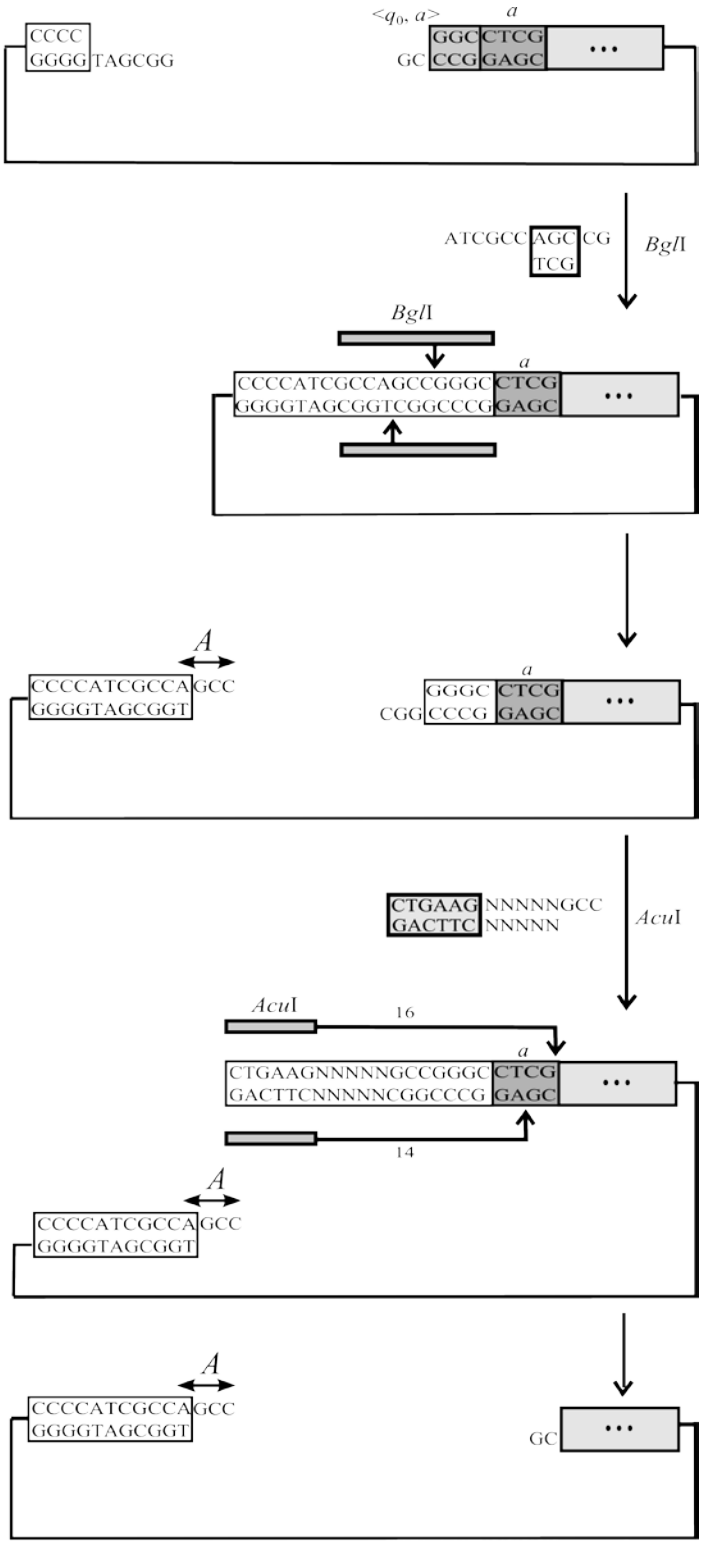
The transition rules and their molecular representations.

Table 1

Transition rule	Transition molecule	Additional molecule	Restriction enzymes
$(q_0, a, \perp) \rightarrow (q_0, A\perp)$			*Bgl*I *Acu*I
$(q_0, a, A) \rightarrow (q_0, AA)$			*Bgl*I *Acu*I
$(q_0, b, A) \rightarrow (q_1, AA)$			*Bgl*I *Acu*I
$(q_1, b, A) \rightarrow (q_1, AA)$			*Bgl*I *Acu*I
$(q_1, c, A) \rightarrow (q_2, \varepsilon)$			*Bgl*I *Acu*I *Bbv*I *Sap*I
$(q_2, c, A) \rightarrow (q_2, \varepsilon)$			*Bgl*I *Acu*I *Bbv*I *Sap*I
$(q_2, \varepsilon, \perp) \rightarrow (q_3, \varepsilon)$			*Bgl*I *Acu*I *Bbv*I

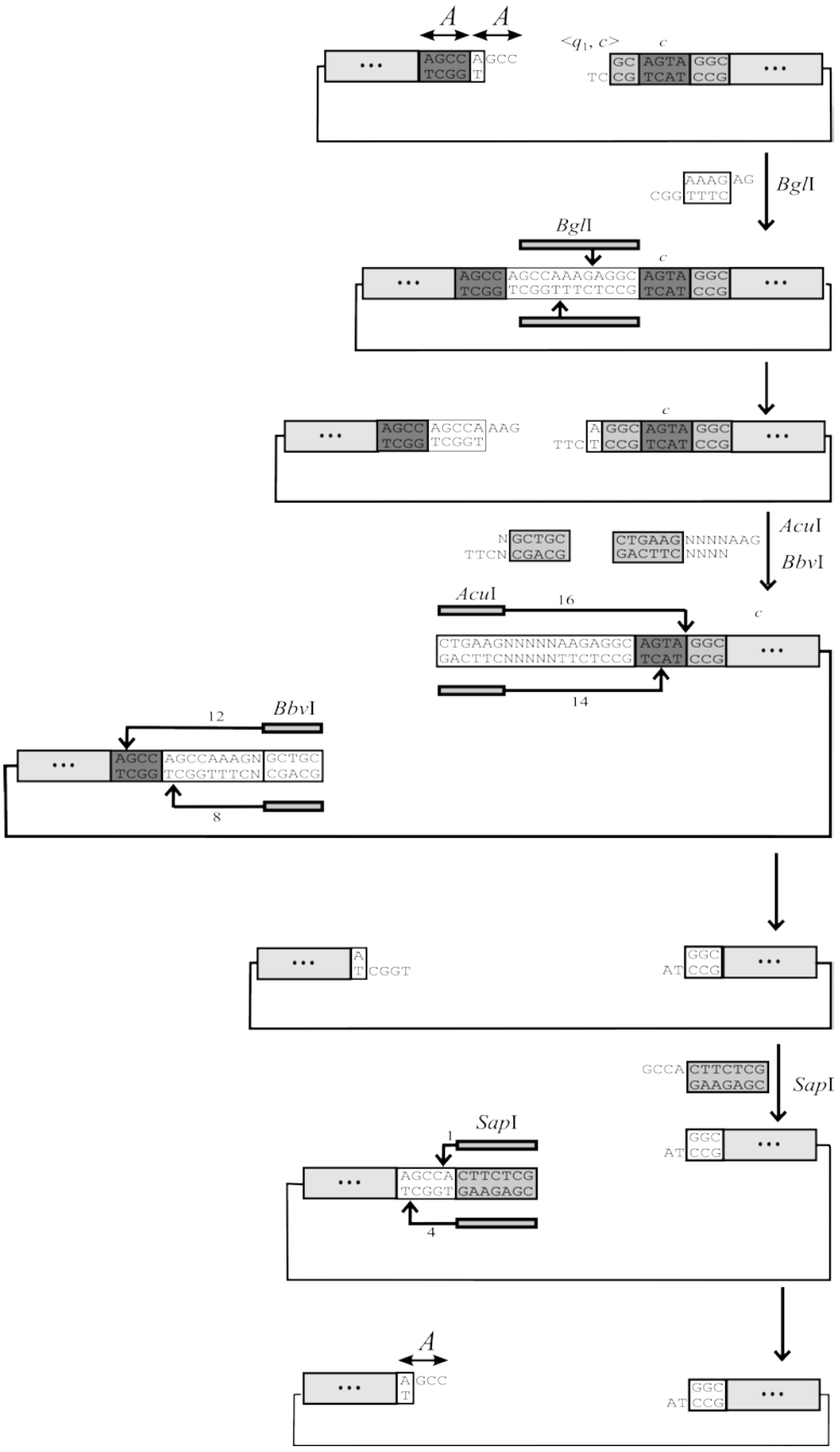
Appendix 2

The push a symbol on the stack $(q_0, a, \perp) \rightarrow (q_0, A \perp)$.



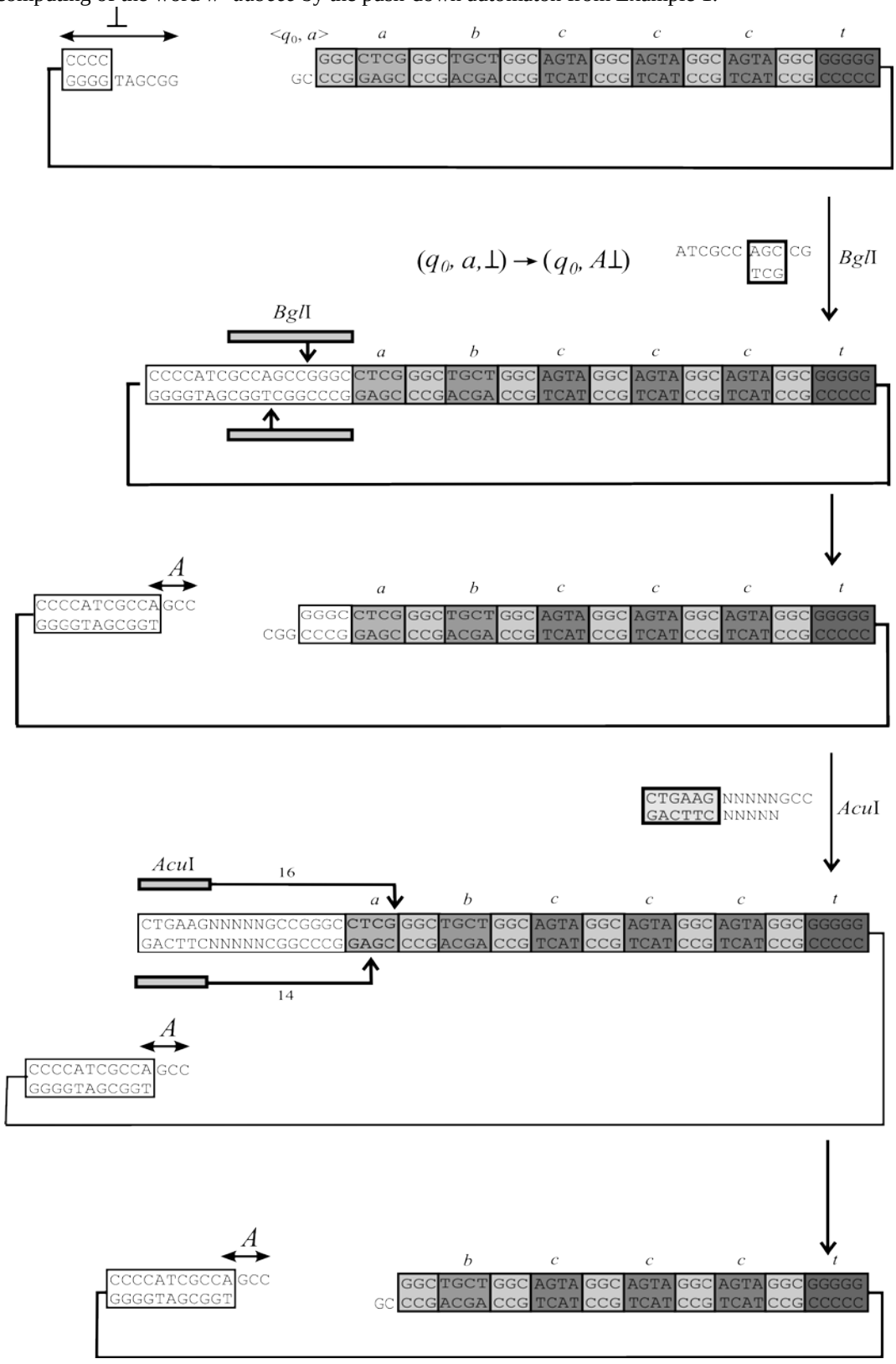
Appendix 3

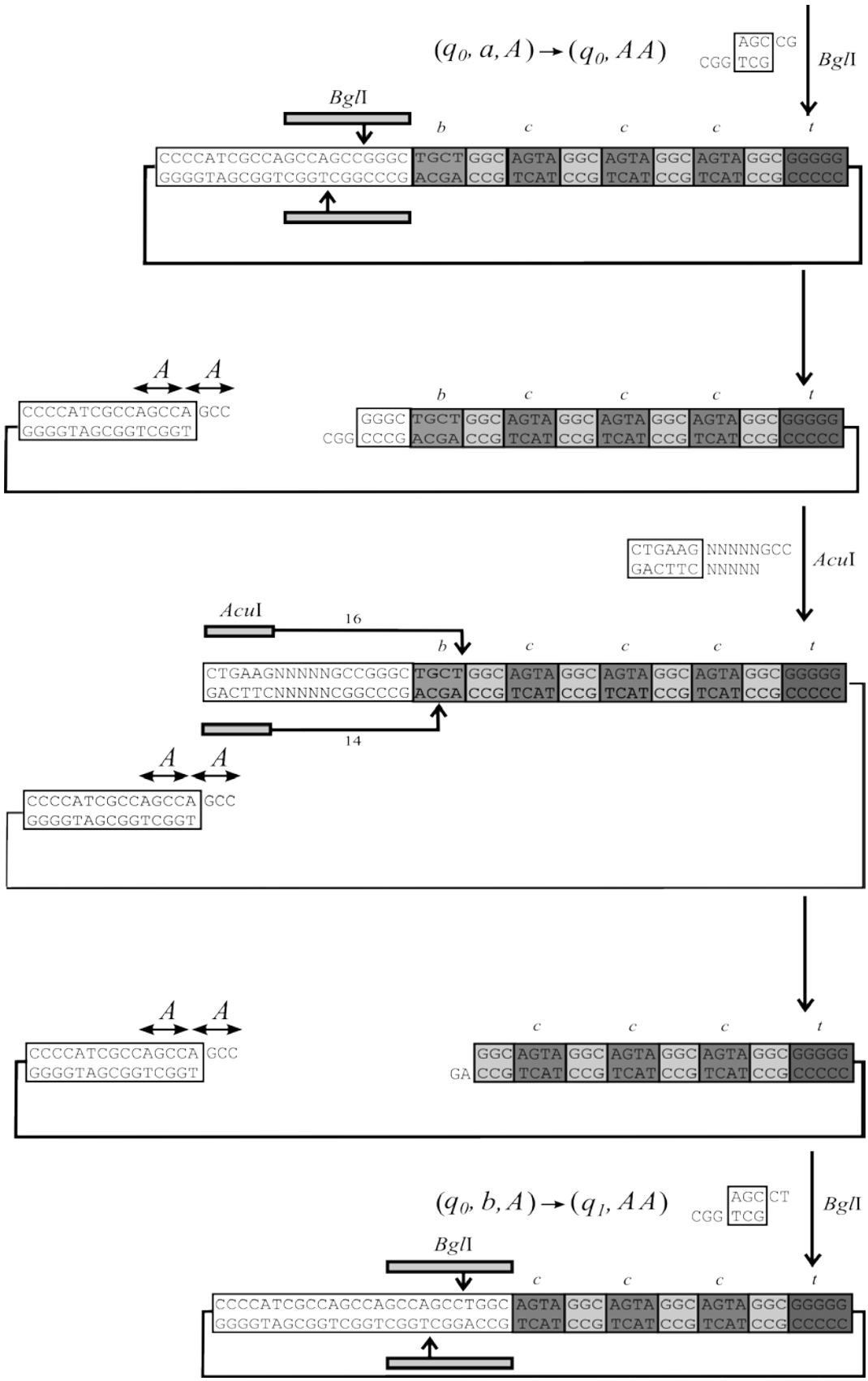
The pop a symbol from the stack $(q_1, c, A) \rightarrow (q_2, \varepsilon)$.

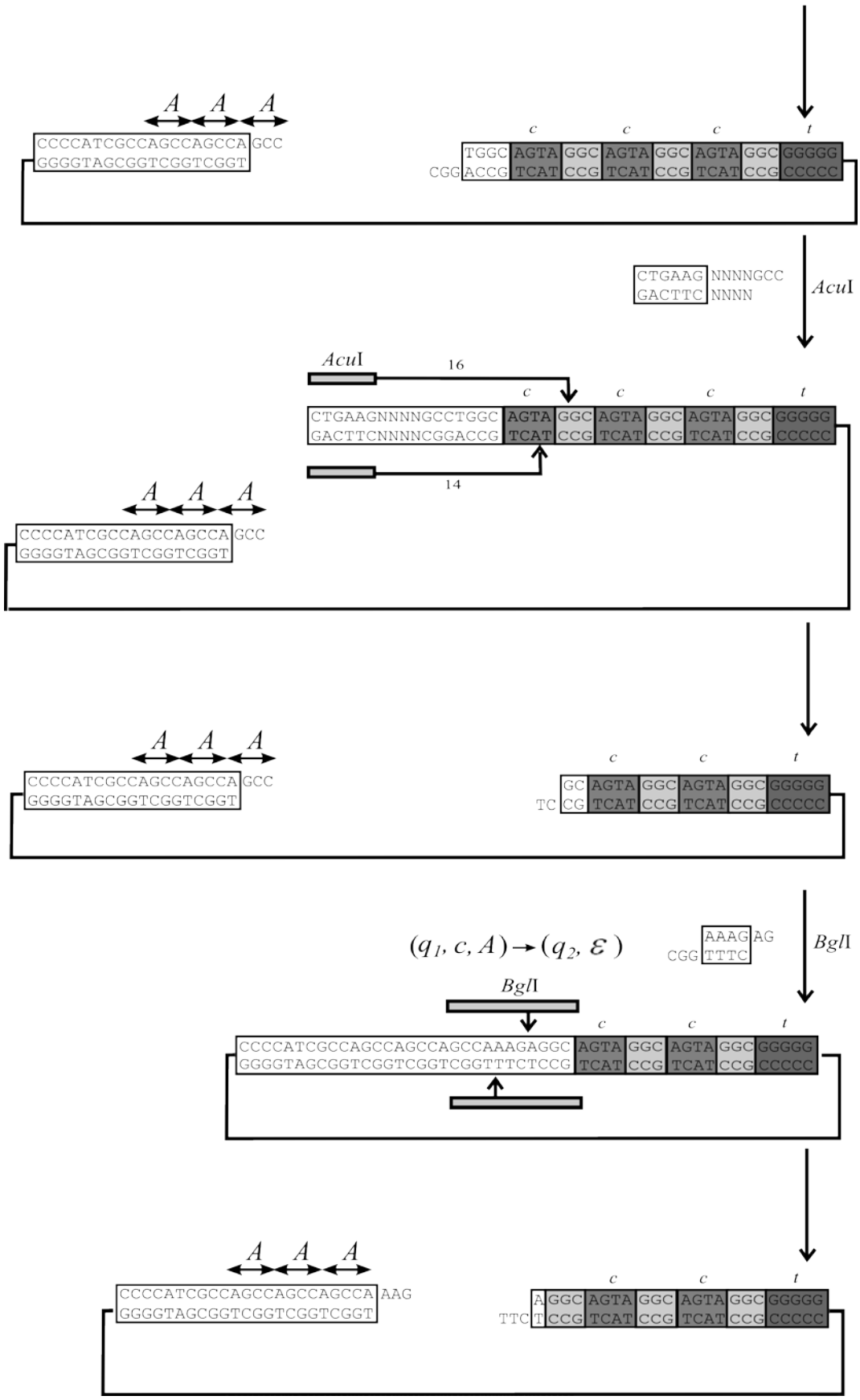


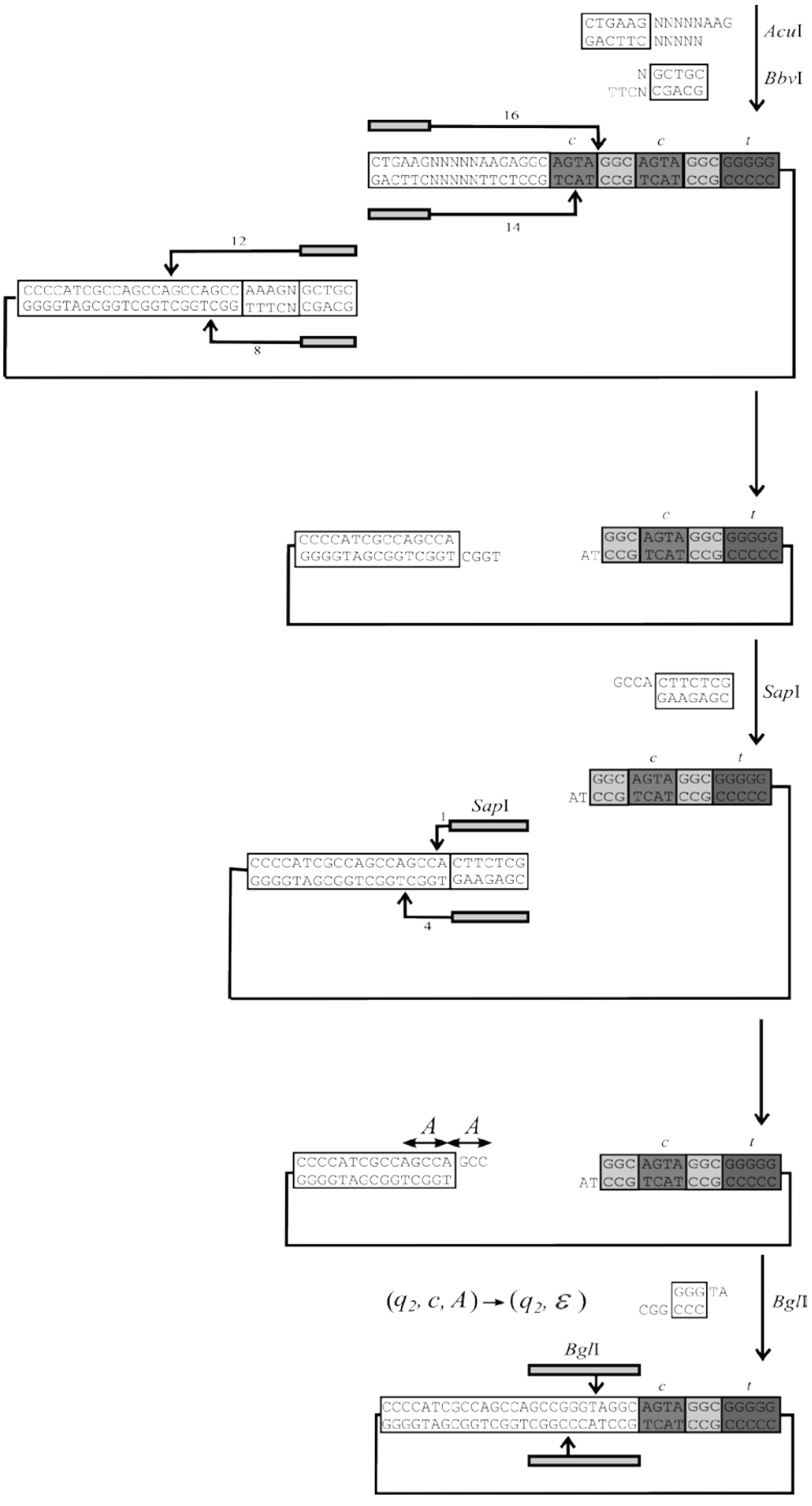
Appendix 4

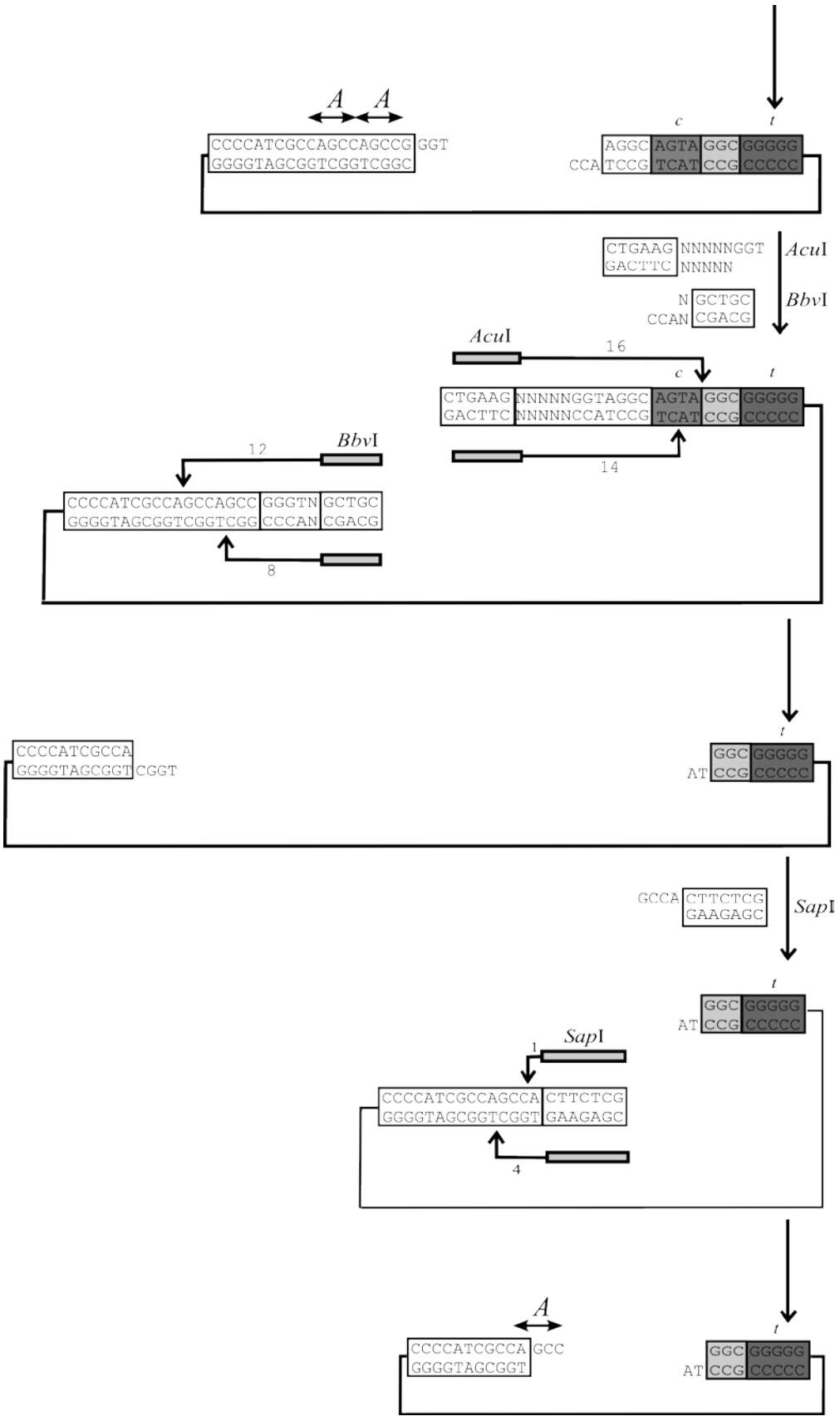
Process of computing of the word *w=aabccc* by the push-down automaton from Example 1.

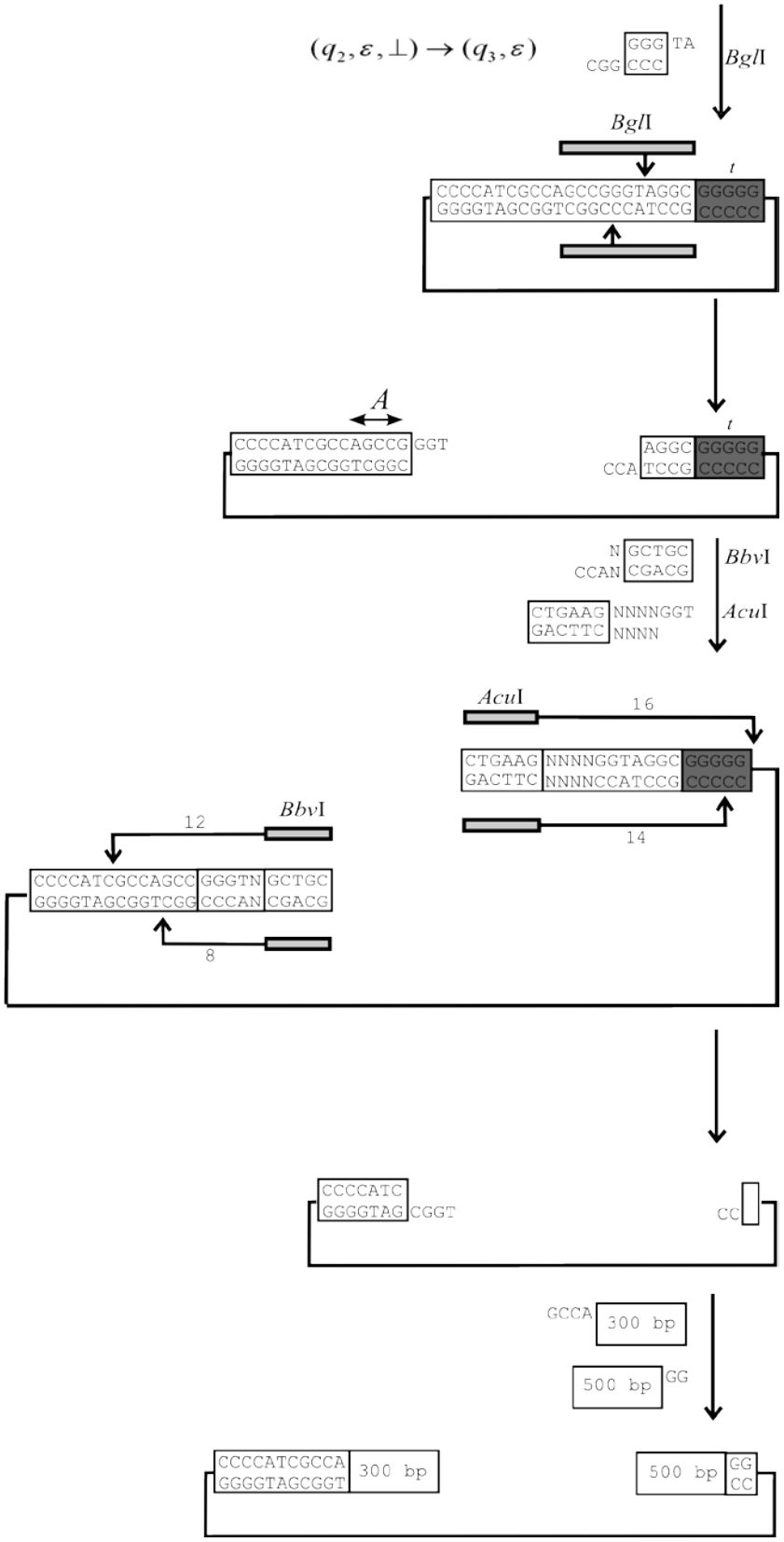



$(q_0, a, A) \rightarrow (q_0, AA)$
AGC CG
CGG TCG
BglI
BglI
b c c c t
CCCCATCGCCAGCCAGCCGGGC TGCT GGC AGTA GGC AGTA GGC AGTA GGC GGGGG
GGGGTAGCGGTCGGTCGGCCCG ACGA CCG TCAT CCG TCAT CCG TCAT CCG CCCCC
A A
CCCCATCGCCAGCCA GCC
GGGGTAGCGGTCGGT
b c c c t
GGGC TGCT GGC AGTA GGC AGTA GGC AGTA GGC GGGGG
CGG CCCG ACGA CCG TCAT CCG TCAT CCG TCAT CCG CCCCC
CTGAAG NNNNNGCC
GACTTC NNNNN
AcuI
AcuI
16
b c c c t
CTGAAGNNNNNGCCGGGC TGCT GGC AGTA GGC AGTA GGC AGTA GGC GGGGG
GACTTCNNNNNCGGCCCG ACGA CCG TCAT CCG TCAT CCG TCAT CCG CCCCC
14
A A
CCCCATCGCCAGCCA GCC
GGGGTAGCGGTCGGT
A A
CCCCATCGCCAGCCA GCC
GGGGTAGCGGTCGGT
c c c t
GA GGC AGTA GGC AGTA GGC AGTA GGC GGGGG
CCG TCAT CCG TCAT CCG TCAT CCG CCCCC
$(q_0, b, A) \rightarrow (q_1, AA)$
AGC CT
CGG TCG
BglI
BglI
c c c t
CCCCATCGCCAGCCAGCCAGCCTGGC AGTA GGC AGTA GGC AGTA GGC GGGGG
GGGGTAGCGGTCGGTCGGTCGGACCG TCAT CCG TCAT CCG TCAT CCG CCCCC


A A A
CCCCATCGCCAGCCAGCCA GCC
GGGGTAGCGGTCGGTCGGT
c c c t
TGGC AGTA GGC AGTA GGC AGTA GGC GGGGG
CGG ACCG TCAT CCG TCAT CCG TCAT CCG CCCCC
CTGAAG NNNNGCC
GACTTC NNNN
AcuI
AcuI
16
CTGAAGNNNNGCCTGGC AGTA GGC AGTA GGC AGTA GGC GGGGG
GACTTCNNNNCGGACCG TCAT CCG TCAT CCG TCAT CCG CCCCC
14
GC AGTA GGC AGTA GGC GGGGG
TC CG TCAT CCG TCAT CCG CCCCC
$(q_1, c, A) \rightarrow (q_2, \varepsilon)$
AAAG AG
CGG TTTC
BglI
BglI
CCCCATCGCCAGCCAGCCAGCCAAAGAGGC AGTA GGC AGTA GGC GGGGG
GGGGTAGCGGTCGGTCGGTCGGTTTCTCCG TCAT CCG TCAT CCG CCCCC
CCCCATCGCCAGCCAGCCAGCCA AAG
GGGGTAGCGGTCGGTCGGTCGGT
A GGC AGTA GGC AGTA GGC GGGGG
TTC T CCG TCAT CCG TCAT CCG CCCCC


CTGAAG NNNNNAAG
GACTTC NNNNN
AcuI
N GCTGC
TTCN CGACG
BbvI
16
c c t
CTGAAGNNNNNAAGAGCC AGTA GGC AGTA GGC GGGGG
GACTTCNNNNNTTCTCCG TCAT CCG TCAT CCG CCCCC
14
12
CCCCATCGCCAGCCAGCCAGCC AAAGN GCTGC
GGGGTAGCGGTCGGTCGGTCGG TTTCN CGACG
8
CCCCATCGCCAGCCA
GGGGTAGCGGTCGGT CGGT
AT CCG TCAT CCG CCCCC
GCCA CTTCTCG
GAAGAGC
SapI
SapI
1
CCCCATCGCCAGCCAGCCA CTTCTCG
GGGGTAGCGGTCGGTCGGT GAAGAGC
4
A A
CCCCATCGCCAGCCA GCC
GGGGTAGCGGTCGGT
(q2, c, A) → (q2, ε)
GGG TA
CGG CCC
BglI
BglI
CCCCATCGCCAGCCAGCCGGGTAGGC AGTA GGC GGGGG
GGGGTAGCGGTCGGTCGGCCCATCCG TCAT CCG CCCCC


A A
CCCCATCGCCAGCCAGCCG GGT
GGGGTAGCGGTCGGTCGGC
c t
AGGC AGTA GGC GGGGG
CCA TCCG TCAT CCG CCCCC
CTGAAG NNNNNGGT
GACTTC NNNNN
AcuI
N GCTGC
CCAN CGACG
BbvI
AcuI
16
c t
CTGAAG NNNNNGGTAGGC AGTA GGC GGGGG
GACTTC NNNNNCCATCCG TCAT CCG CCCCC
14
BbvI
12
CCCCATCGCCAGCCAGCC GGGTN GCTGC
GGGGTAGCGGTCGGTCGG CCCAN CGACG
8
t
CCCCATCGCCA
GGGGTAGCGGT CGGT
GGC GGGGG
AT CCG CCCCC
GCCA CTTCTCG
GAAGAGC
SapI
t
GGC GGGGG
AT CCG CCCCC
SapI
1
CCCCATCGCCAGCCA CTTCTCG
GGGGTAGCGGTCGGT GAAGAGC
4
A
CCCCATCGCCA GCC
GGGGTAGCGGT
t
GGC GGGGG
AT CCG CCCCC


$(q_2, \varepsilon, \perp) \rightarrow (q_3, \varepsilon)$
GGG TA
CGG CCC
BglI
BglI
t
CCCCATCGCCAGCCGGGTAGGC GGGGG
GGGGTAGCGGTCGGCCCATCCG CCCCC
A
CCCCATCGCCAGCCG GGT
GGGGTAGCGGTCGGC
t
AGGC GGGGG
CCA TCCG CCCCC
N GCTGC
CCAN CGACG
BbvI
CTGAAG NNNNGGT
GACTTC NNNN
AcuI
AcuI
16
CTGAAG NNNNGGTAGGC GGGGG
GACTTC NNNNCCATCCG CCCCC
14
BbvI
12
CCCCATCGCCAGCC GGGTN GCTGC
GGGGTAGCGGTCGG CCCAN CGACG
8
CCCCATC
GGGGTAG CGGT
CC
GCCA 300 bp
500 bp GG
CCCCATCGCCA 300 bp
GGGGTAGCGGT
500 bp GG
CC