Decision problems for linear and circular splicing systems *

Paola Bonizzoni[1], Clelia De Felice[2], Giancarlo Mauri[1], and Rosalba Zizza[2]

[1] Dipartimento di Informatica Sistemistica e Comunicazione
Università degli Studi di Milano - Bicocca
Via Bicocca degli Arcimboldi 8, 20126 Milano - Italy
{bonizzoni, mauri}@disco.unimib.it
[2] Dipartimento di Informatica ed Applicazioni,
Università di Salerno, 84081 Baronissi (SA), Italy
{defelice,zizza}@unisa.it

Abstract. We will consider some generative devices inspired by cut and paste phenomena on DNA molecules under the action of restriction and ligase enzymes, *the splicing systems*, introduced by Tom Head in 1987. We will then survey the most important results obtained in this area, and evidence how the classical techniques and definitions in automata theory are a legitimate tool for facing a few unsolved problems.

1 Introduction

We will consider here the *splicing systems*, generative devices inspired by cut and paste phenomena on DNA molecules under the action of restriction and ligase enzymes. A DNA strand can be viewed as a string over a four letter alphabet (the four deoxyribonucleotides Adenine, Guanine, Cytosine and Thymine). Therefore the natural way to model DNA computation is within the framework of formal language theory. In spite of a vast literature on splicing systems, briefly surveyed here, a few problems related to their computational power are still open. We intend to evidence how classical techniques and concepts in automata theory are a legitimate tool for investigating some of these problems.

For example, we quote the concepts of syntactic congruence and of *syntactic monoid* [2, 28, 41], introduced and developed by M.-P. Schützenberger's school in the framework of the algebraic theory of *variable-length codes*, that have been helpful in investigating the class of regular languages, i.e., languages recognized by finite state automata.

The notion of *splicing system* was introduced by Tom Head in 1987 [21]. Different variants of the original definition have been proposed briefly; some of these by Paun and Pixton [36, 39, 40, 44]. A splicing system (or H-system) is a triple $S = (A, I, R)$, where A is a finite alphabet, $I \subseteq A^*$ is the initial language

* Partially supported by MIUR Project *"Linguaggi Formali e Automi: teoria ed applicazioni"* and by the contribution of EU Commission under The Fifth Framework Programme, project MolCoNet IST-2001-32008.

and $R \subseteq (A')^*$, $A \subseteq A'$, is the set of rules for the splicing operation (see Section 3 for the definitions). The formal language generated by the splicing system is the smallest language containing I and closed under the splicing operation. The computational power of a splicing system depends on which level in the Chomsky hierarchy the initial set I and the set of the rules R belong to. This computational power has been deeply investigated [26, 40]. According to some hypotheses on I, R, splicing systems can reach the same power of the Turing machines [26, 36]. At the opposite end of the hierarchy, when we restrict ourselves to splicing systems with a finite set R of rules and a finite set I of strings, we get a proper subclass of regular languages, as shown in [12, 16, 32, 44]. The behaviour of a splicing system under the hypothesis that I belongs to a full abstract family of languages (full AFL) is studied in [42]. Efficient simulations between some of these models and finite state automata are designed in [32, 44]. Other models, such as Uniterated splicing systems, Extended splicing systems, Extended splicing systems with multiplicity, Extended splicing systems with permitting and forbidding contests and Communicating distributed H-systems are described in [15, 26, 30, 37, 38]. Some variants of these models have proved to be computationally equivalent to Turing machines.

The idea of using biological processes as models to compute is the basis of Adleman's experiment, performed some years after the introduction of splicing systems [1]. Adleman showed how to solve the (NP-complete) Hamiltonian Path Problem by manipulating DNA sequences in a lab. This research represents part of the general trend towards the proposal of new (biological and quantistic) models of computation. These new models do not affect the validity of the Church-Turing thesis (the intuitive notion of the effective procedure is the same as the mathematical concept of the Turing machine). Behind these new models there is rather the attempt to approximate the unbounded parallelism of non deterministic Turing machines, with a reduction of the time and space required for the solutions of the so-called intractable problems, pointed out in the framework of the computational complexity theory.

Nowadays, developments in both directions have been achieved. On one side, other systems such as Watson-Crick Automata, Insertion and Deletion Systems, Sticker systems, P-systems have been presented [40]. On the other, Adleman's experiment has implemented the performing of new experiments in the lab, thus solving other NP-complete problems [24, 27, 29, 40].

In detail, and as we have already stated, a DNA strand can be considered as a word on a finite alphabet, therefore, splicing operations on DNA strands can be regarded as operations on strings [21, 26, 30]. As DNA occurs in both linear and circular form, two kinds of splicing operations (linear and circular splicing) were defined.

In nature, linear splicing occurs in two steps. First, restriction enzymes (proteins) recognize a specific pattern in a DNA sequence and cut the molecule in a specific point inside the pattern. The site and the shape of the cut ends are specific for each enzyme. Then ligase enzymes paste together properly matched fragments, under specific chemical conditions. This operation is performed on

two DNA molecules. Therefore, two new sequences are produced from the two DNA sequences. This phenomenon can be easily translated into a formal language framework. There are at least three definitions of linear splicing systems, given by Head, Paun and Pixton respectively and the computational power of these systems is different in the three cases.

Considering that linear splicing has already been investigated thoroughly, we will now focus on its relationship with regular languages. It is known that this class of languages coincides with the class of the languages generated by splicing systems with a regular initial set of words and a finite set of rules, whereas finite linear splicing systems (splicing systems with finite set of rules and finite initial language) generate a proper subclass of regular languages [44]. One of Head's results shows that finite linear splicing systems with a special type of rules always generate strictly locally testable languages, a well-known subclass of regular languages [13, 23]. The search for a characterization of the subclass of regular languages generated by finite linear splicing systems is one of the open problems proposed by Head.

One of the aims of this paper is to show that classical notions in automata theory (syntactic monoid, constant) can be useful in solving this problem. We recall that the *syntactic monoid* associated with a language L is the quotient monoid $\mathcal{M}(L)$ w.r.t. the *syntactic congruence* $\equiv_L$. In [7, 8] the authors give a construction of finite linear (Paun) splicing systems, each of which generates a regular language defined by a condition that uses the notion of syntactic monoid (see Section 3.3). Recently, Goode and Pixton proved that it is possible to decide whether a regular language is generated by a finite Paun linear splicing system [17, 25].

We will now consider circular splicing systems limited to the aspect of their relationship with regular languages. Obviously a string of circular DNA will be represented as a circular word, i.e., as the equivalence class with respect to the conjugacy relation $\sim$, defined by $xy \sim yx$, for $x, y \in A^*$ [33]. A circular language is a set of circular words and we can also give a definition of regular circular language. We will notice that when we look at regular circular languages it is the same as when we look at regular languages closed under the conjugacy relation. The structure of the latter languages is still unknown except for one special case [3, 45]. The search for a characterization of the structure of regular languages closed under the conjugacy relation is one of the interesting problems in this framework (see Section 5).

There are three definitions of circular splicing respectively given by Head, Paun and Pixton. Biological circular splicing occurs in a recombinant mechanism (transposition) between bacteria and plasmids. In this case, a restriction enzyme recognizes a pattern inside a circular DNA molecule and then opens the molecule inside the pattern. This operation is performed on two circular DNA molecules. Ligase enzymes paste them together, producing a new circular DNA sequence (see [21, 26, 40] for further details). Depending on whether or not these ligase enzymes substitute the recognized pattern, we have Pixton's definition or Head's and Paun's definition.

A circular splicing system is once again a triple (A, I, R) where A is a finite alphabet, I is the initial circular language and R is the set of rules. The circular splicing language generated by a circular splicing system is defined as in the linear case. Of course, the computational power of these models has been investigated, but few results are known. One important result, obtained by Pixton, states that with a regular initial set and finite set of rules, under additional hypotheses (reflexivity, symmetry) on the rules and adding self-splicing, the language generated is regular [44]. Even in the finite case (I, R finite sets), the computational power of circular splicing systems is unknown. We do not know which class of languages they generate and we do not even know which regular languages belong to this class (indeed both regular and non regular languages may be generated). We know that they cannot generate all circular regular languages. Indeed, $^\sim((A^2)^* \cup (A^3)^*)$ cannot be generated by finite circular splicing systems, even if the more powerful Pixton definition of circular splicing is applied [7].

We will now present partial results to this problem. In particular, we will present a family of circular regular languages generated by finite circular splicing systems, namely the so-called star languages $^\sim X^*$ [5]. A main example of languages belonging to this class is provided by free monoids X^* generated by regular group codes X (see Section 5.1 for further details).

We will indicate the complete characterization of the languages over a one-letter alphabet generated by finite (Paun) circular splicing systems. In particular these languages are exactly the regular languages $L = L_1 \cup \{a^g \mid g \in G\}^+$, where L_1 is a finite subset of a^* and all the words in L_1 are shorter than the words in $\{a^g \mid g \in G\}^+$. There is also a positive integer n and a subgroup G' of Z_n so that G is a set of representatives of the elements in G'. We have also given a characterization of these languages in terms of automata, proving that we can decide whether a regular language over a one-letter alphabet can be generated by finite (Paun) circular splicing systems [6, 7].

In this paper basics on words and languages are gathered in Section 2; Section 3 is devoted to linear splicing and Section 3.2 presents the general framework and a short description of a few results obtained on the relationship between splicing systems and regular languages. In Section 3.3 we have limited our study to the case of regular languages generated by finite linear splicing systems. The last two sections concern circular splicing. Definitions are presented in Section 4 and the results on the relationship between finite circular splicing systems and circular regular languages are represented in Section 5.

2 Basics

2.1 Words

Let A^* be the free monoid over a finite alphabet A and let $A^+ = A^* \setminus 1$, where 1 is the empty word. For a word (or string) $w \in A^*$, $|w|$ is the length of w and, for a subset X of A^*, we denote $|X|$ the cardinality of X.

In the following $\mathcal{A} = (Q, A, \delta, q_0, F)$ will be a finite state automaton, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and δ is the transition function [28, 41]. A finite state automaton $\mathcal{A}$ is *deterministic* if, for each $q \in Q$, $a \in A$, there is at most one state $q' \in Q$ so that $\delta(q, a) = q'$. Furthermore, $\mathcal{A}$ is *trim* if each state is accessible and coaccessible, i.e., if for each state $q \in Q$ there are $x, y \in A^*$ such that $\delta(q_0, x) = q$ and $\delta(q, y) \in F$.

Given a regular language $L \subseteq A^*$, it is well known that there is a *minimal* finite state automaton $\mathcal{A}$ recognizing it, i.e. $L = L(\mathcal{A})$. This automaton is unique up to a possible renaming of the states, is trim and has the minimal number of states. It can be obtained thanks to standard construction algorithms and is related to the syntactic monoid of L [28].

The *syntactic monoid* associated with L is the quotient monoid $\mathcal{M}(L)$ with respect to the *syntactic congruence* $\equiv_L$ [41]. Let us recall that two words w, w' are equivalent with respect to the *syntactic congruence* if they have the same set of *contexts* $C(w) = \{(x, y) \in A^* \times A^* \mid xwy \in L\}$, i.e.,

$$w \equiv_L w' \;\Leftrightarrow\; [\forall x, y \in A^*, xwy \in L \;\Leftrightarrow\; xw'y \in L] \;\Leftrightarrow\; C(w) = C(w')$$

In the following, $[w]$ will be the congruence class of w modulo $\equiv_L$. If L is a regular language then the index (i.e., the number of congruence classes) of the syntactic congruence is finite and therefore, $\mathcal{M}(L)$ is a finite monoid. A useful characterization of the syntactic congruence states that, given a regular language L recognized by the minimal automaton $\mathcal{A} = (Q, A, \delta, q_0, F)$, $w \equiv_L w'$ if and only if for each $q \in Q$ we have $\delta(q, w) = \delta(q, w')$ [34].

Let us recall the definition of a constant. Let $\mathcal{A}$ be the minimal finite state automaton recognizing a regular language L and let $w \in A^*$. We will set $Q_w(\mathcal{A}) = \{q \in Q \mid \delta(q, w) \;\; \textit{is defined}\}$, simply indicated Q_w when the context makes the meaning evident. Observe that the notation can be extended to $Q_{[x]}$ with $x \in A^*$. The *left* and *right* contexts of a word $w \in A^*$ are therefore defined as follows. Notice that these definitions are slightly different from the ones given in [41].

$$C_L(w) = \{z \in A^* \mid \exists q \in Q_w : \delta(q_0, z) = q\}$$
$$C_{R,q}(w) = \{y \in A^* \mid \delta(q, wy) \in F\},\; C_R(w) = \bigcup_{q \in Q_w} C_{R,q}(w)$$

A word $w \in A^*$ is a *constant* for a regular language L if $C(w) = C_L(w) \times C_R(w)$ [46]. Obviously, if w is a constant for L and $w' \in A^*$ with $w \equiv_L w'$, then w' is also a constant for L.

Finally, we will name $FIN, REG, LIN, CF, CS, RE$ the class of finite, regular, linear, context-free, context sensitive, recursive enumerable languages, respectively.

2.2 Circular words

Circular words have already been exhaustively examined in formal language theory (see [2, 11, 33]). For a given word $w \in A^*$, the circular word $^\sim w$ is the equivalence class of w with respect to the *conjugacy* relation $\sim$ defined by $xy \sim yx$, for $x, y \in A^*$. The notation $|^\sim w|$ will be defined as $|w|$, for any representative w of $^\sim w$. When the context does not make it ambiguous, we will use the notation w for a circular word $^\sim w$. Furthermore, $^\sim A^*$ is the set of all circular words over A, i.e., the quotient of A^* with respect to $\sim$. If $L \subseteq A^*$, $^\sim L = \{^\sim w \mid w \in L\}$ is the *circularization* of L, i.e., the set of all circular words corresponding to the elements of L, while every language L such as $^\sim L = C$, for a given circular language $C \subseteq ^\sim A^*$, is called a *linearization* of C. The set $Lin(^\sim L) = \{w' \in A^* \mid \exists w \in L.\ w' \sim w\}$ of all the strings in A^* corresponding to the elements of $^\sim L$ is the *full linearization* of $^\sim L$.

If FA is a family of languages in the Chomsky hierarchy, $FA^\sim$ is the set of all those circular languages C which have some linearization in FA. In this paper we only deal with $REG^\sim$. It is known that given a regular language $L \subseteq A^*$, its circularization $^\sim L$ is a regular circular language if and only if its full linearization $Lin(^\sim L)$ is a regular language [28].

Consider that circular splicing deals with circular strings and circular languages. Therefore, it deals with formal languages which are closed under the conjugacy relation and with the action of circular splicing over their circularization. The result is that handling of a regular circular language is the same as handling of a regular language closed under the conjugacy relation (see Section 5).

2.3 Cycles

In this section we will determine a few notations and give definitions for handling of the labels c (cycles) of some special closed paths in the transition diagram of a finite state automaton $\mathcal{A}$ [6, 7]. Indeed, if a rational infinite language L is generated by splicing, we must exhibit a finite number of rules which are able to generate words with an unbounded number of occurrences of cycles as factors. The following observation arises in [7]. Let L be a regular language recognized by the minimal finite state automaton $\mathcal{A} = (Q, A, \delta, q_0, F)$. Let $c \in A^+$ be the label of a closed path in the transition diagram of $\mathcal{A}$, i.e., there exists $q \in Q$ such that $\delta(q, c) = q$. Consider the corresponding internal states crossed by the transition $\delta(q, c) = q$. These internal states can either be different from q or not different from q. Furthermore, in the first case, these internal states can be either different from each other or not different from each other. This idea is formalized in an accurate statement in [7]. Furthermore, we can visualize these three cases when we take into account Figure 1: the closed path labelled a is an example of the first case; bab is the label of a closed path which satisfies the second condition, whereas ad is the label of a closed path which satisfies the third condition.

Definition 1 (Cycle). *[6] A word $c \in A^+$ is a* cycle *(resp.* simple cycle*) in $\mathcal{A}$ if $q \in Q$ exists such that $\delta(q, c) = q$ and the internal states crossed by the transition are different from q (resp. from each other and with respect to q).*
In addition, if c is a cycle in $\mathcal{A}$ and c is not simple, then there are $u_1, \ldots, u_{k+1} \in A^*$, cycles $c_1, \ldots, c_k \in A^+$ and positive integers $p_1, \ldots, p_k$ such that $c = u_1 c_1^{p_1} u_2 \cdots u_k c_k^{p_k} u_{k+1}$, where each u_i is the label of a path in which no state is crossed twice and $u_1, u_{k+1} \neq 1$. Furthermore, if $i, j \in \{1, \ldots, k\}$ exist such that $c_i = c_j$, with $j \geq i + 2$, $c_i \neq c_{i'}$, for $i < i' < j$, then $u_{i+1} \cdots u_j \neq 1$.

For example, given the finite state automaton $\mathcal{A}$ depicted in Figure 1, we have that *a*, *bb*, *bab*, *badb* are cycles, whereas *aa*, *baabbb*, *badab*, *ad* are not cycles.

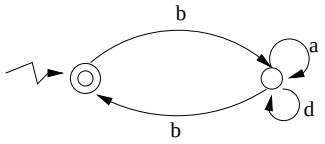

Fig. 1. Automaton for $(b(a + d)^*b)^*$

3 Linear splicing

3.1 Linear splicing systems

Below are three definitions of linear splicing systems, respectively by Head, Paun and Pixton. The difference between the three definitions depends on the biological phenomena that they want to model, but these biological motivations will not be discussed here.

`Head's definition [21].` A *Head splicing system* is a 4-uple $S_H = (A, I, B, C)$, where $I \subset A^*$ is a finite set of strings, called *initial language*, B and C are finite sets of triples (α, μ, β), called *patterns*, with $\alpha, \beta, \mu \in A^*$ and μ called the *crossing* of the triple. Given two words $u\alpha\mu\beta v$, $p\alpha'\mu\beta' q \in A^*$ and two patterns (α, μ, β) and (α', μ, β') that have the same crossing and are both in B or both in C, the splicing operation produces $u\alpha\mu\beta' q$ and $p\alpha'\mu\beta v$ (we also say that $\alpha\mu\beta$, $\alpha'\mu\beta'$ are *sites* of splicing).

`Paun's definition [36].` A *Paun splicing system* is a triple $S_{PA} = (A, I, R)$, where $I \subset A^*$ is a set of strings, called *initial language*, R is a set of *rules*

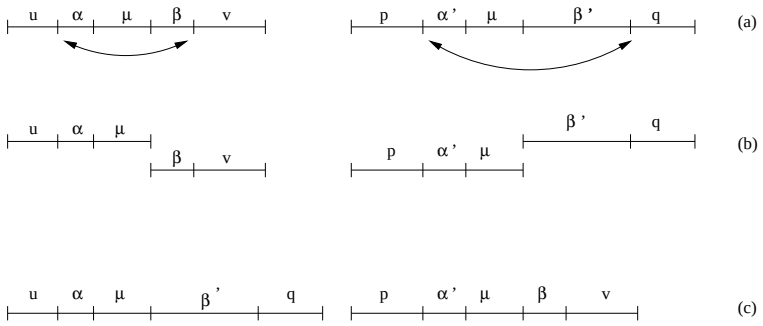

Fig. 2. *Head's splicing: (a) patterns recognition, (b) cut, (c) paste.*

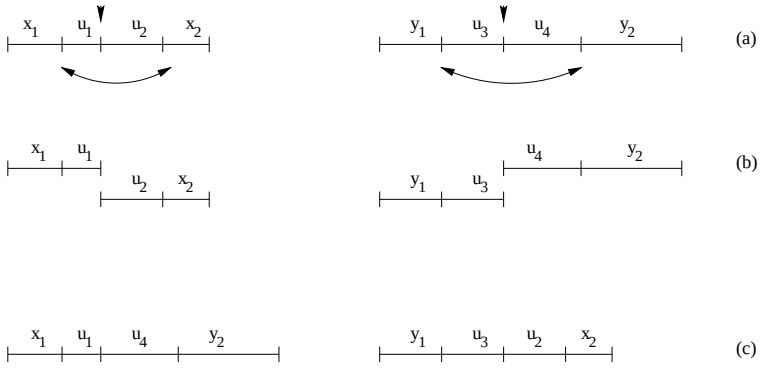

Fig. 3. *Paun's splicing: (a) sites recognition, (b) cut, (c) paste.*

$r = u_1|u_2\$u_3|u_4$, with $u_i \in A^*, i = 1,2,3,4$ and $|, \$ \notin A$. Given two words $x = x_1u_1u_2x_2, y = y_1u_3u_4y_2$, $x_1, x_2, y_1, y_2 \in A^*$ and the rule $r = u_1|u_2\$u_3|u_4$, the splicing operation produces $w = x_1u_1u_4y_2$ and $w' = y_1u_3u_2x_2$.

`Pixton's definition [44].` A *Pixton splicing system* is a triple $S_{PI} = (A, I, R)$, where $I \subset A^*$ is a set of strings, called *initial language*, R is a finite collection of rules $r = (\alpha, \alpha'; \beta)$, $\alpha, \alpha', \beta \in A^*$. Given two words $x = \epsilon\alpha\eta$, $y = \epsilon'\alpha'\eta'$ and the rule $r = (\alpha, \alpha'; \beta)$, the splicing operation produces $w = \epsilon\beta\eta'$ and $w' = \epsilon'\beta\eta$.

Given a splicing system S_X, with $X \in \{PA, PI\}$, $(x, y)\vdash_r (w', w'')$ indicates that the two strings w', w'' are produced from (or spliced by) x, y when a rule r is applied. Let $L \subseteq A^*$ and let S_X be a splicing system. We denote

$$\sigma'(L) = \{w, w' \in A^* \mid (x, y)\vdash_r (w, w'),\ x, y \in L, r \in R\}$$

We can analogously define $\sigma'(L)$ for Head's splicing systems S_H by substituting R with the sets B, C of triples and r with two patterns. Thus, we also denote

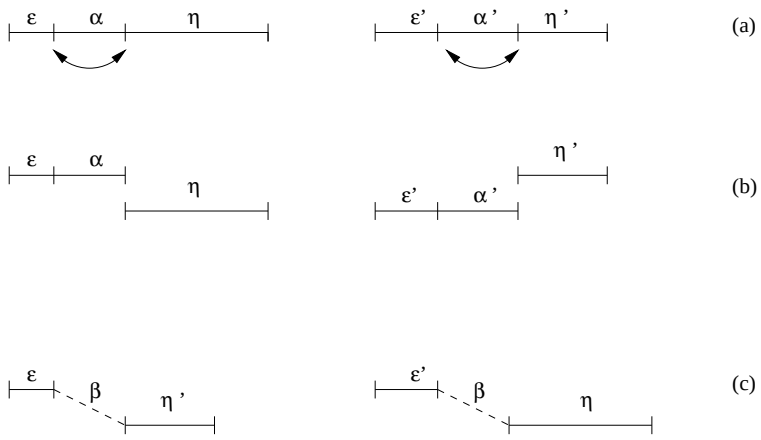

Fig. 4. *Pixton's splicing: (a) sites recognition, (b) cut, (c) paste with substitution.*

$$\begin{aligned} \sigma^0(L) &= L, \\ \sigma^{i+1}(L) &= \sigma^i(L) \cup \sigma'(\sigma^i(L)),\ i \geq 0, \\ \sigma^*(L) &= \bigcup_{i \geq 0} \sigma^i(L). \end{aligned}$$

Definition 2 (Linear splicing language). *Given a splicing system S_X, with initial language I and $X \in \{H, PA, PI\}$, the language $L(S_X) = \sigma^*(I)$ is the language generated by S_X. A language L is S_X generated if a splicing system S_X exists such that $L = L(S_X)$.*

The problem of comparing the computational power of the three types of splicing operations is examined here below.

Problem 1. Given $L = L(S_X)$, with $X \in \{H, PA, PI\}$, does S_Y exist, with $Y \in \{H, PA, PI\} \setminus X$, such that $L = L(S_Y)$?

Problem 1 has been solved for finite linear splicing systems (I, R, B, C finite sets): the computational power increases when we substitute Head's systems with Paun's systems and Paun's systems with Pixton's systems. This result has been demonstrated in [10], together with examples of regular languages separating the above mentioned classes of splicing languages.

3.2 Computational power of H-systems

In this section we will describe the well-known results on the computational power of (iterated) linear splicing systems. Our study will consider having an

unlimited number of copies of each word in the set, so that a pair of strings (x, y) can generate more than one pair of words with the use of different rules. We will consider words which model single stranded structures of DNA instead of double stranded structures. A model for the latter could be domino languages, as already used in [12]. It would be more realistic to consider the bidimensional structure of DNA, but this is too difficult. Furthermore, as observed in [21, 38], there is no loss of information if the double stranded DNA molecule is identified with a single stranded DNA sequence.

We repeat that the computational power of splicing systems depends on which level of the Chomsky hierarchy I, R belong to. Precisely, let F_1, $F_2 \in \{FIN, REG, LIN, CF, CS, RE\}$, we denote $H(F_1, F_2) = \{\sigma^*(I) \mid S_{PA} = (A, I, R)$ with $I \in F_1, R \in F_2\}$ the class of languages generated by Paun splicing systems, where the initial language I belongs to F_1 and the set of rules R belongs to F_2. In a similar way, we can give analogous definitions with respect to Head's or Pixton's splicing systems, but they will be not used in the next part of this paper.

The following table, reported in [26], collects the results we already know on the level of the Chomsky hierarchy $H(F_1, F_2)$ belongs to. These results have been obtained in separate papers. Furthermore, if we look at the table, either $H(F_1, F_2)$ is a specific class of languages in the above hierarchy (one entry in the table), or it is strictly intermediate between two of them (two entries in the table).

I \ R	FIN	REG	LIN	CF	CS	RE
FIN	FIN,REG	FIN,RE	FIN,RE	FIN,RE	FIN,RE	FIN,RE
REG	REG	REG,RE	REG,RE	REG,RE	REG,RE	REG,RE
LIN	LIN, CF	LIN,RE	LIN,RE	LIN,RE	LIN,RE	LIN,RE
CF	CF	CF, RE	CF, RE	CF, RE	CF, RE	CF, RE
CS	CS, RE	CS, RE	CS, RE	CS, RE	CS, RE	CS, RE
RE	RE	RE	RE	RE	RE	RE

The above table lets us answer some classical decision problems regarding computational models, namely whether or not emptyness, membership and equivalence are decidable or undecidable for a given language generated by a splicing system. Indeed, for a given splicing system, the answer to these questions directly follows as a byproduct of the computational power of the equivalent classical model. In particular, it is obvious that the above mentioned questions are decidable for splicing systems which are equivalent to finite state automata.

If we look at the table, we see that $H(REG, FIN) = REG$ for Paun's splicing systems. The same equation for Pixton's splicing systems is stated below and was presented for the first time in [44].

Theorem 1. *[44] (Regularity preserving theorem) Given a Pixton splicing system $S_{PI} = (A, I, R)$, where $I \in REG$ and $R \in FIN$, we have $L(S_{PI}) \in REG$.*

Theorem 1 has been proved after partial results on the characterization of the subclass of the splicing systems which are equivalent to finite state automata. These partial results can be found in [12, 14, 21, 32, 42]. In some of these papers, many efforts have been made in effectively contructing finite state automata associated to a given splicing system. In particular, Head and Gatterdam studied a specific type of finite Head splicing systems (persistent in [21]; persistent, reduced and crossing disjoint in [16]), whereas non persistent Head splicing systems have been studied by Culik and Harju [12], via domino languages. For Head's splicing systems S_H with regular initial language and a finite set of rules, Kim showed the regularity of $L(S_H)$ by designing an algorithm which constructs a finite state automaton recognizing the splicing language generated by S_H [32]. An extension of Theorem 1 for both Paun's and Pixton's definitions to full AFL is given by Pixton in Theorem 2. Observe that REG, CF and RE are full AFL.

Theorem 2. *[26, 43] If FA is a full AFL, then $H(FA, FIN) \subseteq FA$.*

Finally, in [9] the authors show how REG coincides with finite marked splicing systems (systems which have a different definition of the splicing operation and with "marked rules").

3.3 Finite linear splicing systems

The already known results stress that $H(FIN, FIN)$ is strictly intermediate between FIN and REG [26]. Infact, $(aa)^*b$ is a regular language which belongs to $H(FIN, FIN)$ (Example 1). Furthermore, no finite linear splicing system can generate $(aa)^*$. This result has been proved for Head's systems in [16] and it is easy to state it for Pixton's systems. Indeed, on the contrary, let us suppose that there is a finite Pixton splicing system $S_{PI} = (A, I, R)$ such that $L(S_{PI}) = (aa)^*$. It is clear that each rule in R must have the form $(a^i, a^j; a^k), i, j, k \geq 0$. Now, for each $i, j, k \in \boldsymbol{N}$, let us consider $n, m \in \boldsymbol{N}$ with $2n \geq i$ and $2m \geq j$. For each $t, t' \in \boldsymbol{N}$ with $0 \leq t \leq 2n - i$, $0 \leq t' \leq 2m - j$, the rule $(a^i, a^j; a^k)$ applied to $(aa)^n$, $(aa)^m$ generates $a^{2n+(k-i)+(t'-t)}$ and $a^{2m+(k-j)+(t-t')}$. Thus, we can always choose n, m, t, t' so that $2n + (k - i) + (t' - t)$ or $2m + (k - j) + (t - t')$ is an odd number, i.e., $a^{2n+(k-i)+(t'-t)}$ or $a^{2m+(k-j)+(t-t')}$ does not belong to $(aa)^*$.

Now two problems arise and, as we have already said, Problem 3 is the special question we will focus on in this paper.

Problem 2. Given $L \in REG$, can we decide whether $L \in H(FIN, FIN)$?

Problem 3. Characterize regular languages which belong to $H(FIN, FIN)$.

Kim has faced Problem 2 for Head's systems in [31]. In particular, he designed an algorithm that, for a given regular language L and a given finite set of triples X, allows us to decide whether a (initial) language $I \subseteq L$ exists such that L is generated by $S_H = (A, I, X)$ (Kim's algorithm does not need to separate the triples into the two disjoint sets B and C).

Another partial result has been provided by Head in [22]. In that paper, the author considers the generating power of the (Paun) finite splicing systems with a *one-sided context*, i.e., each rule has the form $u|1\$v|1$ (resp. $1|u\$1|v$). It is also supposed that for each rule having the above form, rules $u|1\$u|1$, $v|1\$v|1$ (resp. $1|u\$1|u$, $1|v\$1|v$) are also in R (reflexive hypothesis). Head proved that we can decide whether a regular language is generated by one-sided context splicing systems. Very recently, Goode and Pixton showed that we can decide whether a regular language is generated by a finite Paun linear splicing system [17, 25]. Their characterization uses the notions of constant and syntactic monoid (see Section 2.1).

As far as Problem 3 is concerned, efforts have been made in comparing the class of languages generated by finite splicing systems with already known subclasses of regular languages (we refer to [34, 41] for the definitions of these subclasses). An example worth being mentioned is that $H(FIN, FIN)$ is not comparable with LT (locally testable) languages [4].

Head has given different characterizations of the family of SLT (strictly locally testable) languages. He has proved that this family of languages coincides with the family of languages generated by Head's splicing systems with the triples presenting the form $(1, x, 1)$ (*null context splicing systems* - *NCH*) [21]. He also pointed out in [23] that the same class of SLT languages is the union of the families of languages generated by a special hierarchy of splicing systems (*simple H-systems* - *SH*, each crossing of a triple is a letter), introduced in [35] as a subclass of NCH systems. We can decide whether a regular language is generated by SH systems and in the case of unary languages $L \subseteq a^*$, they have a very simple regular expression ($L = a^*$ or $L = a^+$).

In [18] simple H-systems are generalized by considering *semi-simple Paun splicing systems* where all rules have the form $a|1\$b|1$, $a, b \in A$. These systems are a particular case of the Paun systems with one-sided context, and by applying a result in [22], the authors demonstrated that semi-simple Paun splicing languages are SLT languages.

In [7] the authors introduced a family of languages, called *marker languages*, and showed that marker languages are generated by (Paun) finite linear splicing systems. Let us sketch their definition that uses the above mentioned classical notions of syntactic congruence and constants. Consider a regular language L and the minimal finite state automaton $\mathcal{A}$ recognizing it. Suppose that a word w and a syntactic congruence class $[x]$ exists so that in the transition diagram of $\mathcal{A}$ we find only a path π, starting from $q \in Q$, with label wx and this is the only path that allows us to read the labels w and x. Also suppose that x is the label of a closed path. In addition, when we consider the transition graph of the classical automaton recognizing the reverse of L, the part of this transition graph starting from q is deterministic. Thus, $w[x]$ is a *marker*. Let q_0, q_f be the initial state and a final state in $\mathcal{A}$, respectively. Then we can generate, through a finite linear splicing system the language $L(w[x])$ of all the words in L which are labels of paths going from q_0 to q_f and having wx' as a factor, with x' in $[x]$.

Theorem 3. *[7] Let L be a regular language and let $w[x]$ be a marker for L. Then there is a finite splicing system $S_{PA} = (A, I, R)$ such that $L(w[x]) = \{z \in L \mid z = z_1 m z_2, m \in w[x]\} = L(S_{PA})$.*

Example 1. The regular language $L = (aa)^*b$ is a marker language (with marker b). L is generated by $S_{PA} = (A, I, R)$, where $I = \{b, aab\}$ and $R = \{1 \mid aab\$1 \mid b\}$.

The construction of a splicing system generating $L(w[x])$, is indicated in the proof of Theorem 3. This construction is obtained thanks to a relationship between syntactic congruence and linear splicing. A generalization of Theorem 3 will be discussed in a future paper [8].

4 Circular splicing

4.1 Models

As in linear splicing, there are at least three basic definitions for circular splicing; many variants are also known, some are obtained by adding new hypotheses on R.

`Head's definition [48].` A *Head circular splicing system* is a 4-uple $SC_H = (A, I, T, P)$, where $I \subseteq {}^{\sim}A^*$ is the *initial* circular language, $T \subseteq A^* \times A^* \times A^*$ and P is a binary relation on T, such that, if $(p, x, q), (u, y, v) \in T$ and $(p, x, q)P(u, y, v)$ then $x = y$. Thus, given ${}^{\sim}hpxq$, ${}^{\sim}kuxv \in {}^{\sim}A^*$ with $(p, x, q)P(u, x, v)$, the splicing operation produces ${}^{\sim}hpxvkuxq$.

`Paun's definition [26, 42].` A *Paun circular splicing system* is a triple $SC_{PA} = (A, I, R)$, where $I \subseteq {}^{\sim}A^*$ is the *initial* circular language and $R \subseteq A^*|A^*\$A^*|A^*$, with $|, \$ \notin A$, is the set of rules. Then, given a rule $r = u_1|u_2\$u_3|u_4$ and two circular words ${}^{\sim}hu_1u_2$,${}^{\sim}ku_3u_4$, the rule cuts and linearizes the two strings obtaining u_2hu_1 and u_4ku_3, and pastes and circularizes them obtaining ${}^{\sim}u_2hu_1u_4ku_3$.

`Pixton's definition [44].` A *Pixton circular splicing system* is a triple $SC_{PI} = (A, I, R)$ where A is a finite alphabet, $I \subseteq {}^{\sim}A^*$ is the *initial* circular language, $R \subseteq A^* \times A^* \times A^*$ is the set of rules. R is such that if $r = (\alpha, \alpha'; \beta) \in R$ then there exists β' such that $\overline{r} = (\alpha', \alpha; \beta') \in R$. Thus, given two circular words ${}^{\sim}\alpha\epsilon$,${}^{\sim}\alpha'\epsilon'$, the two rules $r, \overline{r}$ cut and linearize the two strings, obtaining $\epsilon\alpha, \epsilon'\alpha'$ and then paste, substitute and circularize them, producing ${}^{\sim}\epsilon\beta\epsilon'\beta'$.

As in the linear case, we suppose we have an unlimited number of copies of each word in the set, so that a pair of words (x, y) can generate more than one pair of words by applying different rules. Usually T and R are finite sets in the three definitions of the circular splicing indicated above. This paper is limited to examining finite (resp. regular) circular splicing systems according to Paun's and Pixton's definitions, without additional hypotheses (even without self-splicing, which is usually incorporated in Paun's definition). We recall that a finite (resp. regular) circular splicing system is a circular splicing system with a circular finite (resp. regular) initial language.

Given a circular splicing system SC_X, with $X \in \{PA, PI\}$, $(w', w'')\vdash_r z$ indicates that z is spliced by w', w'', through the use of rule r, with respect to one of the two definitions. It is clear that in Pixton's systems, the splicing operation is the combined action of a pair of rules, r and $\bar{r}$, not necessarily different.

Given a circular splicing system SC_X, with $X \in \{PA, PI\}$, and a language $C \subseteq {}^{\sim}A^*$, we denote

$$\sigma'(C) = \{z \in {}^{\sim}A^* \mid \exists w', w'' \in C,\ \exists r \in R.\ (w', w'')\vdash_r z\}$$

We can analogously define $\sigma'(C)$ for Head's circular splicing systems SC_H by substituting R with the set T of triples and r with two triples. Thus, we define $\sigma^*(C)$ as in the linear case.

Definition 3 (Circular splicing language). *Given a splicing system SC_X, with initial language $I \subseteq {}^{\sim}A^*$ and $X \in \{H, PA, PI\}$, the circular language $C(SC_X) = \sigma^*(I)$ is the language generated by SC_X. A circular language C is C_X generated (or C is a circular splicing language) if a splicing system SC_X exists such that $C = C(SC_X)$.*

Naturally, we can compare the three definitions of the circular splicing and Proposition 1 below shows that the computational power of circular splicing systems increases when we substitute Head's systems with Paun's systems and Paun's systems with Pixton's systems.

Proposition 1. *[5] If $C \subseteq {}^{\sim}A^*$ is generated by a finite Head circular splicing system then C is C_{PA} generated. Moreover, if $C \subseteq {}^{\sim}A^*$ is generated by a finite Paun circular splicing system, then C is C_{PI} generated.*

Example 2. We can check that $\{{}^{\sim}(aa)^n | n \geq 0\}$ is C_{PA} generated, by choosing $SC_{PA} = (A, I, R)$, with $A = \{a\}, I = \{{}^{\sim}aa\} \cup 1, R = \{aa|1\$1|aa\}$ [6] (see [48] for Head's systems).
Moreover, let us consider $C = {}^{\sim}(aa)^*b$. We can see that C cannot be C_H or C_{PA} generated (see also [6]). On the contrary, C is C_{PI} generated if we choose $SC_{PI} = (A, I, R)$, with $A = \{a, b\}$, $I = \{{}^{\sim}b, {}^{\sim}a^2b, {}^{\sim}a^4b\}$, $R = \{(a^2, a^2b; a^2), (a^2b, a^2; 1)\}$.

Obviously, we can find analogies and differences between linear and circular splicing. Therefore, we must consider languages which are closed under the conjugacy relation and the action of these two types of splicing operations on them. As a matter of fact, these two types of splicing operations are not comparable: we have already seen that the regular language $(aa)^*$ is not generated by a finite linear splicing system (see Section 3.3), but the corresponding circular language ${}^{\sim}(aa)^*$ is C_{PA} generated (Example 2). Furthermore, the regular language $(aa)^*b$ is generated by a finite linear splicing system (Example 1), but the corresponding circular language ${}^{\sim}((aa)^*b)$ cannot be C_{PA} generated [6] . Some ideas on this matter can be found in the proof of Theorem 4.

Notice that any set R of rules in Paun's or Pixton's circular splicing system is implicitly supposed to be symmetric, i.e., for each rule $r = u_1|u_2\$u_3|u_4$ (resp.

$r = (\alpha, \alpha'; \beta)$) in the splicing system SC_{PA} (resp. SC_{PI}), rule $\overline{r} = u_3|u_4\$u_1|u_2$ (resp. $\overline{r} = (\alpha', \alpha; \beta')$) belongs to R.

In [26, 42–44] additional hypotheses are considered for Paun's or Pixton's definitions, namely

Hypothesis 1. R is a *reflexive* scheme, i.e., for each rule $u_1|u_2\$u_3|u_4$ (resp. $(\alpha, \alpha'; \beta)$) in the splicing system SC_{PA} (resp. SC_{PI}), rules $u_1|u_2\$u_1|u_2$ and $u_3|u_4\$u_3|u_4$ (resp. $(\alpha, \alpha; \alpha)$ and $(\alpha', \alpha'; \alpha')$) belong to R.

Hypothesis 2. *Self-splicing.* Self-splicing is defined in a splicing system SC_{PA} (resp. SC_{PI}) producing ${}^{\sim}u_4xu_1$ and ${}^{\sim}u_2yu_3$ from ${}^{\sim}xu_1u_2yu_3u_4$ and the rule $u_1|u_2\$u_3|u_4$ (resp. ${}^{\sim}\beta\epsilon'$,${}^{\sim}\beta'\epsilon$ starting from ${}^{\sim}\alpha\epsilon\alpha'\epsilon'$ and the rules $(\alpha, \alpha'; \beta)$, $(\alpha', \alpha; \beta')$).

We will end this section with a very short survey on other variants of these models. One main variant, described in [20, 44, 48], considers the so-called *mixed splicing*, in which both linear and circular words are allowed. In [48] the authors proposed several splicing operations, either acting on circular strings only (for instance self-splicing) or dealing with mixed splicing.

In [49] a new type of circular extended splicing system (CH systems) is proposed and the universality of this model is demonstrated. In particular, linear and circular strings are involved, without multiplicity, and with I, R both finite sets. Starting from [49], the CH systems have been considered in [47] as components of a Communicated Distributed circular H-system, i.e., the circular splicing counterpart of the parallel communicating grammar systems.

4.2 Computational power of circular splicing systems

As in the linear case, one main question is the investigation of the computational power of circular splicing systems. Once again, we denote $C(F_1, F_2) = \{\sigma^*(I) \mid SC_{PA} = (A, I, R) \text{ with } I \in F_1^{\sim}, R \in F_2\}$ where F_1, F_2 are families of languages in the Chomsky hierarchy. Clearly, the same definition may be given for the other two definitions of circular splicing systems.

Few results are known and the problem can be split into several subproblems, each of them concerning a level of the Chomsky hierarchy containing I and by adding or not adding Hypothesis 1 and/or Hypothesis 2.

As in the linear case, in this paper we will focus on finite circular splicing systems and on their relationship with regular circular languages. In other words, we will consider the following problem.

Problem 4. Characterize $REG^{\sim} \cap C(FIN, FIN)$.

Note that an answer to this problem does not describe the computational power of finite circular splicing systems. Indeed, it is known that in contrast with the linear case, $C(FIN, FIN)$ is not intermediate between two classes of

languages in the Chomsky hierarchy. For example, $\sim a^n b^n$ is a circular context-free language which is not a circular regular language (since its full linearization is not regular). Moreover, it is quite easy to check that $\sim a^n b^n$ is generated by $SC_H = (A, I, R)$ with $I = 1 \cup \sim ab$ and $R = \{(a, 1, b), (b, 1, a)\}$ (also see [48]).

On the other hand, $\sim (aa)^* b$ is a regular circular language that cannot be generated by a finite SC_{PA} splicing system (Example 2), whereas the circular regular language $\sim((A^2)^* \cup (A^3)^*)$ cannot be generated by a finite SC_{PI} splicing system (without any additional hypotheses) [7]. Furthermore, there are examples of circular regular languages which cannot be generated by finite circular splicing systems with additional Hypotheses 1-2 [44, 48]. It is not yet clear if $C(FIN, FIN)$ contains any context-sensitive or recursive enumerable language which is not context-free.

One of the first results regarding the computational power of Head's circular splicing systems has been proved in [48]. This result states that a finite Head circular splicing system with rules having the form $(1, x, 1)$, with $|x| = 1$, always generates regular circular languages. This result has been reinforced by Pixton as stated in Theorem 4 below.

Theorem 4. *[44] Let $SC_{PI} = (A, I, R)$ be a circular splicing system with I a circular regular language and R reflexive and symmetric. Then $C = C(SC_{PI})$ is regular.*

A similar closure property of splicing has been demonstrated for Paun's systems with I in a full AFL [26, 42, 43]. This result is the counterpart for circular splicing of Theorem 2. In [44] a link between regular circular languages and automata is pointed out, along with examples of circular splicing languages which are not regular and which are generated by finite circular splicing systems, leading to Hypotheses 1, 2. Proof of Theorem 4 is obtained by indicating the construction of an automaton which accepts the circular splicing language. In [43] the author gives a simpler demonstration of Theorem 4 together with a version of this theorem for mixed splicing.

5 Finite circular splicing systems

5.1 Star languages and codes

We will now present a class of regular languages whose circularization is C_{PA} generated [5]. As we have already said, we will consider languages L closed under the conjugacy relation, and the action of circular splicing over their circularization $\sim L = C$.

Definition 4. *[5] A* star language *is a language $L \subseteq A^*$ which is closed under the conjugacy relation and such that $L = X^*$, where X is a regular language.*

As already observed, the crucial point in generating languages through splicing is the existence of a finite set of rules producing an unbounded number of occurrences of labels c of a closed path in $\mathcal{A}$. Even if we limit ourselves to a fixed

automaton $\mathcal{A}$, c can be a cycle in $\mathcal{A}$ and also the label of another (not necessarily closed) path in $\mathcal{A}$. In [5] the authors overcome this difficulty, by considering star languages L satisfying the requirement that a special representation of each cycle c (fingerprint of c) belongs to L.

Given $L = L(\mathcal{A})$, for every cycle c in $\mathcal{A}$, the *fingerprint* $F(c)$ of c is inductively defined as follows: if c is a simple cycle, then $F(c) = \{c\}$; if c is not simple (i.e., $c = u_1 c_1^{p_1} u_2 c_2^{p_2} \cdots u_k c_k^{p_k} u_{k+1}$, where u_i are labels of transitions in which no state is crossed twice, $u_1, u_{k+1} \neq 1$, c_i are cycles, $p_i \geq 1$, and if $c_i = c_j$, with $j \geq i+2$, $c_i \neq c_{i'}$, for $i < i' < j$, then $u_{i+1} \cdots u_j \neq 1$), $F(c_1), \cdots, F(c_k)$ are fingerprints of $c_1, \ldots, c_k$, then $F(c) = u_1 F(c_1) u_2 F(c_2) \cdots u_k \ F(c_k) u_{k+1}$.

A language $L = L(\mathcal{A})$ is a *fingerprint closed language* with respect to $\mathcal{A}$ whenever $\mathcal{C}(L) = \bigcup_{c \ cycle} F(c) \subseteq L$, where $\mathcal{C}(L)$ is the union of the fingerprints $F(c)$ for every cycle c in $\mathcal{A}$.

Theorem 5. *[5, 6] Let X^* be a fingerprint closed star language with respect to a trim deterministic automaton $\mathcal{A}$ recognizing X^*. Then ${}^{\sim}X^* \in C(Fin, Fin)$.*

Proof. **(Sketch)** Let $\mathcal{C}(X^*)$ be the set of the fingerprints of X^* with respect to $\mathcal{A}$ and let $I_1 = \mathcal{C}(X^*) \cup I_2$, $I_2 = \{w \in A^* \mid \delta(q_0, w) \in F$ and $[\forall x, z \in A^*, c \in A^+$, $q \in Q$ if $w = xc^k z$ with $\delta(q_0, x) = q$, $\delta(q, c) = q$, then $c = F(c)$ is a cycle and $k = 1]\}$. Let us define $SC_{PA} = (A, I, R)$ where $I = {}^{\sim}I_1$ and $R = \{1|1\$1|\beta \ ; \ \beta \in F(c), \ F(c) \in \mathcal{C}(X^*), \ c$ cycle$\}$. The languages I_1, I, R are finite sets (I_1 is a set of labels of paths in $\mathcal{A}$ in which no state is crossed more than twice). We can prove that ${}^{\sim}X^* \subseteq C(SC_{PA})$ by induction on the length of words in ${}^{\sim}X^*$. In addition, $C(SC_{PA}) \subseteq {}^{\sim}X^*$, since $I \subseteq {}^{\sim}X^*$ and X^* is a submonoid closed under the conjugacy relation.

As an example of fingerprint closed star languages, let us consider free monoids X^* generated by a regular group code X. We recall that X is a *group code* if a group G and a surjective morphism $\phi\colon A^* \to G$ exist such that $X^* = \phi^{-1}(H)$, where H is a subgroup of G [2]. As an example, uniform codes A^d, $d \geq 1$ are regular group codes. We also know that a free monoid X^* is closed under the conjugacy relation if and only if X is a regular group code [2, 3, 45]. So, X^* is a star language. Furthermore, group codes are biprefix codes. Therefore, it is easy to see that for every cycle c in a trim deterministic finite state automaton $\mathcal{A}$ recognizing X^*, we have that $c \in X^*$. As a consequence star languages X^*, with X being a regular group code, are fingerprint closed languages.

5.2 The case of a one-letter alphabet

Contrarily to the general case, the structure of unary languages generated by finite (Paun) circular splicing systems can be described exhaustively. Each language $L \subseteq a^*$ is closed under the conjugacy relation since we have ${}^{\sim}w = \{w\}$, so $L = {}^{\sim}L$ and we can identify each word (resp. language) with its circularization.

Below, $\boldsymbol{Z}_n$ indicates the *cyclic group* of order $n \in \boldsymbol{N}$, $Fact(L)$ is the set of all the factors of the elements of $L \subseteq A^*$ and, for $D \subseteq \boldsymbol{N}$, we can set $a^D = \{a^d \mid d \in D\}$.

Proposition 2. *[6] A subset $L =^{\sim} L$ of a^* is C_{PA} generated if and only if there exists a finite subset L_1 of a^*, a positive integer n and a (periodic) subgroup G' of $\boldsymbol{Z}_n$ such that $L = L_1 \cup (a^G)^+$, where G is a set of representatives of the elements in G' and $a^G \cap Fact(L_1) = \emptyset$. Furthermore, $G = \{m + \lambda_m n \mid m \in G'\}$ with $\lambda_m = min\{\mu \mid a^{m+\mu n} \in (a^G)^+\}$.*

The reader should compare this result with the known characterization of unary regular languages: a language $L \subseteq a^*$ is regular if and only if L is ultimately periodic, i.e., there exist $h \geq 0$ and $g > 0$ such that for all $m \geq h$ we have $a^m \in L$ if and only if $a^{m+g} \in L$ [19]. Furthermore, Proposition 2 can be easily extended to a particular class of languages over A with $|A| > 1$, i.e., languages with the form $L = A^J = \bigcup_{j \in J} A^j$, with $J \subseteq \boldsymbol{N}$ [7].

A decidability question obviously follows this result. If $L \subseteq a^*$, can we decide whether L is a C_{PA} generated language? If no hypothesis is made over L, the answer to this question is no, according to the Rice theorem [28]. Proposition 3 gives a positive answer to this question when L is supposed to be a regular language.

We recall that results already obtained describe the structure of every deterministic finite state automaton $\mathcal{A}= (\{a\}, Q, \delta, q_0, F)$ recognizing a regular language $L \subseteq a^*$: the transition graph of $\mathcal{A}$ has a (frying-)pan shape. For example, consider the regular C_{PA} generated language $L = \{a^3, a^4\} \cup \{a^6, a^{14}, a^{16}\}^+$. A deterministic finite state automata recognizing L is depicted in Figure 5.

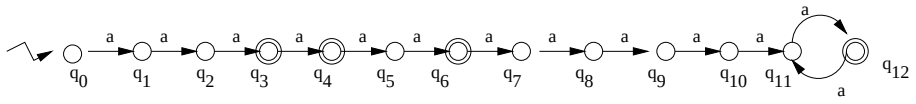

Fig. 5. Automaton for $L = \{a^3, a^4\} \cup \{a^6, a^{14}, a^{16}\}^+$.

Thus, in Proposition 3 two conditions are given for the minimal finite state automaton recognizing L: we must check that $\mathcal{A}$ satisfies these two conditions in order to state that the language $L(\mathcal{A})$ recognized by the automaton $\mathcal{A}$ is a C_{PA} generated language. Intuitively, in the minimal automaton $\mathcal{A}$ recognizing a C_{PA} generated language $L = L(\mathcal{A})$, we have a unique final state $q_{n'}$ on the closed path in $\mathcal{A}$ and there exist $p, s \in \boldsymbol{N}$ such that $n' = ps$ and $p = v - l + 1$ is the length of this closed path. No conditions are given for the handle. Integers n (Proposition 2) and n' (Proposition 3) are related to each other since the proof of Proposition 3 points out that $n' = \lambda n$ with $\lambda \in \boldsymbol{N}$.

Proposition 3. *Let $L \subseteq a^*$ be a regular language. Let $\mathcal{A}= (\{a\}, Q, \delta, q_0, F)$ be the minimal finite state automaton recognizing L, where $Q = \{q_0, q_1, \ldots, q_v\}$, for each $i \in \{0, \ldots v-1\}$ we have $\delta(q_i, a) = q_{i+1}$ and there exists $l \in \{0, \ldots, v\}$ such that $\delta(q_v, a) = q_l$. We have that $L = L(\mathcal{A})$ is C_{PA} generated if and only if either $q_h \notin F$, for all $h \geq l$ or there exist $n', p, s \in \boldsymbol{N}$ such that $\mathcal{A}$ satisfies the two conditions that follow.*

1. $\{q_h \mid q_h \in Q,\ h \geq l\} \cap F = q_{n'}$.
2. $n' = ps$ *and* $v - l + 1 = p$.

For example, the (minimal) automaton in Figure 5 satisfies the two conditions reported in Proposition 3.

This section ends with a result concerning the descriptional complexity of a circular splicing system generating a circular language $L \subseteq a^*$. The notion of *minimal splicing system* was naturally introduced in [36, 40] as a counterpart of the minimal automaton for regular languages. Here the minimality of the system could be referred to the cardinality of the set of the rules (resp. triples) or to the length of the rules (resp. triples). Some initial steps towards this notion and for Head's linear splicing systems were implicitely taken in [16]. Superfluous patterns were individuated and an algorithm reducing the size of $B \cup C$ without changing the generated language, was designed. In this sense a result for one-letter alphabet languages and Paun's systems has been achieved in Theorem 6.

Theorem 6. *[6] Let $L \subseteq a^*$ be a C_{PA} generated language. Then, there is a (minimal) circular finite splicing system $SC_{PA} = (A, I, R)$ generating L with either $R = \emptyset$ or $R = \{r\}$ containing only one rule.*

References

1. L. M. Adleman (1994), Molecular computation of solutions to combinatorial problems, *Science* **226**, pp. 1021 – 1024.
2. J. Berstel, D. Perrin (1985), *Theory of codes*, Academic Press, New York.
3. J. Berstel, A. Restivo (1981), Codes et sousmonoides fermes par conjugaison, *Sem. LITP* **81-45**, 10 pages.
4. P. Bonizzoni, *private communication.*
5. P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza (2001), DNA and circular splicing, *in* "DNA 2000", Lecture Notes in Computer Science **2054**, pp. 117–129, Springer-Verlag.
6. P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza (2001) Circular splicing and regularity, *submitted.*
7. P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza (2002), On the power of linear and circular splicing, *submitted.*
8. P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza (2002), Linear splicing and regular languages, *manuscript.*
9. P. Bonizzoni, C. Ferretti, G. Mauri (1998), Splicing systems with marked rules, *Romanian J. of Information Science and Technology*, Vol. 1, No. 4, pp. 295 – 306.
10. P. Bonizzoni, C. Ferretti, G. Mauri, R. Zizza (2001), Separating some splicing models, *Information Processing Letters* **76 (6)**, pp. 255 – 259.
11. C. Choffrut, J. Karhumaki (1996), Combinatorics on Words, *in* "Handbook of Formal Languages" (G. Rozenberg & A. Salomaa, Eds.), Vol. 1, pp. 329–438, Springer-Verlag.
12. K. Culik, T. Harju (1991), Splicing semigroups of dominoes and DNA, *Discrete Applied Mathematics*, **31**, pp. 261 – 277.

13. A. De Luca, A. Restivo (1980), A characterization of strictly locally testable languages and its application to subsemigroups, *Information and Control*, **44**, pp. 300 – 319.
14. K.L. Denninghoff, R.W. Gatterdam (1989), On the undecidability of splicing systems, *Intern. J. Computer Math.*, **27**, pp. 133 – 145.
15. R. Freund, L. Kari, G. Paun (1999), DNA Computing based on splicing: the existence of universal computers, *Theory of Computing Systems*, **32**, pp. 69 – 112.
16. R. W. Gatterdam (1992), Algorithms for splicing systems, *SIAM Journal of Computing*, **21:3**, pp. 507 – 520.
17. E. Goode, T. Head, D. Pixton (2002), *private communication.*
18. E. Goode, D. Pixton (2001), Semi-simple splicing systems, *in* "Where Mathematics, Computer Science, Linguistics and Biology Meet" (C. Martin-Vide & V. Mitrana, Eds.), Kluwer Academic Publ., Dordrecht, pp. 343 – 357.
19. M. H. Harrison (1978), Introduction to Formal Language Theory, Addison Wesley.
20. T. Head (1992), Splicing schemes and DNA, *in* "Lindenmayer Systems: Impacts on Theoretical Computer Science and Developmental Biology", Springer-Verlag, Berlin, pp. 371 – 383.
21. T. Head (1987), Formal Language Theory and DNA: an analysis of the generative capacity of specific recombinant behaviours, *Bull. Math. Biol.* **49**, pp. 737 – 759.
22. T. Head (1998), Splicing languages generated with one sided context, *in* "Computing with Bio-molecules: Theory and Experiments" (Gh. Paun, Ed.), Springer-Verlag, Singapore.
23. T. Head (1998), Splicing representations of strictly locally testable languages, *Discrete Applied Math.*, **87**, pp. 139 – 147.
24. T. Head (1999), Circular suggestions for DNA Computing, *in* "Pattern Formation in Biology, Vision and Dynamics", (A. Carbone, M. Gromov, P. Pruzinkiewicz, Eds.), World Scientific, Singapore and London, to appear.
25. Tom Head (2002), Spicing systems: regular languages and below, *Extended abstract*, DNA8.
26. T. Head, Gh. Paun, D. Pixton (1996), Language theory and molecular genetics: generative mechanisms suggested by DNA recombination, *in* "Handbook of Formal Languages" (G. Rozenberg & A. Salomaa, Eds.), Vol. 2, pp. 295 – 360, Springer-Verlag.
27. T. Head, G. Rozenberg, R. Bladergroen, C. Breek, P. Lommerse, H. Spaink (2000), Computing with DNA by operating on plasmids, *BioSystems* **57**, pp. 87 – 93.
28. J.E. Hopcroft, R. Motwani, J.D. Ullman (2001), *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Mass.
29. N. Jonoska, S. Karl (1996), A molecular computation of the road coloring problem, *in* "Proc. of 2nd Annual Meeting on DNA Based Computers" (DIMACS Workshop), pp. 148 – 158.
30. L. Kari (1999), DNA computing: an arrival of biological mathematics, *The Mathematical Intelligence* **19:2**, pp. 9 – 22.
31. S.M. Kim (1997), An Algorithm for Identifying Spliced Languages, *in* "Proc. of Cocoon97", Lecture Notes in Computer Science **1276**, pp. 403 – 411.
32. S.M. Kim (1997), Computational modeling for genetic splicing systems, *SIAM Journal of Computing* **26**, pp. 1284 – 1309.

33. M. Lothaire (1983), *Combinatorics on Words*, Encyclopedia of Math. and its Appl., Addison Wesley Publishing Company.
34. R. McNaughton, S. Papert (1971), *Counter-Free Automata*, MIT Press, Cambridge, Mass.
35. A. Mateescu, Gh. Paun, G. Rozenberg, A. Salomaa (1998), Simple splicing systems, *Discr. Appl. Math.* **84**, pp. 145 – 163.
36. Gh. Paun (1996), On the splicing operation, *Discrete Applied Math.* **70**, pp. 57 – 79.
37. Gh. Paun (1996), Regular extended H systems are computationally universal, *Journal of Aut., Lang., Combinatorics.* **1:1**, pp. 27 – 36.
38. Gh. Paun, A. Salomaa (1996), DNA computing based on the splicing operation, *Math. Japonica*, **43:3**, pp. 607 – 632.
39. Gh. Paun, G. Rozenberg, A. Salomaa (1996), Computing by splicing, *Theoretical Computer Science* **168:2**, pp. 321 – 336.
40. G. Paun, G. Rozenberg, A. Salomaa (1998), *DNA computing, New Computing Paradigms*, Springer-Verlag.
41. D. Perrin (1990), Finite Automata, *in* "Handbook of Theoretical Computer Science" (J. Van Leeuwen, Ed.), Vol. B, pp. 1 – 57, Elsevier.
42. D. Pixton (2000), Splicing in abstract families of languages, *Theoret. Comp.Science* **234**, pp. 135 – 166.
43. D. Pixton (1996), Linear and Circular Splicing Systems, *in* "Proc. of 1st Int. Symp. on Int. in Neural and Biological Systems", pp. 181 – 188.
44. D. Pixton (1996), Regularity of splicing languages, *Discrete Applied Math.* **69**, pp. 101 – 124.
45. C. Reis, G. Thierren (1979), Reflective star languages and codes, *Information and Control* **42**, pp. 1 – 9.
46. M. P. Schützenberger (1975), Sur certaines opérations de fermeture dans le langages rationnels, *Symposia Mathematica* **15**, pp. 245 – 253.
47. R. Siromoney (2000), Distributed Circular Systems, *presented* at Grammar Systems 2000, Austria.
48. R. Siromoney, K.G. Subramanian, A. Dare (1992), Circular DNA and Splicing Systems, *in* "Proc. of ICPIA", Lecture Notes in Computer Science **654**, pp. 260 – 273, Springer-Verlag.
49. T. Yokomori, S. Kobayaski, C. Ferretti (1999), On the power of circular splicing systems and DNA computability, *in* "Proc. of IEEE Int. Conf. Evol. Comp." ICEC99.