Computing by Observing Bio-Systems: the Case of Sticker Systems

Artiom ALHAZOV[1,2], **Matteo CAVALIERE**[1]
[1] Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tárraco 1, 43005 Tarragona, Spain
E-mail: {artiome.alhazov,matteo.cavaliere}@estudiants.urv.es

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

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 [2].

In this paper we apply this strategy to sticker systems, [7, 9]. 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, and then take the result of this observation as the final output. Inspired by this a new approach to computing, called system/observer, has been introduced in [2].

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 [10]. 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, [2], 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.

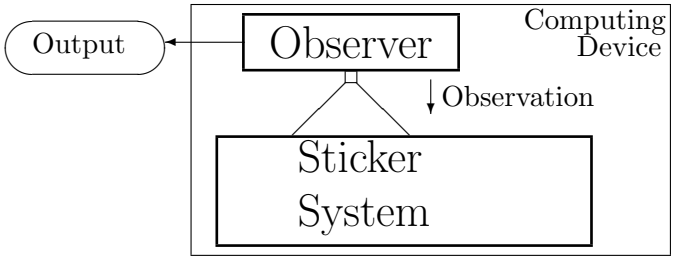

Figure 1: Conceptual view of a sticker-system/observer architecture

In [4], 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 [3]: 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.

Sticker systems were introduced in [7] 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 [9]). 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 Preliminaries: Sticker Systems

In this section 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. In what follows we suppose the reader familiar with basic notions of formal languages (as introduced, for instance, in [11]).

Consider an alphabet V and a symmetric relation $\rho \subseteq V \times V$ over V (of *complementarity*). Following [9], 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 [9], we use *single strands*, which are 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).

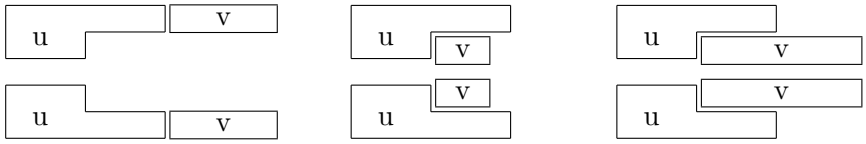

Figure 2: Sticking operation

Given a well started molecule $x \in R_\rho(V)$ and a single strand $y \in S(V)$, we recall in Figure 2 the partial operation $\mu : R_\rho(V) \times S(V) \longrightarrow R_\rho(V)$ of sticking, as defined in [9]. 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 $u = \binom{x}{\lambda}$ (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*. Given $u, v \in R_\rho(V)$, we write $u \Rightarrow w$ iff $w = \mu(u, v)$ for some $v \in D$. A sequence $(w_i)_{1 \leq i \leq k} \subseteq R_\rho(V)$ is called a *complete computation* if $w_1 \in A$, $w_i \Rightarrow w_{i+1}$ for $1 \leq i < k$ 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* (for the proof of this result the reader can consult [9]).

3 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.

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

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

Therefore, following the idea also used in [2], 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 (the elements in $R_\rho(V)$) and outputs a singular 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 $w_1, \ldots, w_n$ of $n \geq 1$ of molecules in $R_\rho(V)$ we write $O(w_1, \ldots, w_n)$ 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 molecule is observed, then $\perp$ is produced and thus the entire sequence is to be rejected. Then, using the intersection with the set Σ^*, it is possible to filter out the strings which contain the special symbol $\perp$.

4 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 $\phi = (\gamma, O)$:

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

with the observer O defined by the following mapping:

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

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		$\begin{pmatrix} a \\ \lambda \end{pmatrix}$	$\begin{pmatrix} a \\ \lambda \end{pmatrix}$	$\begin{pmatrix} \lambda \\ t \end{pmatrix}$	$\begin{pmatrix} \mathbf{c} \\ \lambda \end{pmatrix}$	$\begin{pmatrix} \lambda \\ t \end{pmatrix}$	$\begin{pmatrix} \lambda \\ \mathbf{g} \end{pmatrix}$
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 $\begin{pmatrix} a \\ \lambda \end{pmatrix}$ or $\begin{pmatrix} \lambda \\ t \end{pmatrix}$, 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 $d^{l_1-l_2-1} \cdot \lambda$. 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, $L(\gamma) = L_1$.

5 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 5.1 *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 $\phi = (\gamma, O)$:

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

$$D = \{\begin{pmatrix} a \\ \lambda \end{pmatrix}, \begin{pmatrix} \lambda \\ a \end{pmatrix}, \begin{pmatrix} b \\ \lambda \end{pmatrix}, \begin{pmatrix} \lambda \\ b \end{pmatrix}, \begin{pmatrix} \mathbf{c} \\ \lambda \end{pmatrix}, \begin{pmatrix} \lambda \\ \mathbf{c} \end{pmatrix}\}, U = \{a, b\}$$

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.

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 stands of the axiom during the computation. When a complete molecule is obtained, the computation stops. To understand the explanation, think of $\mathbf{c}$ 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 $\mathbf{c}$ 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 $\mathbf{c}$, 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 $\mathbf{c}$ 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 $\mathbf{c}$. 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 $\mathbf{c}$ is added (during this phase no output is produced because the lower strand is not empty).

Starting from a certain step, new symbols are added to the lower strand, obtaining $cx\mathbf{c}$. Because, during this phase, a symbol might be added at any step to the right of the marker in the upper strand (thus stopping the output), the string produced during this phase is in $Sub(xc)$. Hence, the full output is in the set $Pref(x) \cdot Sub(xc)$.

Therefore, we have shown that the language generated by ϕ is exactly L_2. □

6 Using an Observer with Rejection: Universality

After Theorem 6.1 it is natural to ask which is the class of sticker systems that is universal when observed by a finite state automaton. Somehow expected, from Theorem 6.1 to get universality we do not need "complicated" sticker systems but simple regular sticker systems with dominoes

of length at most 4 suffice. On the other hand we need to use an observer that is able to discard any "bad" evolution, as the one described in Section 3.

Theorem 6.1 *For each $L \in RE$ there exists an observable sticker system $\phi = (\gamma, O)$, $\gamma = (V, \rho, A, D)$, with $length(D) \leq 4$ such that $\widehat{L}(\phi) = L$ (See appendix for the proof.)*

Using theorem 6.1, the definition of $\widehat{L}(\phi)$ and the fact that recursive languages are closed under intersection with regular languages, we get:

Corollary 6.1.a *There exists an observable sticker system $\phi = (\gamma, O)$, $\gamma = (V, \rho, A, D)$, with $length(D) \leq 4$ such that $L(\phi)$ is a non-recursive language.*

In other words, also if we do not discard any evolution of the observed sticker system then we still get something not recursive; that is really surprising because intuitively the observer's ability to reject bad evolution could seem a powerful and essential feature to get something not "trivial".

7 Concluding Remarks and Research Proposals

In this paper we have presented a new way to look at the generation of languages in the framework of sticker systems. We have applied the system/observer architecture, introduced in [2], to the sticker systems and, in particular, we have introduced the class of observable sticker systems.

In an observable sticker system we have a simple regular sticker system and an observer (that is a finite state automaton with singular output) that watches the "evolution" of the molecule, producing as output a label at each step of the computation. We have shown that the combination of a sticker system with an observer can be very powerful even using simple components; in fact, "observing" a *simple regular* sticker system with elementary (i.e., of length 1) dominoes it is possible to obtain even some non context-free languages (the family of languages generated, in the standard way, by such kind of sticker systems is subregular). If we use a more "clever" observer, able to reject "bad" computations, then we get the universality just using simple regular sticker systems with dominoes of length 4.

Such results have a clear significance for DNA computing. Sticker systems are theoretical models of the annealing operation essentially used in many DNA computing experiments, starting with the pioneering one of Adleman. Simple regular simple systems - hence of the same kind as those corresponding to the annealing operation from Adleman experiment - generate only regular languages. Observing their evolution by other finite state devices leads, surprisingly, to universality. Informally speaking, "simple" experiments, observed in a clever manner by "simple" tools can thus compute whatever much more complex processes can compute.

Many problems have been left *open*: we have considered a very restricted (and then interesting) kind of observable sticker systems, where the underlying sticker system uses only dominoes of length 1: we do not know what is the lower bound on the generative capacity of such class. Can we get all the regular languages? We conjecture that the answer to question is yes. Can every context-free language can be generated?

Due to Corollary 6.1.a, we know that, also without rejecting any computation, just observing simple sticker systems with dominoes of length 4, it is possible to get non-recursive languages. Is it possible to get every RE language? A positive answer to this question would be quite surprising. Moreover, can the length of the dominoes used in Theorem 6.1 be decreased? (and then, what is the minimal length to get universality?)

These research suggestions that we have shortly presented are only some of the ones that the reader can "extract" from our paper; we believe that other interesting results can be found in this direction of research.

Acknowledgements. The first author acknowledges IST-2001-32008 project "MolCoNet" and also the Moldovan Research and Development Association (MRDA) and the U.S. Civilian Research and Development Foundation (CRDF), Award No. MM2-3034 for providing a challenging and fruitful framework for cooperation.

References

[1] L.M. Adleman, Molecular computation of solutions to combinatorial problems, *Science*, 226, 1994, 1021–1024.

[2] M. Cavaliere, P. Leupold, Evolution and Observation – a New Way to Look at Membrane Systems. In: C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenber, A. Salomaa (eds.): Membrane Computing, Lecture Notes in Computer Science 2933, Springer, 2004, 70–88.

[3] M. Cavaliere, P. Leupold, Evolution and Observation - A Non-Standard Way to Generate Formal Languages, Theoretical Computer Science, accepted.

[4] M. Cavaliere, N. Jonoska, (Computing by) Observing Splicing Systems, submitted, 2004.

[5] J. Dassow, Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, Heidelberg, 1989.

[6] R. Freund, Bidirectional Sticker Systems and Representations of RE Languages by Copy Languages, In: Gh. Păun (ed.) *Computing with Bio-Molecules: Theory and Experiments*, Springer-Verlag, Singapore, 1998, 182–199.

[7] L. Kari, Gh. Păun, G. Rozenberg, A. Salomaa, S. Yu, DNA computing, sticker systems, and universality, *Acta Informatica*, 35, 5 (1998), 401–420.

[8] J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[9] Gh. Păun, G. Rozenberg, A. Salomaa, *DNA Computing - New Computing Paradigms*, Springer-Verlag, Berlin, 1998.

[10] G. Rozenberg, A. Salomaa, *Watson-Crick Complementarity, Universal Computations and Genetic Engineering*, Technical Report 96-28, Dept. of Computer Science, Leiden University, 1996.

[11] A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.

8 Appendix – On Universality

Theorem 6.1 (For each $L \in RE$ there exists an observable sticker system $\phi = (\gamma, O)$, $\gamma = (V, \rho, A, D)$, $length(D) \leq 4$ such that $\widehat{L}(\phi) = L$)

Proof. (sketch) For a given language $L \in RE$ there exists a (binary) *conditional* grammar $G = (N, T, P, S)$, generating L. We use the following notations: E, F, Z, M_1, M_2 are new symbols, $U = N \cup T \cup \{E, F, Z, M_1, M_2\}$, $U' = \{X' \mid X \in U\}$, $U'' = \{X'' \mid X \in U\}$, $U_0'' = U'' - \{M_1, M_2, E, F\}$, $U_1 = U \cup U' - \{M_1, M_2, M_1', M_2'\}$, $U_2 = U'' \cup Lab(P) - \{M_1'', M_2''\}$. We associate distinct labels to the productions in P, we write the set of all labels as $Lab(P)$. For every rule $(r : B \to x)$ we define

$$cod(r) = \begin{cases} Z', & \text{if} \quad x = \lambda, \\ C', & \text{if} \quad x = C \in (N \cup T), \\ B_1' M_1' M_2' B_2', & \text{if} \quad x = B_1 B_2 \in (N \cup T)(N \cup T). \end{cases}$$

We construct the following observable sticker system $\phi = (\gamma, O)$:

$$\begin{aligned}
\gamma &= (V = U \cup U' \cup U'' \cup Lab(P), \rho, A = \{\begin{bmatrix} E \\ E'' \end{bmatrix} \begin{pmatrix} M_1 M_2 S M_1 M_2 E \\ \lambda \end{pmatrix}\}, D), \\
\rho &= \{(X, X''), (X', X''), (X'', X), (X'', X') \mid X \in U\} \\
&\cup \{(B, r), (r, B), (B', r), (r', B) \mid (r : B \to x) \in P\}, \\
D &= \{\begin{pmatrix} B \\ \lambda \end{pmatrix}, \begin{pmatrix} B' \\ \lambda \end{pmatrix}, \begin{pmatrix} \lambda \\ B'' \end{pmatrix} \mid B \in U\} \cup \{\begin{pmatrix} cod(r) \\ \lambda \end{pmatrix}, \begin{pmatrix} \lambda \\ r \end{pmatrix} \mid r \in Lab(P)\}.
\end{aligned}$$

We consider a morphism $h : U \cup U' \to U$ defined for every $x \in U$ as

$$h(x) = h(x') = \begin{cases} \lambda, & \text{if} \quad x \in \{M_1, M_2, Z\}, \\ x, & \text{otherwise.} \end{cases}$$

To describe the observer, we define a set H of "molecule blocks" (representing the sentential forms of G)

$$\begin{aligned}
H &= \{\begin{bmatrix} x_1 E \\ x_2 E'' \end{bmatrix} \mid \exists (r : B \to x, R) \in P : h(x_1) \in R, \\
&\quad x_2 \in (M_1'' M_2'' (Lab(P) \cup U_0''))^* \cap U''^* r U''^*\},
\end{aligned}$$

Finally, the mapping of observer O is given below.

$$O(w) = \begin{cases}
\lambda, & \text{if} \quad w \in \begin{bmatrix} E \\ E'' \end{bmatrix} H^* \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{pmatrix} x_3 E x_4 \\ \lambda \end{pmatrix} \text{ and} \\
& \quad ((Suff(Ex_4), Suff(Ex_1)) \in U_1 \times \{M_1''\} \cup \{M_1, M_1'\} \times \{M_1''\} \\
& \quad \cup \{M_1, M_1'\} \times \{M_2''\} \cup \{M_2, M_2'\} \times \{M_2''\} \cup \{M_2, M_2'\} \times U_2 \\
& \quad \cup \{(X, X'') \mid X \in U - \{M_1, M_2\} \\
& \quad \cup \{(cod(r), r) \mid (r : B \to x, R) \in P\}\}, \\
t(X), & \text{if} \quad w \in \begin{bmatrix} E \\ E'' \end{bmatrix} H^* \begin{bmatrix} x_1 X \\ x_2 \end{bmatrix} \begin{pmatrix} x_3 F \\ \lambda \end{pmatrix} \text{ and } x_1 X x_2 \in (U \cup U' - N)^*, \\
\lambda, & \text{if} \quad w \in \begin{bmatrix} E \\ E'' \end{bmatrix} H^* \begin{bmatrix} (U \cup U' - N)^* F \\ (U'')^* F'' \end{bmatrix}, \\
\perp, & \text{otherwise,}
\end{cases}$$

where $t : U \cup U' \longrightarrow T$ is a morphism defined as $t(X) = \lambda$ if $X \notin T \cup T'$ and $t(X) = t(X') = X$ if $x \in T$.

We now proceed to the explanations of the construction above.

Simulating G. γ assembles the molecule representing the concatenation of sentential forms of the derivation in G. Symbol by symbol, we copy the current sentential form, applying some production once (copying or rewriting a symbol takes six steps). O checks the regular condition, rejecting "incorrect" molecules.

Symbols used. In the concatenation of sentential forms, E represents the separator; F marks the final one; Z represents erased symbols. M_1 and M_2 are the spacers, used to synchronize the extension of the upper and lower strands. N and T are symbols of G.

The union of all the above is denoted by U. In the upper strand, the symbols in U are the ones copied from the previous sentential form and the symbols in U' represent the result of rewriting some non terminal symbols in the previous sentential form. In the lower strand, the symbols determine the future behavior: those in U'' are to be copied, while the ones in $Lab(P)$ represent the production to be applied.

Regular condition. A sentential form x is represented on the upper strand by x (shuffled with Z^*, with some symbols primed), where each symbol is preceded by the spacers M_1 and M_2 (h is a morphism "recovering" x).

We call a "block" a double strand encoding a sentential form, as described above. By H we denote the set of "good blocks": the lower strand contains exactly one symbol from $Lab(P)$ - one rewriting rule is applied; the upper strand satisfies the regular condition, associated to this rule.

The sticker system. Extending the upper strand corresponds to writing a new sentential form, while extending the lower one corresponds to reading the current sentential form. The overhang of the current molecule represents the unread part of the current sentential form together with already produced part of the new one.

To the upper strand we can adjoin the symbols of U (if copied from the previous sentential form), or codes of the right-hand sides of rules in G (defined by the function *cod*). To the lower strand we add the symbols of U'' (for copying) or the labels of the rules (for applying them).

The observer checks a few conditions, rejecting the result of the computation (by writing $\perp$) if they are not satisfied.

The "correct evolution" assumes that the lower strand is extended first, and a strand is never extended twice in a row before F is placed. So, the observer requires that the spacers (from $S_1 = \{M_1, M_1'\}$, from $S_2 = \{M_2, M_2'\}$) and non-spacers (from U_1) alternate in the upper strand (of the form $E(S_1S_2U_1)^*(\{\lambda\} \cup S_1 \cup S_1S_2)$); M_1'', M_2'' and those from U_2, respectively, in the lower strand. If the synchronization fails, then the last symbols of the strands will either be in $S_1 \times U_2$, or in $S_2 \times \{M_1''\}$, or in $U_1 \times \{M_2''\}$, and the result is rejected.

After we "read" a symbol, we either copy it, or rewrite it by some production in G. The observer checks that what we write in the upper strand corresponds to what we read in the lower strand.

It is also the duty of the observer to check the correctness of the blocks: for each non-final sentential form x exactly one rule of G is applied, and that x respects the regular condition, associated to this rule.

When we arrive to the terminal sentential form, F is added to the upper strand. Starting from this step, only the lower strand is extended, and the observer outputs the result by applying the morphism t to the symbols being complemented. □

The following example shows the technique used in the previous theorem. **Example.** Consider a conditional grammar $G = (\{S\}, \{a\}, \{(r_1 : S \to aS, Sa^*), (r_2 : S \to \lambda, a^*S)\}, S)$, we illustrate the simulation of the derivation $S \to aS \to a$. The observer outputs the symbol a in step 33, and λ in other cases, halting at step 39.

Step	0							2	4	6				8	10	12
Upper	$\mathbf{E}$	$\mathbf{M_1}$	$\mathbf{M_2}$	$\mathbf{S}$	$\mathbf{M_1}$	$\mathbf{M_2}$	$\mathbf{E}$	M_1	M_2	$\mathbf{a'}$	$\mathbf{M_1'}$	$\mathbf{M_2'}$	$\mathbf{S'}$	M_1	M_2	E
Lower	$\mathbf{E''}$	M_1''	M_2''	r_1	M_1''	M_2''	E''	M_1''	M_2''	a''	M_1''	M_2''	r_2	M_1''	M_2''	E''
Step	0	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29

Step	14	16	18	20	22	24	26	28	30
Upper	M_1	M_2	a	M_1	M_2	$\mathbf{Z'}$	M_1	M_2	F
Lower	M_1''	M_2''	a''	M_1''	M_2''	Z''	M_1''	M_2''	F''
Step	31	32	33	34	35	36	37	38	39