Claudio Ferretti
Giancarlo Mauri
Claudio Zandron (Eds.)

LNCS 3384

DNA Computing

10th International Workshop
on DNA Computing, DNA10
Milan, Italy, June 2004, Revised Selected Papers

Springer

Claudio Ferretti   Giancarlo Mauri
Claudio Zandron (Eds.)

DNA Computing

10th International Workshop
on DNA Computing, DNA10
Milan, Italy, June 7-10, 2004
Revised Selected Papers

Springer

Volume Editors

Claudio Ferretti
Giancarlo Mauri
Claudio Zandron
Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
via Bicocca degli Arcimboldi 8, 20126, Milano, Italy
E-mail: {ferretti,mauri,zandron}@disco.unimib.it

Library of Congress Control Number: 2005927141

CR Subject Classification (1998): F.1, F.2.2, I.2.9, J.3

ISSN	0302-9743
ISBN-10	3-540-26174-5 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-26174-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper   SPIN: 11493785   06/3142   5 4 3 2 1 0

Preface

Biomolecular computing has emerged as an interdisciplinary field that draws together chemistry, computer science, mathematics, molecular biology, and physics. Our knowledge of DNA nanotechnology and biomolecular computing increases dramatically with every passing year. The International Meeting on DNA Computing has been a forum where scientists with different backgrounds, yet sharing a common interest in biomolecular computing, meet and present their latest results. Continuing this tradition, the 10th International Meeting on DNA Computing (DNA10) focused on the current experimental and theoretical results with the greatest impact.

The meeting took place at the University of Milano-Bicocca, Milan, Italy, from June 7 to June 10, 2004, and it was organized by the University of Milano-Bicocca and the Department of Informatics of the University of Milano-Bicocca. Papers and poster presentations were sought in all areas that relate to biomolecular computing, including (but not restricted to): demonstrations of biomolecular computing (using DNA and/or other molecules), theoretical models of biomolecular computing, biomolecular algorithms, computational processes in vitro and in vivo, analysis and theoretical models of laboratory techniques, biotechnological and other applications of DNA computing, DNA nanostructures, DNA devices such as DNA motors, DNA error evaluation and correction, in vitro evolution, molecular design, self-assembled systems, nucleic acid chemistry, and simulation tools.

Authors were asked to choose between two different tracks:

Track A — Full paper, for authors who wished to submit a full paper for presentation at DNA10 (oral or poster), and publication in the conference proceedings.

Track B — One-page abstract, for authors submitting experimental results, and who planned to submit their manuscript to a scientific journal, rather than publish it in the conference proceedings.

We received 67 submissions in track A and 27 in track B. Among them, 30 papers were selected for oral presentation. About 140 people attended the meeting.

The first day of the meeting, June 7, 2004, was dedicated to the following tutorials: N. Pavelka (Univ. of Milano-Bicocca), "Gene Expression Studies Using Microarrays," H.J. Hoogeboom (Leiden University), "Basic Concepts of Computing for Biologists," C. Henkel (Leiden University), "Basic Molecular Biology for Nonspecialists," and T.H. LaBean (Duke University), "Self-Assembly."

The next three days were devoted to invited plenary lectures and regular oral presentations. The invited plenary lectures were by K. Benenson (Weizmann Institute of Science, Israel), "An Autonomous Molecular Computer for Logical Control of Gene Expression," C. Flamm (University of Vienna, Austria), "Computational Design of Multi-stable Nucleic Acid Sequences," G. Păun (Institute of Mathematics of the Romanian Academy, Romania), "Membrane Computing — Power and Efficiency. An Overview," J. Reif (Duke University, USA), "DNA-Based Nano-engineering: DNA and Its Enzymes as the Engines of Creation at the Molecular Scale," and W.M. Shih (Harvard University, USA), "Clonable DNA Nanotechnology."

The editors would like to thank all contributors to and participants in the DNA10 conference, the Program Committee (A. Carbone, J. Chen, N. Jonoska, L. Kari, C. Mao, G. Mauri, G. Păun, J. Rose, P. Rothemund, Y. Sakakibara, N. Seeman, E. Shapiro, L. Smith, R. Weiss, and H. Yan), and the external reviewers.

Finally, we wish to thank Brainspark plc. Comerson, the Department of Informatics, Systems and Communications of the University of Milano-Bicocca, Etnoteam, the European Commission, STMicroelectronics, and the University of Milano-Bicocca for the support and sponsorship of the conference.

January 2005

Claudio Ferretti,
Giancarlo Mauri,
Claudio Zandron

Table of Contents

Computing by Observing Bio-systems: The Case of Sticker Systems

Artiom Alhazov[1,2] and Matteo Cavaliere[3]

[1] Research Group on Mathematical Linguistics,
Rovira i Virgili University,
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
artiome.alhazov@estudiants.urv.es

[2] Institute of Mathematics and Computer Science,
Academy of Sciences of Moldova,
Str. Academiei 5, Chişinău, MD 2028, Moldova
artiom@math.md

[3] Department of Computer Science and Artificial Intelligence,
University of Sevilla,
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
martew@inwind.it

Abstract. A very common approach in chemistry and biology is to observe the progress of an experiment, and take the result of this observation as the final output. Inspired by this, a new approach to computing, called system/observer, was introduced in [3].

In this paper we apply this strategy to sticker systems, [8, 11]. In particular we use finite automata (playing the role of observer) watching the "evolution" of a sticker system and translating such "evolution" into a readable output.

We show that this way of "computing by observing" brings us results quite different from the ones obtained when considering sticker systems in the standard manner. Even regular simple sticker systems (whose generative power is subregular) become universal when considered in this new framework. The significance of these results for DNA computing (by sticker systems) is briefly discussed.

1 Introduction: Observing Sticker Systems

A usual procedure in chemistry and biology is to observe the progress of an experiment, taking the result of observation as the output. Inspired by this a new approach to computing, called system/observer, has been introduced in [3].

There it was shown how a computing device can be constructed using two less powerful systems: the first one, which is a mathematical model of a biological system, "lives" (evolves), passing from one configuration to the next, producing in this way a "behavior"; the second system, called "observer", is placed outside and watches the biological system. Following a set of specific rules the observer translates the behavior of the underlying system into a "readable" output: it associates a label to each configuration of the bio-system and writes these labels according to their chronological order onto an output tape; in this way the pair composed by the biological system and the observer can be considered a computing (generating) device, as described in Figure 1.

This idea recalls a discussion by G. Rozenberg and A. Salomaa in [12]. They remarked that the result of a computation can be seen as already present in nature: we only need to look (in an appropriate way) at it. In their case this observation is made applying a (generalized) finite state sequential transducer to the so-called twin-shuffle language, a language closely related to the structure of DNA molecules. In our case the observer is applied not only to the final result, but to the entire evolution of the system. In other words, in our architecture, the computation is made by observing the full "life" of a biological system.

Until now, the system/observer architecture has been applied in different frameworks; in the first work, [3], the evolution of a membrane system (a formal model inspired by the functioning of the living cells) has been observed. In that paper it has been shown how the system composed of a "not powerful" membrane-system (with context-free power) and a finite state automaton in the role of observer, is universal. This can be considered the first (surprising) "hint" of the fact that computing by observing is a very powerful approach.

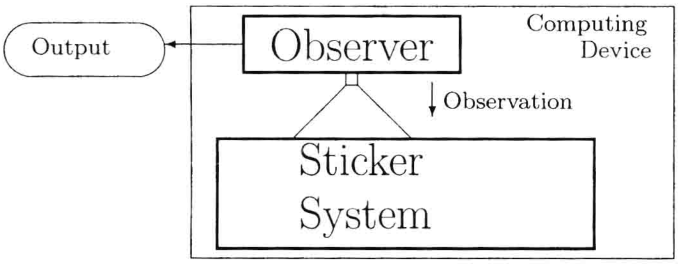

Fig. 1. Conceptual view of a sticker-system/observer architecture

In [5], a finite automaton observes the evolution of "marked" strings of a splicing system (a formal system inspired by the recombination of DNA strands that happens under the action of restriction enzymes). Also in this case, the observation adds much power to the considered bio-system. In particular, it has been shown that just observing the evolution of marked strings in a splicing system (using finite axioms and rules) it is even possible to obtain non-recursive languages (we recall that the generative power of this class of splicing systems, considered in the standard way, is subregular).

Finally, a more general application of the system/observer framework has been presented in [4]: the "evolution" of a grammar has been observed using a finite automaton. In this case, the universality is obtained using a finite state automaton observing a context-free grammar.

Here, we investigate *observable sticker-systems*, where the bio-system is a sticker system.

The main reason for investigating sticker systems in the system/observer framework comes from the fact that, using a recent lab-technique named *FRET*, [9], it is possible, under biologically relevant conditions, to observe the dynamics of a single molecule. Therefore we believe that it is extremely interesting to investigate how much we can compute just by observing the evolution of DNA molecules and sticker systems might represent an optimal way to formalize this investigation.

Sticker systems were introduced in [8] as a formal model of the operation of *annealing* (and *ligation*) operation that is largely used in DNA computing area, since the successful experiment of L.M. Adleman in 1994, [1]. The basic operation of a sticker system is the *sticking* operation that constructs double stranded sequences out of "DNA dominoes" (*polyominoes*) that are sequences with one or two sticky ends, or single stranded sequences, attaching to each other by *ligation* and *annealing.*

The informal idea of an observable sticker system can be expressed in the following way: an observer (for example, a microscope) is placed outside the "test tube", where (an unbounded number of copies of) DNA strands and DNA dominoes are placed together. Some of these molecules are marked (for example, with a fluorescent particle). The molecules in the solution will start to self-assemble (to stick to each other) and, in this way, new molecules are obtained The observer watches the evolution of the marked molecules and stores such evolution on an external tape in a chronological order.

For each possible "evolution" of the marked molecules a certain string is obtained. Collecting all the possible "evolutions" of such marked strands we obtain a language.

Many different variants of sticker systems can be considered, using different kinds of dominoes and different restrictions on the sticking operation (see details in [11]). In this paper we consider a very restricted and simple variant of sticker system, whose power is subregular, and we show that, when we consider such variant in the system/observer framework, then we get much more generative power and even universality.

2 Formal Language Pre-requisites

In what follows we suppose the reader familiar with basic notions of formal languages (as introduced, for instance, in [13]).

We will denote a finite set (the alphabet) by V, the set of words over V by V^*. For $x \in V^*$, $Pref(x) = \{y \in V^* \mid x = yz_2, z_2 \in V^*\}$, $Suff(x) = \{y \in V^* \mid x = z_1y, z_1 \in V^*\}$ and $Sub(x) = \{y \in V^* \mid x = z_1yz_2, z_1, z_2 \in V^*\}$ are the sets of all prefixes, suffixes and subwords of x, respectively.

A *shuffle* of words $x_1 \in T_1^*$ and $x_2 \in T_2^*$ $(T_1 \cap T_2 = \emptyset)$ is a word $y \in (T_1 \cup T_2)^*$ such that $h_{T_i}(y) = x_i$, $i \in \{1,2\}$, where h_{T_i} are the projection morphisms: $h_{T_i}(a) = a$ if $a \in T_i$ and λ otherwise, for $i \in \{1,2\}$.

By CF, CS, and RE we denote the classes of languages generated by context-free, context-sensitive, and unrestricted grammars respectively.

We shortly recall the basic notions of a *conditional grammar* used in the following theorem (for more details the reader can consult [6]).

A (context-free) *conditional grammar* is a construct $G = (N, T, P, S)$, where N and T are nonterminal and terminal symbols, S is the axiom and P is a finite set of rules of the form $(A \rightarrow \alpha, R)$, where $A \in N$, $\alpha \in (N \cup T)^*$ and R is a regular language over $N \cup T$. We say that $uAv \Rightarrow u\alpha v$ if there is a rule $(A \rightarrow \alpha, R) \in P$ such that $uAv \in R$.

For every language $L \in RE$ there exists a conditional grammar generating L. Without restricting generality we can assume that the rules of the grammar are binary ($|\alpha| \leq 2$ for every $(A \rightarrow \alpha, R) \in P$).

3 Preliminaries: Sticker Systems

We recall the basic notions of sticker systems. As it was already mentioned in the introduction, sticker systems can be considered a formal (language) model inspired by the annealing and ligation operations. The basic idea is to have initially DNA strands, called axioms, and dominoes that are DNA strands with sticky ends. Starting from the axioms and iteratively using the operation of sticking, complete double stranded sequences are obtained.

The collection of all the complete double stranded sequences obtained is the language generated by the sticker system.

Consider a symmetric relation $\rho \subseteq V \times V$ over V (of *complementarity*). Following [11], we associate with V the monoid $V^* \times V^*$ of pairs of strings. Because it is intended to represent DNA molecules, we also write elements $(x_1, x_2) \in V^* \times V^*$ in the form $\binom{x_1}{x_2}$ and $V^* \times V^*$ as $\binom{V^*}{V^*}$. We denote by $\begin{bmatrix} V \\ V \end{bmatrix}_\rho = \{\begin{bmatrix} a \\ b \end{bmatrix} \mid a, b \in V, (a, b) \in \rho\}$ the set of *complete double symbols*, and $WK_\rho(V) = \begin{bmatrix} V \\ V \end{bmatrix}_\rho^*$ is the set of the *complete double-stranded sequences (complete molecules)* also written as $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, where x_1 is the *upper strand* and x_2 is the *lower strand*.

As in [11], we use *single strands* – the elements of $S(V) = \binom{\lambda}{V^*} \cup \binom{V^*}{\lambda}$ and the molecules with (a possible) *overhang* on the right, which are the elements of $R_\rho(V) = \begin{bmatrix} V \\ V \end{bmatrix}_\rho^* S(V)$, from now on called *well-started* molecules (upper and lower strand are defined as in the case of complete molecules).

Given a well started molecule $u \in R_\rho(V)$ and a single strand $v \in S(V)$, we recall in Figure 2 the partial operation $\mu : R_\rho(V) \times S(V) \longrightarrow R_\rho(V)$ of sticking, as defined in [11]. We point out that we use a case of sticking, restricted to pasting a single strand to the right side of a well-started molecule (with a possible overhang on the right), corresponding to the *simple regular* sticker systems. Furthermore, we define length of a single strand

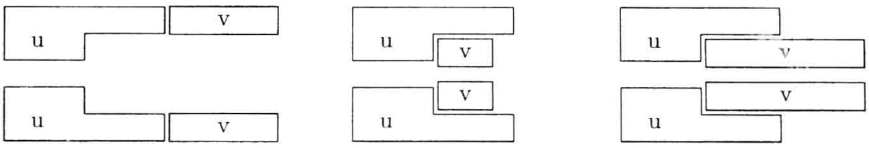

Fig. 2. Sticking operation

$u = \binom{x}{\lambda}$ (or $u' = \binom{\lambda}{x}$) as $|u| = |u'| = |x|$, and for a finite $H \subseteq S(V)$ we say $length(H) = max\{|u| \mid u \in H\}$.

A (simple regular) sticker system is a construct $\gamma = (V, \rho, A, D)$, where $A \subseteq R_\rho(V)$ is the (finite) set of axioms, and $D \subseteq S(V)$ is the (finite) set of *dominoes* (in this case these are single strands). Given $u, w \in R_\rho(V)$, we write $u \Rightarrow w$ iff $w = \mu(u, v)$ for some $v \in D$. A sequence $(w_i)_{1 \le i \le k} \subseteq R_\rho(V)$ is called a *complete computation* if $w_1 \in A$, $w_i \Rightarrow w_{i+1}$ for $1 \le i < k$ and $w_k \in WK_\rho(V)$.

The *language* generated by a sticker system γ is the set of upper strands of all complete molecules derived from the axioms. We remark the fact that the *family of languages generated by simple regular sticker systems is strictly included in the family of regular languages* (see [11] for the proof).

4 The Observer: Automata with Singular Output

For the observer (the "microscope") as described in the introduction we need a device mapping DNA molecules (also incomplete) into just one symbol.

For an alphabet V, our *double-symbol alphabet* constructed over V is

$$V_d = \begin{bmatrix} V \\ V \end{bmatrix}_\rho \cup \binom{V}{\lambda} \cup \binom{\lambda}{V} .$$

Therefore, following the idea also used in [3], we define a variant of finite state automata: the states are labeled by the symbols of the output alphabet Σ or with λ. Any computation of the automaton produces as output the label of the state it halts in (we are not interested in accepting computations and therefore do not consider the final states); because the observation of a certain string should always lead to a fixed result, we consider here only deterministic and complete automata.

An automaton with a singular output reads a molecule (element of $R_\rho(V)$) and outputs one symbol. Every well-started molecule in $R_\rho(V) \subseteq V_d^*$ is read, in a classical way, from left to right, scanning one double symbol from V_d at a time.

Formally, an automaton with singular output is a tuple $O = (Z, V_d, \Sigma, z_0, \delta, \sigma)$ with a state set Z, input alphabet V_d, initial state $z_0 \in Z$, and a complete transition function δ as known from conventional finite automata , that maps elements of $(V_d \times Z)$ into Z. Furthermore, there is the output alphabet Σ and a labeling function $\sigma : Z \longrightarrow \Sigma \cup \{\lambda\}$.

For a molecule $w \in R_\rho(V)$ and an automaton O we write $O(w)$ to indicate such output; for a sequence $w_1, \dots, w_n$ of $n \geq 1$ of molecules in $R_\rho(V)$ we write $O(w_1, \dots, w_n)$ for the string $O(w_1) \cdots O(w_n)$. For simplicity, in what follows, we present only the mapping defined by the observer without giving its real implementation as a finite automaton.

Moreover, we will also want the observer to be able to reject some words. To do this we simply choose a special symbol $\perp \notin \Sigma$ and an extended output alphabet $\Sigma_\perp = \Sigma \cup \{\perp\}$; σ then is a mapping from the set of states Z to $\Sigma_\perp \cup \{\lambda\}$. If a "bad" (not of interest) molecule is observed, then $\perp$ is produced and thus the entire sequence is to be rejected (hence, the criterion of rejecting is exactly the regular language, recognized by a finite automaton like O described above, but without output and with a single final state $\perp$). Then, using the intersection with the set Σ^*, it is possible to filter out the strings which contain $\perp$.

5 Observable Sticker Systems

An *observable sticker system* with output alphabet Σ is a construct $\phi = (\gamma, O)$, where γ is the sticker system with alphabet V, and O is the observer with input alphabet V_d constructed over V and with output alphabet Σ.

We denote the collection of all complete computations of ϕ by $\mathcal{C}(\phi)$. The language, over the output alphabet Σ, generated by an observable sticker system ϕ, is defined as $L(\phi) = \{O(s) \mid s \in \mathcal{C}(\phi)\}$. If we want to filter out the words that contain the special symbol $\perp$, then we consider the language $\widehat{L}(\phi) = L(\phi) \cap \Sigma^*$.

Here is a simple example that illustrates how an observable sticker system works. At the same time this example shows how one can construct an observable sticker system generating a non regular language (despite the fact that the power of simple regular sticker systems, when considered in the classical way, is subregular). Consider the following observable sticker system $\phi = (\gamma, O)$:

$$\gamma = (V = \{a, \mathbf{c}, \mathbf{g}, t\}, \rho = \{(a,t), (\mathbf{c},\mathbf{g}), (t,a), (\mathbf{g},\mathbf{c})\}, A = \{\begin{bmatrix} a \\ t \end{bmatrix}\}, D),$$

$$D = \{\begin{pmatrix} a \\ \lambda \end{pmatrix}, \begin{pmatrix} \lambda \\ t \end{pmatrix}, \begin{pmatrix} \mathbf{c} \\ \lambda \end{pmatrix}, \begin{pmatrix} \lambda \\ \mathbf{g} \end{pmatrix}\},$$

with the observer O defined by the following mapping:

$$O(w) = \begin{cases} b, & \text{if } w \in \begin{bmatrix} a \\ t \end{bmatrix}^* \begin{pmatrix} a^* \\ \lambda \end{pmatrix} \cup \begin{pmatrix} \lambda \\ t^* \end{pmatrix}, \\ d, & \text{if } w \in \begin{bmatrix} a \\ t \end{bmatrix}^* \begin{pmatrix} a^*\mathbf{c} \\ \lambda \end{pmatrix} \cup \begin{pmatrix} \lambda \\ t^*\mathbf{g} \end{pmatrix}, \\ \lambda, & \text{otherwise.} \end{cases}$$

The language generated by γ is $L_1 = \{b^m d^n \mid m \geq n, m \geq 1, n \geq 0\} \notin REG$.

Below is an example of computation of ϕ (generating $bbbbdd$):

Step	0	1	2	3	4	5	6
Added		$\binom{a}{\lambda}$	$\binom{a}{\lambda}$	$\binom{\lambda}{t}$	$\binom{\mathbf{c}}{\lambda}$	$\binom{\lambda}{t}$	$\binom{\lambda}{\mathbf{g}}$
Molecule	a / t	aa / t	aaa / t	aaa / tt	$aaa\mathbf{c}$ / tt	$aaa\mathbf{c}$ / ttt	$aaa\mathbf{c}$ / $ttt\mathbf{g}$
Output	b	b	b	b	d	d	λ

The idea of the system ϕ is the following: think of symbols $\mathbf{c}$, $\mathbf{g}$ as "markers". While we stick to the current molecule either $\binom{a}{\lambda}$ or $\binom{\lambda}{t}$, the observer maps the result (a molecule without markers) to b. As soon as we attach to the current molecule a marker, the observer maps the resulting molecule to d, until the strand with a marker is extended or until the molecule is completed.

Suppose that, when the first marker is attached, the length of the strand with that marker is l_1, the length of the other strand is l_2 (clearly, $l_1 > l_2$), and then the output produced so far is $b^{l_1+l_2-2}d$. To complete the molecule by extending the strand without the marker, we need to attach $l_1 - l_2$ symbols to it, and in this case the observer outputs $d^{l_1-l_2-1}\lambda$. Thus, the resulting string x consists of $l_1 + l_2 - 2$ b's and $l_1 - l_2$ d's. Since $l_2 \geq 1$, the difference between the number of b's and the number of d's is $l_1 + l_2 - 2 - (l_1 - l_2) = 2l_2 - 2 \geq 0$. (Recall that in case we attach a symbol to a string with the marker, the observer only outputs λ, so the inequality $m = |x|_b \geq |x|_d = n$ remains valid, and all the combinations (m, n), $m \geq n$ are possible). Hence, $L(\gamma) = L_1$.

6 Small Observable Sticker Systems

The previous example is a preliminary "hint" on how, observing a sticker system, we can get more power with respect to the case when sticker systems are considered in the classical way.

The idea of the previous example can be extended and it is possible to show that there exist observable (simple regular) sticker systems, generating non-context-free languages, even using dominoes of length 1. In other words, the "simple" observation of the evolution of the sticker system permit us to "jump" from a subclass of regular language to non-context-free languages.

Theorem 1. *There exists an observable sticker system $\phi = (\gamma, O)$, $\gamma = (V, \rho, A, D)$, length$(D) = 1$ such that $L(\phi) \notin CF$.*

Proof. Consider the following observable sticker system $\phi = (\gamma, O)$:

$$\gamma = (V = \{a, b, c\}, \rho = \{(a,a), (b,b), (c,c)\}, A = \{\begin{bmatrix} c \\ c \end{bmatrix}\}, D),$$

$$D = \{\binom{a}{\lambda}, \binom{\lambda}{a}, \binom{b}{\lambda}, \binom{\lambda}{b}, \binom{\mathbf{c}}{\lambda}, \binom{\lambda}{\mathbf{c}}\}$$

with the observer O defined by the following mapping,

$H_1 = \begin{bmatrix} c \\ c \end{bmatrix} \begin{pmatrix} U^*a \\ \lambda \end{pmatrix}$, $H_2 = \begin{bmatrix} c \\ c \end{bmatrix} \begin{pmatrix} U^*b \\ \lambda \end{pmatrix}$, $H_3 = \begin{bmatrix} c \\ c \end{bmatrix} \begin{pmatrix} U^*\mathbf{c} \\ \lambda \end{pmatrix}$,

$H_4 = \begin{bmatrix} cU^*a \\ cU^*a \end{bmatrix} \begin{pmatrix} U^*\mathbf{c} \\ \lambda \end{pmatrix}$, $H_5 = \begin{bmatrix} cU^*b \\ cU^*b \end{bmatrix} \begin{pmatrix} U^*\mathbf{c} \\ \lambda \end{pmatrix}$, $H_6 = \begin{bmatrix} cU^*\mathbf{c} \\ cU^*\mathbf{c} \end{bmatrix}$

$O(w) = a$ if $w \in H_1 \cup H_4$, $O(w) = b$ if $w \in H_2 \cup H_5$, $O(w) = c$ if $w \in H_3 \cup H_6$, $O(w) = \lambda$, otherwise. , U={a,b}

The language generated by γ is $L_2 = \bigcup_{x \in U^*} \Big(xc \cdot Pref(xc) \cup Pref(x) \cdot Sub(xc) \Big)$. Notice that $L_2 \cap U^*cU^*c = \{xcxc \mid x \in U^*\} \notin CF$, and hence $L_2 \notin CF$.

The computation of the system starts from the axiom $\begin{bmatrix} c \\ c \end{bmatrix}$ (at this point we can consider both strands "empty"), and pieces ("symbols") from D can be adjoined to the strands of the axiom during the computation. When a complete molecule is obtained, the computation stops. To understand the explanation, think of **c** as a marker.

While the marker is not added to the upper strand and the lower strand is "empty" (for molecules of the form H_1 or H_2), the observer outputs, one by one, the symbols added to the upper strand. After some symbol is added to the lower strand, the symbols added to the upper strand are not output anymore (i.e., the observer outputs λ).

As soon as the system adds **c** to the upper strand (for the molecules of the form H_4 or H_5), the observer starts to output the symbols that are adjoined to the lower strand.

If, at some step, a symbol is added to the upper (or lower) strand to the right of the marker **c**, then, starting from such step, the observer will not produce any input anymore.

We can distinguish three main cases in the way the system ϕ works. We can get the string $s = xcxc$ by first adding the symbols of $x\mathbf{c}$ to the upper strand until the marker **c** is adjoined (letting the observer to output xc, symbol by symbol), and then adding the symbols of $x\mathbf{c}$ to the lower strand (letting the observer to output xc again). The observer cannot guarantee that, first the upper strand is completed, and then the lower strand is completed. Therefore, strings different from $xcxc$ can also be generated.

The system ϕ can produce strings in the set $xc \cdot Pref(xc)$ in the following case: suppose the upper strand is completed (obtaining $cx\mathbf{c}$) and the lower strand is being completed; before it finishes, a symbol might be added to the upper strand, at the right of the marker **c**. Starting from this step the observer will output λ until the computation halts.

On the other hand, the system ϕ can also generate strings in the set $Pref(x) \cdot Sub(xc)$. The symbols corresponding to a prefix of x are added to the upper strand (the observer produces $Pref(x)$ as output of this phase). At some step, some symbols (i.e., a prefix of $x\mathbf{c}$) are added to the lower strand, and during this phase the observer outputs λ. At some time the upper strand is completed and **c** is added (during this phase no output is produced because the lower strand is not empty).