The Microsoft Research - University of Trento Centre for Computational and Systems Biology

Technical Report CoSBi 11/2006

DNA Splicing: Computing by Observing

Matteo Cavaliere

Microsoft Research – University of Trento,
Centre for Computational and Systems Biology,
Trento, Italy

cavaliere@cosbi.eu

Nataša Jonoska

Department of Mathematics
University of South Florida,
Tampa, FL 33620, USA

jonoska@math.usf.edu

Peter Leupold

Research Group on Mathematical Linguistics,
Rovira i Virgili University,
Tarragona, Spain

klauspeter.leupold@estudiants.urv.es

This is the preliminary version of a paper that will appear in
Natural Computing, 8,1, 2009
available at
http://www.springerlink.com

Abstract

Motivated by several techniques for observing molecular processes in real-time we introduce a computing device that stresses the role of the observer in biological computations and that is based on the observed behavior of a splicing system. The basic idea is to introduce a marked DNA strand into a test tube with other DNA strands and restriction enzymes. Under the action of these enzymes the DNA starts to splice. An external observer monitors and registers the evolution of the marked DNA strand. The input marked DNA strand is then accepted if its observed evolution follows a certain expected pattern. We prove that using simple observers (finite automata), applied on finite splicing systems (finite set of rules and finite set of axioms), the class of recursively enumerable languages can be recognized.

1 Introduction: (Bio)Accepting Devices

Several techniques for monitoring the dynamics and changes of a single DNA molecule within a given biomolecular process have been developed recently. For instance, a well established methodology is the *FRAP*, fluorescent recovery after photobleaching. Other known methodologies are *FRET*, fluorescence resonance energy transfer, and fluorescent correlation spectroscopy *FCS* [17]. A survey on observation techniques for biomolecular dynamics with their advantages and disadvantages can be found in [12]. Usually these techniques can be used to observe only three different colors in fluorescent microscope, but it is possible to obtain more colors by *multiplexing*, as suggested in [11].

One of the recent ways to mark (and then, to observe) single DNA molecules is represented by *quantum dots*; with this technique it is possible to tag individual DNA molecules; in other words they can be used like fluorescent biological labels, as suggested by [2, 8]. A more current review on the use of quantum dots in vivo imaging can be found in [14].

In many techniques the study of biomolecular dynamics is divided in two separate phases (see for ex. [12]): registration of the dynamics (on a special support like channels of data) and then investigation of the collected data. Inspired by these techniques we present a theoretical computational model using an “observer” and a “decider” as two independent devices. A variation of this evolution/ observation strategy, was initially introduced [6] in a formal computing model inspired by the functioning of living cells, known as membrane systems.

Since then, the evolution/observation idea has been considered in different formal models of biological systems [1, 4, 7]. In all these consideratioins the underlying idea is that a *generative* device is constructed by using two systems: a mathematical model of a biological system that “lives” (evolves) and an observer that watches the entire evolution of this system and translates it into a readable output.

The main idea of this approach is that the computation is obtained by observing the entire evolution (in time) of a biological system. In [5], the evolution/observation strategy is used to construct an *accepting* device. There, it is suggested that it is possible to imagine any biological system as an accepting device. This is achieved by taking a model of a biological system, introducing an input to such a system and observing its evolution. If the evolution of the system is of an expected type, (for example follows a regular predetermined pattern) the input is accepted by the (bio)system, otherwise it can be considered rejected.

An external *observer* is fundamental in extracting a more abstract, formal behavior from the evolution of the biological system. A *decider* is the machine that checks whether the behavior of the biological system is of the expected type.

Splicing systems belong to a formal model of recombination of double stranded DNA molecules (for simplicity we call them DNA strands) under the action of a ligase and restriction enzymes (endonucleases), [10]. The main purpose of this paper is to illustrate the accepting strategy of observer/decider to splicing systems. For the motivations and background on splicing systems we refer to the original paper [10] or to the corresponding chapter in [16].

In [4] an observer was associated to splicing systems to construct a *generative device*. Here we construct an *accepting device* by joining a *decider* to the observer of the splicing system. We call such a system *Splicing Recognizer* (in short, SR). A schematic view of the model is depicted in Figure 1.

The SR works in the following way. An input *marked* DNA strand (represented by a string w) is inserted in a test tube. Due to the presence of restriction enzymes, the input strand changes, as it starts to recombine with other DNA strands in the test tube. A sequence of *intermediate* marked DNA strands is generated. This constitutes the evolution of the input marked DNA strand. Schematically this is presented with the sequence of w, w', w'', w''' in Figure 1.

The external observer associates to each intermediate marked strand a certain label (i.e., a meaning) taken from a finite set of possible labels. It writes these labels onto an output tape in their chronological order. In Figure 1 this corresponds to the string $a_1a_2a_3a_4$. This string represents a code of the obtained evolution. When the marked strand becomes of a certain predetermined "type" the observation stops.

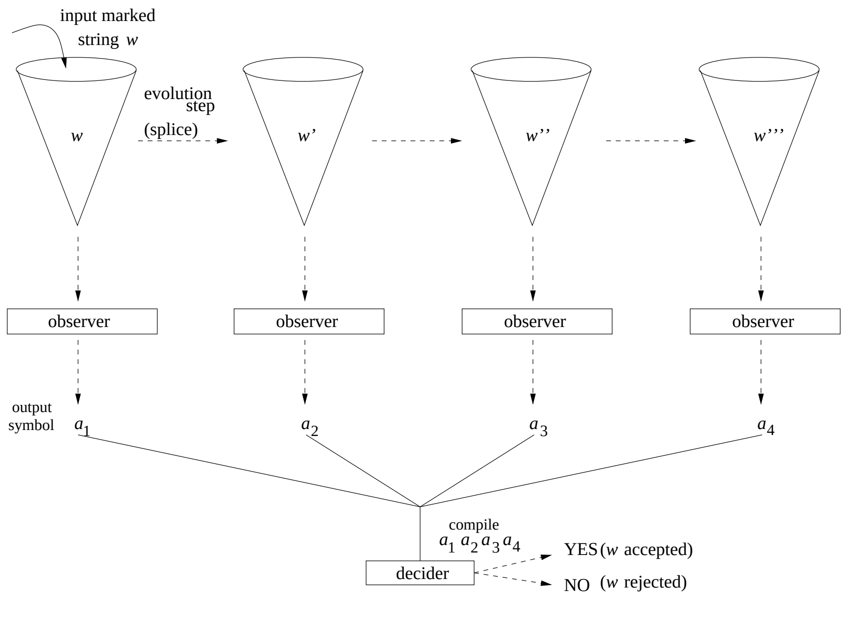

Figure 1: The splicing/observer architecture.

At this point the decider checks if the entire evolution of the input marked DNA strand described by the string $a_1a_2a_3a_4$ has followed a certain pattern, i.e. if it is in a certain language. If this is true, the input string w is accepted by the SR; otherwise it is considered to be rejected.

In this paper we show that using this strategy, it is possible to obtain computationally universal accepting systems even in the case when simple components are used.

For instance, we show that having just a finite state automaton as observer of the evolution of a finite splicing system (with a finite set of splicing rules) is already enough to simulate a Turing machine. This is a remarkable jump in acceptance power since it is well known that a finite splicing system by itself can generate only a subclass of the class of regular languages. The results are not surprising, since by putting extra control with the decider, the computational power of the whole system increases. Similar results, but in the generative sense, were obtained without the decider in [4] but these required a special observation of a right-most evolution, which is not the case with the results presented here.

The general idea of the evolution/observation strategy outlined above recalls what was already discussed by G. Rozenberg and A. Salomaa [18], who remarked that the result of a computation is already present in nature – we only need to look (in an appropriate way) at it. While in their case the observation is made by applying a *gsm* machine to the language obtained using the (biologically inspired) *twin-shuffle* operation, in our framework the observer (as well as the decider) is not applied to the final result, but rather to the entire evolution of the system.

2 Splicing Recognizer: Definition

In what follows we use basic concepts from formal language theory. For more details on this subject the reader should consult the standard books in the area, for instance, [19, 20].

Briefly, we fix the notations used here. We denote a finite set (the alphabet) by V, the set of words over V by V^*. By REG, CF, CS, and RE we denote the classes of languages generated by regular, context-free, context-sensitive, and unrestricted grammars respectively.

2.1 Splicing and marked strings

As an underlying biological system we consider a splicing system (more precisely an H scheme, following the terminology used in [16]).

First we recall some basic notions concerning splicing systems. However, in what follows, we suppose the reader is already familiar with this subject, as for instance, presented in [16].

Consider an alphabet V (splicing alphabet) and two special symbols $\#$ and $\$$ not in V. A splicing rule (over V) is a string of the form $u_1\#u_2\$u_3\#u_4$, where $u_1, u_2, u_3, u_4 \in V^*$.

For a splicing rule $r = u_1\#u_2\$u_3\#u_4$ and strings $x, y, z_1, z_2 \in V^*$ we write $(x, y) \Longrightarrow_r (z_1, z_2)$ iff $x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2,\ z_1 = x_1u_1u_4y_2,\ z_2 = y_1u_3u_2x_2$. We refer to z_1 (z_2) as the first (second) string obtained by applying the splicing rule r.

An H scheme is a pair $\sigma = (V, R)$ where V is an alphabet, and $R \subseteq V^*\#V^*\$V^*\#V^*$ is a set of splicing rules. For a given H scheme $\sigma = (V, R)$ and a language $L \subseteq V^*$ we define $\sigma(L) = \{z_1, z_2 \in V^* \mid (x, y) \Longrightarrow_r (z_1, z_2)$, for some $x, y \in L,\ r \in R\}$.

When restriction enzymes (and a ligase) are present in a test tube, their action does not terminate after a single cut and paste operation, but it continues such that the content of the tube is changed iteratively. Hence, given an *initial language* $L \subseteq V^*$ and an H scheme $\sigma = (V, R)$ we define the iterated splicing as: $\sigma^0(L) = L,\ \ \sigma^{i+1}(L) = \sigma^i(L) \cup \sigma(\sigma^i(L)), i \geq 0$.

We are interested in observing the evolution of an *input marked string* until the string does not belong to a specific target language. A *sequence of marked strings* for an initial language L, a *target language* $L_t \subseteq V^*$, an *input marked string* $w \in L, w \notin L_t$, and an H scheme σ, is any sequence of strings $\langle w_0, w_1, \cdots, w_k\rangle$, $k > 0$, $w_i \in V^*, 0 \leq i \leq k$, such that:

- $w_0 = w$; the first string of the sequence is the input marked string.

- For $0 \leq i \leq k-1$, there is $y \in \sigma^i(L)$ such that $(w_i, y) \Longrightarrow_r (w_{i+1}, z)$, $r \in R$ or $(y, w_i) \Longrightarrow_r (w_{i+1}, z)$, $r \in R, z \in V^*$; each new marked string is obtained by splicing the previous marked string.
- $w_k \in L_t$ and for $0 \leq i < k$, $w_i \notin L_t$; the sequence ends as soon as the current marked string is in the target language.

Note that for $0 \leq i \leq k$, we have that $w_i \in \sigma^i(L)$. Moreover, we always consider that each result of splicing two words consists of an ordered pair of words and we take the first string from this pair as the new marked string.

Because of the non-determinism in applying splicing rules, it is possible to have different sequences of marked strings for a given initial language L, an input marked string w, a target marked language L_t, and an H scheme σ. *The collection of all the possible sequences of evolution of a marked string w is denoted by $\sigma(w, L, L_t)$.*

Radius of a splicing system

For a splicing rule $r = u_1\#u_2\$u_3\#u_4$ we denote by $rad(r)$ the length of the longest string u_1, u_2, u_3, u_4; we say that this is the *radius* of r. The radius of an H scheme is the maximal radius of its rules.

2.2 Observer

The observer is a device mapping arbitrarily long strings into just one symbol. As in [7] we use a special variant of finite automata with some feature known from Moore machines: the set of states is labelled with the symbols of an output alphabet Σ. Any computation of the automaton produces as output the label of the state it halts in (we are not interested in accepting / not-accepting computations and therefore also not interested in the presence of final states). The observation of a certain string should always lead to a fixed result, so only deterministic and complete automata are considered.

Formalizing this, a *monadic transducer* is a tuple $O = (Z, V, \Sigma, z_0, \delta, l)$ with state set Z, input alphabet V, initial state $z_0 \in Z$, and a complete deterministic transition function δ as known from conventional finite automata; further there is the output alphabet Σ and a labelling function $l : Z \to \Sigma$. The output of the monadic transducer is the label of the state it stops in. For a string $w \in V^*$ and a transducer O we then write $O(w)$ for this output; for a sequence $\langle w_1, \ldots, w_n\rangle$ of $n \geq 1$ strings over 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 regular partitions defined by the observers without giving detailed implementations for them.

2.3 Decider

The *Decider* is a devices accepting a language over the output alphabet Σ of the corresponding observer as just introduced. For this we rely on conventional finite automata with input alphabet Σ. The output of the decider consists of a simple `yes` or `no`. For decider D, and input word $u \in \Sigma^*$, the output is denoted with $D(u)$.

2.4 Splicing Recognizer

Putting together the components just defined in the way informally described in the Introduction, a *splicing recognizer* (in short *SR*) is a quintuple $\Omega = (\sigma, O, D, L, L_t)$; $\sigma = (V, R)$ is a finite H scheme, O is an observer $(Z, V, \Sigma, z_0, \delta, l)$, D is a decider with input alphabet Σ, L and L_t are finite languages, respectively, the initial and the target marked language for σ.

The *language accepted by SR* Ω is the set of all words $w \in V^*$ for which there exists a sequence $s \in \sigma(w, L, L_t)$ such that $D(O(s)) =$ `yes`; formally

$$L(\Omega) := \{\, w \in V^* \mid \exists s \in \sigma(w, L, L_t)\, [D(O(s)) = \texttt{yes}]\,\}.$$

Radius of a splicing recognizer

Given an SR $\Omega = (\sigma, O, D, L, L_t)$ the radius of Ω is the radius of the employed H scheme σ.

3 Going over Regular

It is well-known that the family of languages generated by splicing systems using only a finite set of splicing rules and a finite initial language is strictly included in the family of regular languages (see, e.g., [16]).

In the following theorem we show that an SR composed by an H scheme with a finite set of rules, finite initial language, finite target marked language and finite state automata as observer and decider, can recognize non regular languages.

Theorem 3.1 *There is an SR Ω of radius ≤ 2 such that $L(\Omega)$ is a non regular, context-free language.*

Proof.

We construct an SR recognizing the language $\{o_l a^n b^n o_r \mid n \geq 0\}$ that is known to be non-regular. The SR $\Omega = (\sigma, O, D, L, L_t)$ is defined as follows: the H scheme is $\sigma = (V, R)$, with

$$\begin{array}{rl} V = \{o_l, o_r, a, b, a', b', X_1, Y_1, X_2, Y_2\} \text{ and} & \\ R = \{ \;\; r_1 \;:\; \#bo_r\$X_2\#b'o_r, & \\ r_2 \;:\; o_la'\#Y_2\$o_la\#, & . \\ r_3 \;:\; \#b'o_r\$X_1\#o_r, & \\ r_4 \;:\; o_l\#Y_1\$o_la'\#\} & \end{array}$$

The symbols o_l, o_r indicate the left and the right beginning of the string that is to be accepted, the symbols a', b' are help symbols such that the splicing would remove the appearance of a's and b's one at the time and ones after the other, and in that fashion check whether the count of a's and b'sis the same. The symbols X_1, X_2, Y_1, Y_2 are symbols that appear in words of the initial language such that the splicing is facilitated.

The initial language has four words: $L = \{X_2b'o_r, o_la'Y_2, X_1o_r, o_lY_1\}$ such that only one rule can be applied to each string. The target marked language is $L_t = \{o_lo_r\}$.

The observer O has input alphabet V and output alphabet $\Sigma = \{l_0, l_1, l_2, l_3, \perp\}$ and implements the following mapping:

$$O(w) = \begin{cases} l_0 & \text{if } w \in o_la^*b^*o_r, \\ l_1 & \text{if } w \in o_la^*b^*b'o_r, \\ l_2 & \text{if } w \in o_la'a^*b^*b'o_r, \\ l_3 & \text{if } w \in o_la'a^*b^*o_r, \\ \perp & \text{else.} \end{cases}$$

The decider D is a finite state automaton, with input alphabet Σ, that gives a positive answer exactly if a word belongs to the regular language $l_0(l_1l_2l_3l_0)^*$. Note that the first word in the splicing evolution (marked string) has to be in $o_l(a^*b^*)o_r$ as l_0 is the first symbol of every string accepted by D. If $w \notin o_l(a^*b^*)o_r$ then the SR does not accept w i.e., $w \notin L(\Omega)$.

The observer checks whether the splicing rules are applied in the order r_1, r_2, r_3, r_4, and this corresponds to remove, in an alternating way, a b from the right and an a from the left of the input marked string. Rule r_1 changes the suffix bo_r of w into $b'o_r$. Then rule r_2 changes the prefix o_la into o_la' and rules r_3 and r_4 remove b' and a' respectively. In this way, at least one of the evolutions of the input marked string is of the kind accepted by the decider if, and only if, the input marked string is in the language $\{o_la^nb^no_r \mid n \geq 0\}$.

To clarify the operation of the SR Ω we show the acceptance of the input marked string $w_0 = o_laabbo_r$. For simplicity, we only show the evolution of the input marked string and the output of the observer, step by step.

Step	**splicing rule applied**	**new marked string**	**observer map**
0		$w_0 = o_laabbo_r$	$O(w_0) = l_0$
1	$(o_laabbo_r, X_2b'o_r) \Longrightarrow_{r_1} (o_laabb'o_r, X_2bo_r)$	$w_1 = o_laabb'o_r$	$O(w_1) = l_1$
2	$(o_laabb'o_r, o_la'Y_2) \Longrightarrow_{r_2} (o_la'abb'o_r, o_laY_2)$;	$w_2 = o_la'abb'o_r$	$O(w_2) = l_2$
3	$(o_la'abb'o_r, X_1o_r) \Longrightarrow_{r_3} (o_la'abo_r, X_1b'o_r)$;	$w_3 = o_la'abo_r$	$O(w_3) = l_3$
4	$(o_la'abo_r, o_lY_1) \Longrightarrow_{r_4} (o_labo_r, o_la'Y_1)$;	$w_4 = o_labo_r$	$O(w_4) = l_0$
5	$(o_labo_r, X_2b'o_r) \Longrightarrow_{r_1} (o_lab'o_r, X_2bo_r)$;	$w_5 = o_lab'o_r$	$O(w_5) = l_1$
6	$(o_labb'o_r, o_la'Y_2) \Longrightarrow_{r_2} (o_la'b'o_r, o_laY_2)$;	$w_6 = o_la'b'o_r$	$O(w_6) = l_2$
7	$(o_la'b'o_r, X_1o_r) \Longrightarrow_{r_3} (o_la'o_r, X_1b'o_r)$;	$w_7 = o_la'o_r$	$O(w_7) = l_3$
8	$(o_la'o_r, o_lY_1) \Longrightarrow_{r_4} (o_lo_r, o_la'Y_1)$;	$w_8 = o_lo_r$	$O(w_7) = l_0$

Obviously the entire observed evolution $l_0l_1l_2l_3l_0l_1l_2l_3l_0$ belongs to the language accepted by the decider D, so the string w_0 is accepted by the SR Ω.

□

4 Going over Context-Free

An SR can accept even non context-free languages as stated in the following theorem. The trick used here consists in the rotation of symbols in the input marked string, during its evolution. The regular observer controls whether this kind of rotation is done in a correct way. In this case the radius of the system is ≤ 3.

Theorem 4.1 *There is an SR Ω of radius ≤ 3 such that $L(\Omega)$ is a non context-free, context-sensitive language.*

Proof. We construct an SR Ω accepting the non context-free language
$\{o_l w o_r \mid w \in \{a,b,c\}^+, |w|_a = |w|_b = |w|_c\}$. The idea is similar to the one used in Theorem 3.1.

The SR $\Omega = (\sigma, O, D, L, L_t)$ is defined as follows: the H scheme is $\sigma = (V, R)$, with $V = \{a,b,c,o_l,o_r,X_1,X_2,X_3,X_4,X_5,X_6,X_a,X'_a,X_b,X'_b,X_c,X'_c\}$. As above, the symbols o_l and o_r are the left and the right "end of string" indicators. Symbols $X_1 - X_6$ are symbols used in the words of the initial language that initiate the splicing, and symbols $X_a - X'_c$ are symbols that are used to search for appearances of a, b and c and then to remove (by splicing) these symbols one by one.

The set of splicing rules of R is divided in two groups, according to their use. The first group consists rules that rotate the marked string.
$r_1: \ \{d\#o_r\$X_1\#X_a o_r \mid d \in \{a,b,c\}\}$,
$r_2: \ \{\#dX_e o_r\$X_2\#X_d X_e o_r \ \mid e,d \in \{a,b,c\}, e \neq d\}$
$r_3: \ \{o_l X'_e\#X_3\$o_l\#d, \mid e,d \in \{a,b,c\}\}$
$r_4: \ \{\#X_d X_e o_r\$X_4\#X_e o_r \ \mid e,d \in \{a,b,c\}, e \neq d\}$
$r_5: \ \{o_l e\#X_5\$o_l X'_e\# \mid e \in \{a,b,c\}\}$.

The second group of splicing rules is used to remove one of the symbols a, b, or c from the marked string.
$r_6: \ \#aX_a o_r\$X_6\#X_b o_r$,
$r_7: \ \#bX_b o_r\$X_6\#X_c o_r$,
$r_8: \ \#cX_c o_r\$X_6\#X_a o_r$.

The initial language of the SR is $L = \{X_1X_e o_r, o_l X'_e X_3, X_4 X_e o_r, o_l e X_5 \mid e \in \{a,b,c\}\} \cup \{X_2 X_d X_e o_r \mid d,e \in \{a,b,c\}, e \neq d\} \cup \{X_6 X_b o_r, X_6 X_c o_r, X_6 X_a o_r\}$. Notice the language is finite. The target marked language is $L_t = \{o_l X_a o_r\}$.

The observer O has input alphabet V and output alphabet $\Sigma = \{l_0, \perp\} \cup \{l_{e,1}, l_{e,2}, l_{e,3}, l_{e,4} \mid e \in \{a,b,c\}\}$.

The mapping implemented by the observer is

$$O(w) = \begin{cases} l_0 & \text{if } w \in o_l\{a,b,c\}^+ o_r, \\ l_{e,1} & \text{if } w \in o_l\{a,b,c\}^+ X_e o_r, \ e \in \{a,b,c\} \\ l_{e,2} & \text{if } w \in o_l\{a,b,c\}^* X_d X_e o_r, \ e,d \in \{a,b,c\} \\ l_{e,3} & \text{if } w \in o_l X'_d\{a,b,c\}^* X_d X_e o_r, \ e,d \in \{a,b,c\} \\ l_{e,4} & \text{if } w \in o_l X'_d\{a,b,c\}^* X_e o_r, \ e,d \in \{a,b,c\} \\ \lambda & \text{if } w \in \{o_l X_a o_r\} \\ \perp & \text{else.} \end{cases}$$

The decider D is a finite state automaton, with input alphabet Σ, that gives a positive answer exactly if and only if, a word belongs to the regular language
$l_0(l_{a,1}(l_{a,2}l_{a,3}l_{a,4}l_{a,1})^* l_{b,1}(l_{b,2}l_{b,3}l_{b,4}l_{b,1})^* l_{c,1}(l_{c,2}l_{c,3}l_{c,4}l_{c,1})^*)^+$.

At the beginning of the computation the input marked string is of the kind $o_l\{a,b,c\}^+ o_r$ and it is mapped by the observer to l_0. If the input marked string is not of this type, then the observer outputs something different than l_0, and the entire evolution is not accepted by the decider D. In the first step, the splicing rule $d\#o_r\$X_1\#X_a o_r$ from r_1 is used, and in this way a new marked string of the type $o_l\{a,b,c\}^+ X_a o_r$ is obtained and mapped by the observer to $l_{a,1}$. The introduced symbol X_a indicates that we want to search (and then to remove) a symbol a from the obtained marked string. This searching is done by rotating the marked string, until a symbol a becomes the symbol immediate to the left of X_a. The rotation of the string is done by using the splicing rules given in the first group.

A rotation of the string consists in moving the symbol immediately to the left of X_a, to the right of o_l; one rotation is done by applying, in a consecutive way, a rule from r_2, from r_3, from r_4 and finally from r_5 (the precise rules to apply depend on the symbol to move during the rotation). The sequence of marked strings obtained during a rotation is mapped by the observer to the string $l_{a,2}l_{a,3}l_{a,4}l_{a,1}$. The $*$ present in the regular expression describing the decider language, indicates the possibility to have 0, or more consecutive rotations before a symbol a comes to be the symbol immediately to the left of X_a.

The observer checks that each rotation is made in a correct way; that is, the symbol removed from the left of X_a by using a rule from r_4, is exactly the same symbol introduced to the right of o_l, by using the corresponding rule in r_3. This condition is checked in the fourth line of the observer mapping; if this regular condition is not respected, then the observer outputs $\perp$ and the entire evolution of the input marked string is not accepted by the decider D.

Once a symbol a becomes the symbol immediately to the left of X_a, rotation stops. It is deleted by using the splicing rule r_6. When rule r_6 is applied, the new marked string obtained belongs to $o_l\{a,b,c\}^+X_bo_r$ and is mapped by the observer to $l_{b,1}$. The inserted symbol X_b, indicates that now we search the symbol b.

In a similar way, by using consecutive rotations, a symbol b is placed immediately to the left of X_b and is being removed by rule r_7. In this case, the required sequence of marked strings obtained during each rotation is mapped by the observer to $l_{b,2}l_{b,3}l_{b,4}l_{b,1}$. Once rule r_7 is applied, the new marked string obtained belongs to $o_l\{a,b,c\}^+X_co_r$ and is mapped by the observer to $l_{c,1}$.

Analogously, the symbol c is searched for and then deleted by using rule r_8. In this case, the required sequence of marked strings obtained during each rotation is mapped by the observer to the string $l_{c,2}l_{c,3}l_{c,4}l_{c,1}$. At this point the entire process can be iterated. By searching and removing a new symbol a, and then again a b, and again a c, until the marked string $o_lX_ao_r$, from the target language is reached. This string is obtained when all symbols a, b and c, have been deleted from the input marked string. Note that at each step the current marked string is spliced with a string from the initial language.

With this argument we show that all strings from the language $\{\, w \in o_l\{a,b,c\}^+o_r : |w|_a = |w|_b = |w|_c\}$ can indeed be accepted by Ω. The fact that only such strings can be accepted is controlled by the particular form of sequences accepted by the decider in combination with the very specific form of the observed strings leading to such a sequence.

□

5 Computational Completeness

In this section we prove that SRs with radius ≤ 4 have computational power equivalent to a general Turing machine. Informally, *it is possible to simulate an accepting Turing machine by observing, with a simple observer, the evolution of a simple splicing system.*

The universality is not unexpected since, H systems with observer and decider are similar to splicing systems with regular target languages, known to be universal [15].

The proof follows the idea used in Theorem 4.1.

Theorem 5.1 *For each RE language L over the alphabet A there exists an SR Ω of radius ≤ 4 such that Ω accepts the language $\{o'_lwo'_r \mid w \in L\}$, with $o'_l, o'_r \notin A$.*

Proof. Utilizing Church-Turing thesis we only show that, for any Turing machine, an equivalent SR system Ω can be constructed. In this proof we use off-line Turing machines with only a single combined input/working tape. The set δ of Turing machine transitions is a function form $Q \times A \rightarrow Q \times A \times \{+,-\}$, where Q is the set of states, A the tape alphabet, and $+$ or $-$ denotes a move to the right or left, respectively. An input word is accepted, if and only if, the Turing machine stops in a state that belongs to $F \subset Q$ of final states. Without loss of generality, we suppose that the machine M accepts the input, if and only if it reaches a configuration where the tape is entirely empty, and M is in a state that belongs to F. The initial state of M is $q_0 \in Q$. The special letter $\Box \in A$ denotes an empty tape cell.

We construct an SR Ω simulating M. Before giving the formal details, we outline the basic idea of the proof. The input string to the Turing machine is inserted as input marked string to the SR Ω, delimited by two external markers o'_l, o'_r. This does not restrict the generality of the theorem, because these two symbols could be added to any input string in two beginning steps by the SR. However, we want to spare ourselves the technical details of this.

Initially, an arbitrary number of empty tape cells $\Box$ is added to the left and to the right of the input marked string. When this phase is terminated, some new markers o_l and o_r are added to the left and right of the produced marked string; starting from this step, the transitions of the Turing machine M are simulated on the current marked string; the marked string contains, at any time, the content of the tape of M, the current state and the position of the head of M over the tape. To read the entire tape of M the marked string is rotated using a procedure very similar to the one described in the proof of Theorem 4.1; like there, the observer can check that the rotations are done in a correct way. The computation of Ω stops when the target marked string is reached, that is when a marked string representing an empty tape is obtained.

Formally, the SR $\Omega = (\sigma, O, D, L, L_t)$ is constructed in the following way.

The H scheme $\sigma = (V, R)$ has alphabet $V = \{o_r, o_l, o'_r, o'_l, X_1, X_2, \cdots, X_{12}\} \cup A' \cup \{X_e, X'_e \mid e \in A'\}$ where $A' = A \cup (A \times Q)$.

The splicing rules present in R are divided in groups, according to their use.

Initialization

$r_1: \{o'_l(a, q_0)\#X_1\$o'_la\# \ \mid a \in (A - \{\Box\})\}$;

$r_2: \{\#o'_r\$X_2\#\Box o'_r\}$;

$r_3: \{o'_l\Box\#X_3\$o'_l\#\}$;

$r_4: \{\#o'_r\$X_4\#o_r\}$;

$r_5: \{o_l\#X_5\$o'_l\#\}$;

Rotations

$r_6: \{a\#eo_r\$X_6\#X_eo_r \ \mid e \in A', a \in A\}$;

$r_7: \{o_lX'_e\#X_7\$o_l\#f \ \mid e, f \in A'\}$;

$r_8: \{a\#X_eo_r\$X_8\#o_r \ \mid e \in A', a \in A\}$;

$r_9: \{o_le\#X_9\$o_lX'_e\#f \ \mid e, f \in A'\}$;

Transitions

$r_{10}: \{\#(a, q_1)bo_r\$X_{10}\#c(b, q_2)o_r \ \mid q_1, q_2 \in Q, a, b, c \in A, \ (q_1, a) \rightarrow (q_2, c, +) \in \delta \ \}$;

$r_{11}: \{\#b(a, q_1)do_r\$X_{11}\#(b, q_2)cdo_r \ \mid q_1, q_2 \in Q, a, b, c, d \in A, \ (q_1, a) \rightarrow (q_2, c, -) \in \delta\}$;

Halting phase

r_{12} : $\{o_l\#\$X_{12}\#o_r\}$.

The initial language L is the finite language containing the strings used by the mentioned splicing rules; in particular, $L = \{o'_l(a, q_0)X_1 \mid a \in (A - \{\Box\})\}$ $\cup\{X_2o'_r, o'_l\Box X_3, X_4o_r, o_lX_5, X_8o_r, X_{12}o_r\}\cup\{X_6X_eo_r, o_lX'_eX_7, o_leX_9 \mid e \in A'\}\cup\{X_{10}c(b, q_2)o_r \mid q_2 \in Q, c, b \in A\} \cup \{X_{11}(b, q_2)cdo_r \mid b, c, d \in A, q_2 \in Q\}$.

The target marked language is $L_t = \{o_lo_r\}$. The observer has input alphabet V and output alphabet $\Sigma = \{l_0, l_1, \cdots, l_8, l_f, \perp\}$.

The mapping implemented by the observer is

$$O(w) = \begin{cases} l_0 & \text{if } w \in o'_l(A - \{\Box\})^+o'_r, \\ l_1 & \text{if } w \in o'_l(a, q_0)(A - \{\Box\})^*o'_r,\ a \in (A - \{\Box\}), \\ l_2 & \text{if } w \in o'_l(A' - \{\Box\})^+(\Box)^+o'_r, \\ l_3 & \text{if } w \in o'_l(\Box)^+(A' - \{\Box\})^+(\Box)^+o'_r, \\ l_4 & \text{if } w \in \{o'_lw'o_r \mid w' \in (\Box)^*(A' - \{\Box\})^+(\Box)^*, length(w') \geq 3\}, \\ l_5 & \text{if } w \in (o_l(A')^+o_r - \{w \mid w \in E\}), \\ l_6 & \text{if } w \in o_l(A')^*X_eo_r,\ e \in A', \\ l_7 & \text{if } w \in o_lX'_e(A')^*X_eo_r,\ e \in A', \\ l_8 & \text{if } w \in o_lX'_e(A')^*o_r,\ e \in A', \\ l_f & \text{if } w \in E, \\ \perp & \text{else.} \end{cases}$$

where $E = o_l(\Box)^*(\Box, q)(\Box)^+o_r \cup o_l(\Box)^+(\Box, q)(\Box)^*o_r \cup o_l(\Box)^+(\Box, q)(\Box)^+o_r, q \in Q$.

The decider is a finite state automaton, with input alphabet Σ that accepts the regular language $E_1 \cup E_2$, where $E_1 = l_0l_1(l_2)^+(l_3)^*l_4(l_5 \cup l_5l_5)(l_6l_7l_8(l_5 \cup l_5l_5))^*l_f$ and $E_2 = l_0l_1l_4(l_5 \cup l_5l_5)(l_6l_7l_8(l_5 \cup l_5l_5))^*l_f$.

The main point of the proof is to show that, given an input marked string w, at least one of its (observed) evolutions is of the type accepted by the decider if and only if the string w is accepted by the Turing machine M.

We now describe the (observed) evolution of a correct input marked string; from this, we believe it will be clear that non correct strings will not have an evolution of the kind accepted by the decider, and, therefore will not be accepted by the SR Ω. The reader can compare the observed evolution of the input marked string with the language accepted by the decider.

Actually we introduce in the system Ω not the string w but a string of the type $o'_lwo'_r$ where o'_l, o'_r are left and right delimiters. In general the input marked string will be of the type $o'_l(A - \{\Box\})^+o'_r$ and is mapped by the observer to l_0. The pairs in $Q \times A$ are used to indicate in the string the state and the position of the head of M. Initially the head is positioned on the leftmost symbol of the input marked string, starting in state q_0 (by using a rule in r_1); the obtained marked string is of the kind $o'_l(a, q_0)(A-\{\Box\})^*o'_r,\ a \in (A-\{\Box\})$ mapped to l_1 by the observer.

Then empty cells $\Box$ are added to the right and to the left of the marked string using rules in r_2 and in r_3, respectively. The marked string obtained at the end of this phase will be of the kind $o'_l(\Box)^+(A' - \{\Box\})^+(\Box)^+o'_r$ mapped to l_3 by the observer. This phase is optional, and therefore the language of the decider is described by the union of E_1 where the adding of spaces is used and E_2, where no spaces are added, i.e., l_2 and l_3 are missing.

Then, by using rules in r_4 and in r_5 the delimiters o'_l and o'_r are changed into o_l and o_r, respectively. When a rule in r_4 is applied, the marked string obtained is of the kind $o'_lw'o_r, w' \in (\Box)^*(A' - \{\Box\})^+(\Box)^*$ mapped to l_4 if the size of the string w' (possibly, including empty cells) is at least of 3 symbols; this condition is useful during the following phases of rotations and does not imply a loss of generality.

When a rule in r_5 is applied, also o'_l is removed and the marked string obtained is mapped to l_5 by the observer. This means that the symbol indicating the head of M, (a, q_1), is exactly one symbol away from o_r, then a splicing rule in r_{10} or in r_{11} is applied. The one symbol left between the symbol representing the head and the delimiter o_r is useful in case of the simulation of a right-moving transition. The rule sets r_{10} and r_{11} correspond to transitions moving right and left, respectively.

Once a transition is simulated, the obtained marked string is again of the type mapped to l_5 by the observer (this is why it is possible to have in the language of the decider the substring $l_5 l_5$). At any rate it is not possible to have immediately another transition after a transition, because the symbol corresponding to the head of M is moved. At least one rotation must be first executed.

In case the symbol representing the head of M is not exactly one symbol away from o_r, then the marked string is rotated until this condition is not true any more. The rotation of one symbol in the string (i.e., moving the symbol present to the left of o_r, to the immediate right of o_l) is done by applying, in this order, splicing rules from r_6, r_7, r_8 and from r_9. The marked strings obtained during this phase are mapped by the observer to l_6, l_7, l_8 and finally l_5. At the end of a rotation a transition can be simulated; more consecutive rotations can be done until the necessary condition to simulate a transition is reached. This explains why $(l_6 l_7 l_8 (l_5 \cup l_5 l_5))^*$ forms part of the decider language.

When, after a transition, the marked string obtained represents the empty tape of M, then the computation of the SR stops. The marked strings representing an empty tape are the ones in the language E and they are mapped by the observer to l_f. After the observer has output l_f, the splicing rule in r_{12} can be applied and the unique string in the target marked language $o_l o_r$ can be reached. If the rule in r_{12} is applied before the observer outputs l_f, then the entire evolution is not accepted by the decider. Notice that during the entire computation the marked string can be spliced only with a string from the initial language.

From the above explanation, it follows that an input marked string written in the form $o'_l w o'_r$ is accepted by Ω, if and only if, w is accepted by the Turing machine M. □

6 Perspectives and Concluding Remarks

Although the components within a splicing recognizer are elementary (finite H scheme and finite state automata), the computational power of the model is not surprising. The biomolecular evolution of the marked molecule through splicing is potentially rather complex. However, our model first filters out the unwanted behavior (requirement of the target language) then maps the chosen behavior into a string (observer) and finally filters out one more time those strings that are not acceptable (decider). It is well known that the class of finite splicing systems cannot generate languages with complexity higher than regular. The computational power of the splicing recognizer increases dramatically by following the whole process of splicing through additional controls provided with the observer and the decider,.

The proposed approach suggests several interesting problems.

For instance, the process of observation as defined here is non-deterministic; meaning, the initial marked DNA strand is accepted if at least one of its observed evolution follows an expected pattern, while there might be several possible evolutions of this DNA strand since there might be several different ways to splice the strand. It would be interesting to see if by increasing the complexity of the observer and/or the decider, a ("more") deterministic way of generating the splicing evolution can be employed.

We recall that in the model presented here the observers and deciders are simple devices, i.e., finite state automata. Moreover, in the model presented here it is supposed that the observer is able to catch, in the molecular soup, every single change of the marked DNA strand. In practice, it is very questionable whether every step of the evolution can be observed. It should be assumed that only some particular types of changes, within a certain time-interval can be observed (see [12]). Therefore another variant of SR needs to be, at least theoretically, investigated in which an observer with "realistic" limitations on the ability of observation is considered. For instance the observer might be able to watch only a window or a scattered subword of the entire evolution.

There are a variety of technical problems that can be considered. For example, the universal computational power has been obtained by using an SR of radius ≤ 4. We conjecture that it is possible to decrease the radius, hence the question arises: what is the minimum radius that provides universal computation. Moreover what is the power of SRs of radius 1: is it enough to obtain a class larger than regular? (i.e., can the result of Theorem 3.1 be improved?)

It remains also to investigate SRs using simpler and more restricted variants of H schemes, like the ones with simple splicing [13], and semi-simple splicing rules [9]. Note that from a pure theoretical point of view, observer and decider could be joined in a unique finite state automaton, which may provide a better framework for theoretical investigation. In this paper we prefer to leave the two "devices" separated since this situation can be envisioned to be closer to reality.

Moreover, we can interpret a given H scheme with an observer as a device computing a function, by considering as input the input marked string, and as output its (observed) evolution. What kind of functions can be computed in this way?

On other hand, in this paper we have supposed that, to get a specified Turing machine, one has to construct a specified splicing system (bio-system) and a specified observer/decider of such bio-system. Actually, in nature the majority of biological systems cannot be easily "programmed" and changed; these bio-systems have their own rules, fixed by nature, and they just live (evolve) according to such rules (this is true also for splicing systems; some of them are easier to implement than others). On the other hand, it is possible to observe the evolution of a bio-system by using different observers. Therefore it is natural to ask what are the limitations of the observe while keeping the underlying bio-system fixed. For instance, in the presented framework, is it possible to simulate every Turing machine, by keeping the underlying splicing system fixed and changing only the observer and the decider?

The answer to this problem may be actually true, considering that in [3] it has been proven that every (generative) Turing machine can be simulated by observing a fixed context-free grammar, by only choosing the "right" observer. If this is confirmed in our framework, this may mean that any computing device can be constructed by finding the right observer and the right decider for a fixed underlying splicing system.

These are only a few of the possible directions of investigation that arise from the presented approach. We believe that some of these directions may provide useful conditions for using recombinant DNA for computing.

Acknowledgments

The authors want to thank Peter R. Cook for providing extremely useful references.
P. Leupold has been supported by the FPU grant of the Spanish Ministry of Science and Education. N. Jonoska has been supported in part by NSF Grants CCF #0432009 and CCF#0523928. The paper is based on the work presented at the 11th International Meeting on DNA Based Computers held in London, Canada, June 2005.

References

[1] A. Alhazov, M. Cavaliere, Computing by Observing Bio-Systems: the Case of Sticker Systems. *Proceedings of DNA 10 - Tenth International Meeting on DNA Computing*, Lecture Notes in Computer Science, 3384 (C. Ferretti, G. Mauri, C. Zandron eds.), Springer, 2005, pp. 1–13.

[2] M. Bruchez, M. Moronne, P. Gin, S. Weiss, A.P. Alavisatos, Semiconductor Nanocrystals as Fluorescent Biological Labels. *Science*, 281, 1998, pp. 2013-2016.

[3] M. Cavaliere, P. Frisco, H.J. Hoogeboom, Computing by Only Observing. *Proceedings Tenth International Conference on Developments in Language Theory, DLT06*, Lecture Notes in Computer Science, 4036, Springer, 2006, pp. 304–314.

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

[5] M. Cavaliere, P. Leupold, Observation of String-Rewriting Systems. *Fundamenta Informaticae* 74(4), 2006, pp. 447–462.

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

[7] M. Cavaliere, P. Leupold, Evolution and Observation – A Non-Standard Way to Generate Formal Languages. *Theoretical Computer Science*, 321, 2004, pp. 233-248.

[8] W.C.W. Chan, S. Nie, Quantum Dot Bioconjugates for Ultrasensitive Nonisotopic Detection. *Science*, 281, 1998, pp. 2016-2018.

[9] E. Goode, D. Pixton, Semi-Simple Splicing Systems. *Where Mathematics, Computer Science, Linguistics and Biology Meet* (C. Martin-Víde, V. Mitrana eds.), Kluwer Academic Publisher, 2001, pp. 343 – 352.

[10] T. Head, Formal Language Theory and DNA: An Analysis of the Generative Capacity of Specific Recombinant Behaviors. *Bulletin of Mathematical Biology* 49, 1987, pp. 737-759.

[11] J.M. Levsky, S.M. Shenoy, R.C. Pezo, R.H. Singer, Single-Cell Gene Expression Profiling. *Science*, 297, 2002, pp. 836–40.

[12] J. Lippincott-Schwartz, E. Snapp, A. Kenworthy, Studying Protein Dynamics in Living Cells. *Nature Rev. Mol. Cell. Biol.*, 2, 2001, pp. 444–456.

[13] A. Mateescu, Gh. Păun, G. Rozenberg, A. Salomaa, Simple Splicing Systems. *Discrete Applied Mathematics*, 84, 1998, pp. 145–163.

[14] X. Michalet, F.F. Pinaud, L.A. Bentolila, J.M. Tsay, S. Doose, J.J. Li, G. Sundaresan, A.M. Wu, S.S. Gambhir, S. Weiss, Quantum Dots for Live Cells, in Vivo Imaging and Diagnostic. *Science*, 307, 2005, *www.sciencemag.org.*

[15] Gh. Păun, Splicing Systems with Targets are Computationally Universal. *Information Processing Letters*, 59, 1996, pp. 129-133.

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

[17] R. Rigler, E.S. Elson, *Fluorescent Correlation Spectroscopy.* Springer, New-York, 2001.

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

[19] G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages.* Springer-Verlag, Berlin, 1997.

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