1 Introduction: focus and aims

Biology and Computer Science were two different areas until biologists started to use Computer Science to solve problems in Biology (this is *Computational Biology*) and, few years ago, some computer scientists started to use, quite surprisingly, Biology to approach computer science's problems. The latter one is the general idea underlying *Molecular Computing* (for a quite exhaustive bibliography, see [10]).

The rapidly increasing of interest in Molecular Computing started with Adleman's article [1], where the author showed how to solve an instance of the Hamiltonian Path Problem, that is a NP-complete problem, using biological techniques. Starting from this communication, a lot of other "intractable" problems were formulated and solved in lab (for example, see [27]). This is commonly called 'wet' molecular computing, to highlight how the computer scientists enter in the laboratory to make algorithms in test tubes.

Thanks to this renewed interest in the Molecular Computing, new models of computation have been suggested and old ones have been rediscovered, like the *splicing systems*. The splicing systems had been already proposed in [22] as a formal counterpart of the recombination of DNA molecules, which are cut by restriction enzymes and pasted, if some biological conditions on the sticky ends happen, by ligase enzymes, producing new molecules. The idea is that these are generative mechanisms which start from a given set of strings and a given set of splicing rules in order to generate a language. All details about the steps from the biochemical processes to formal languages may be found in [22, 25, 36]. Based on this operation, the H-systems were introduced as unconventional models of computation (see [35] for a survey on the different models). This is another promising area in Molecular Computing that has grown as an area of Formal Language Theory. It is commonly called 'dry' molecular computing, to underline how the aim is that of starting from the biological mechanism in order to develop theoretical models of computations.

My thesis proposal is organized in three different parts. The present introduction is followed by other two sections dealing in turn with Wet Molecular Computing and Dry Molecular Computing, each handling the state of the art, personal contributions and future possible works. The Section 2, devoted to Wet Molecular Computing, offers an overview of the general technique used in molecular algorithms. Its both theoretical and pratical advantages, disadvantages and promises are briefly presented and unexpected links with the classical Theory of Codes [3] are given. In Section 3, I present the principal part of my thesis proposal. The splicing operation for both linear and circular strings is formally defined. I will give the state of the art paying particular attention to the relation existing between splicing systems and regular languages, this being my key research area. As to the latter, I will also present contributions already obtained that close open questions. Finally, I will mention very recently discovered links between splicing and codes, referring to a not much known and developed branch in Splicing Systems Theory.

2 Adleman's approach: molecular algorithms

"In 1959, Richard Feynman gave a visionary talk describing the possibility of building computers that were *sub–microscopic*. Despite remarkable progress in computer miniaturization, this goal is far to be achieved. Here, the possibility of computing directly with molecules is explored". This is the beginning of [1], where Adleman tells that the discovery of a mathematical behaviour in cells may be used to encode the "instantaneous description" of a Turing Machine and so encoding an algorithm in DNA sequences. Let me sketch his experiment without going into details.

Given a direct graph, a Hamiltonian path starts at a fixed vertex v_0, ends at a fixed vertex v_f and passes through the graph visiting every vertex exactly once. Given a direct graph G and two vertices v_0, v_f, it is known that the problem of deciding whether G has a Hamiltonian Path from v_0 to v_f is NP-complete. Adleman's approach is based on a nondeterministic algorithm for the Hamiltonian Path Problem. He choose an instance with 7 vertices and 14 edges, encoded in DNA sequences. Firstly, random paths through the graph were generated and then a verification phase was applied. This last is implemented by sequentially checking of the properties of the sequence that encodes a candidate solution path. Thus, paths that do not meet all requirements are discarded. For doing this, Adleman put DNA sequences, encoding the instance of the problem, in a test tube. Then he applied suitable chemical processes (as PCR, hybridization) simulating the computation and finally he extracted the solution of the problem, making it visible to nacked eye. Adleman after 1 second and in a fiftieth of a teaspoon, had the solution in the test tube, but together with 100000 wrong molecules. Then we was forced to spent 600000 seconds to extract the solution.

2.1 Advantages and problems: related and future works

This is commonly considered the birth of (wet) Molecular Computing. Beyond the real value of the experiment, this paper opened new interactions among different areas. Without going into details, we must note that molecular computers may be considered better than electronic ones with respect to energy efficiency, massive parallelism and density information storage [1, 11]. However, there are some limitations to Adleman's approach. Some of them have already been solved, others still remain, mainly due to the use of some reactions, that are, up to now, considered essential, as we will see later. As Hartmanis observed in [21], in Molecular Computing we are working on a new input and computational model, so we must define a new complexity measure, like the weight of the solution needed for the computation. He showed that repeating the Adleman's experiment on a graph with 200 vertices would mean using a solution whose weight is more than Earth's weight. Then, for the feasibility of molecular algorithms, it is decisive the proposal of alternative techniques during the extraction phase, trying to avoid PCR, the chief agent of time and weight increasing. Furthermore, every molecular algorithm makes massive use of hybridization. This last can happen even if two strands are not perfectly Watson-Crick complementary and this causes errors in computation, i.e. in a test tube wrong molecules are produced with respect to the problem encoding. This happens in PCR, as the copies are initiated by hybridization [28] as well as during the extraction phase, realized by magnetic beads.

Proposal. Errors and weight are the weak points in the molecular computation feasibility. In the literature there are some different attempts to find good encodings for DNA computing. Baum [2] suggests how to avoid unwanted hybridizations through suitable sequences, instead of the random ones chosen by Adleman, giving constraints on the encoding. It could be interesting to analize this problem in Symbolic Dynamic's terms, since one of the main result in [30] states when is possible to pass from an arbitrary space of sequences to a constrained one. Some papers propose different techniques to avoid errors caused by unwanted hybridizations [12, 13, 14]. Furthermore in [13] a new distance is proposed, more suitable than Hamming distance allowing the development of encodings to prevent errors. Generating "good" codes in polynomial time for arbitrary lengths is still an open problem and then it could be interesting to find the family of codes reliable for this encoding. To avoid unwanted hybridization, circular DNA may be useful. For example, the plasmides have a circular form and autoreplicates themselves, without errors and exponential weight increasing.

The *molecular algorithm* presented by Adleman has been generalized to other combinatorial problems. An interesting example is the molecular algorithm proposed in [27] for the Road Coloring Problem (RCP) (see, for example [37]). Let G be a finite direct graph such that all the vertices of G have the same out-degree and G be primitive (strongly connected and the g.c.d. of the cycle lengths is 1). The problem is to decide for such a graph G, whether a labelling turning G into a synchronizing deterministic automaton exists or not. We recall that an automaton is synchronizing if there exists a word w, called *synchronizing*, such that the state reached from any state q after reading w is indipendent from q. So a traveller lost on the graph G can use w to back "home", wherever he actually started from. Up to now, there is no known polynomial time algorithm that solves it. In [27], using the massive parallelism of DNA computation, a polynomial time molecular algorithm, for solving RCP for a given graph, is presented. The procedures involved in solving RCP extensively use the hybridization of DNA.

Proposal. We have said that, up to now, it is unknown whether RCP is decidible. A partial answer is given in [38], where the problem has been positively solved for graphs with particular additional conditions, by using a formulation in Symbolic Dinamic of a theorem on the non commutative factorization of polynomials [41]. This contribution is famous to be the best result obtained towards the Factorization Conjecture of Schützenberger [3], a very deeply investigated and open problem in the Theory of Codes. In this context [15] may be seen. The discovery of this connection is leading me to go on investigating the conjecture mentioned above, researching about new possible perspectives and approaches. It could be interesting to explore the possibility of formulating it in symbolic dinamic terms and to try to built molecular algorithms for code factorizations, looking at [27].

3 Head's approach: splicing systems

As just observed, Formal Language Theory already studied *splicing systems*, i.e. abstract models for the languages generated by strands of DNA under restriction and ligase enzymes reaction. These models were introduced by Head [22]. As DNA occurs in both linear and circular form, two kinds of splicing operation were defined to model different biological phenomena. The biological interpretation of linear splicing may be summed up as follows. Restriction enzymes are proteins that can recognize a specific pattern in a DNA sequence and then they cut the molecule inside the pattern, in a specific point. Ligase enzymes paste together these fragments, under chemical conditions. Thus, two DNA sequences are cut by two restriction enzymes and the fragments may be recombined so that new sequences are produced. Formally, the linear splicing operation concatenates a prefix of one string with a suffix of another string, under some conditions, represented into a (splicing) rule. On the other hand, the biological circular splicing occurs in a recombinant mechanism (transposition) between bacteria and plasmides; moreover, the replication is possible only if DNA has a circular form in the host cell. In this case, restriction enzymes cut two circular DNA sequences while ligase enzymes paste them together, producing a new circular DNA sequence. Formally, the circular splicing opens two circular strings and pastes them together, obtaining a circular string, under some conditions represented into a (splicing) rule.

The research about circular splicing is less developed than the linear case, probably because of the circularity of the words. Neverthless, circular words have already been treated in Formal Language Theory and we believe in a future development of this area, also for some links that we have just discovered with Theory of Codes.

3.1 Definitions

We summarize the three definitions of linear and circular splicing, pointing out that, before using a splicing rule r, the two words, spliced by r, are copied in order to be still used.

Linear case.

`Head's definition [22].` A *splicing system* $S_H = (V, I, B, C)$ consists of a finite alphabet V, a finite set $I \subseteq V^*$ of initial strings, finite sets B, C of triples (c, x, d) with $c, x, d \in V^*$ (*patterns*). We also say that cxd is a *site* and x is a *crossing*. Given two words $ucxdv, pexfq \in V^*$ and two patterns $(c, x, d), (e, x, f)$ both in B or in C, the splicing operation produces $ucxfq, pexdv$. The *splicing language* generated by S_H is $L = L(S_H)$, which is the smallest language obtained by using the splicing operation starting with I. In the following we also refer to $L = L(S_H)$ as an $H-$generated language.

`Paun's definition [25].` Given a finite alphabet V and two special symbols $|, \$$ not in V, a *splicing rule* (over V) is $r = u_1|u_2\$u_3|u_4$, $(u_i \in V^*, i = 1, 2, 3, 4)$. For such a rule r and $x, y, z \in V^*$, we write $(x, y) \vdash_r z$ if and only if $x = x_1u_1u_2x_2, y = y_1u_3u_4y_2$, $z = x_1u_1u_4y_2$, $(x_1, x_2, y_1, y_2 \in V^*)$. A *splicing system* $S_{PA} = (V, I, R)$ consists of a set R of rules and an initial language I. The *splicing language* $L = L(S_{PA})$ is obtained by iteration of the rules in R applied to I. We will say that L is PA-generated.

`Pixton's definition [39].` Given a finite alphabet V, a *splicing rule* is a triple $r = (\alpha, \alpha'; \beta)$ that applied to $w = \epsilon\alpha\eta$ and $w' = \epsilon'\alpha'\eta'$ produces $\epsilon\beta\eta'$. A *splicing system* is $S_{PI} = (V, I, R)$ where R is a finite collection of splicing rules and I is the initial set of strings. The *splicing language* $L = L(S_{PI})$ is the set of words obtained from I by repeated splicing using R. We will say that L is $PI-$generated.

Behind each definition, there is a biological motivation. Indeed, as splicing systems should capture the enzymes' behaviour on DNA, the difference between the three definitions depends on the phenomena that they want to model. Without going into details, we observe that the enzymes' action on DNA produces blunt ends and staggered cleavages and since in the recombinant processes of two segments the overhangs must match under some conditions, Head divides them into two classes (this explains why pair B, C appears in his definition). On the contrary, Paun and Pixton work more abstractly on single strands of DNA.

Circular case.

A circular word $^\sim w$ is nothing but the equivalence class of $w \in V^*$, V being a finite alphabet, with respect to the *conjugacy* relation $\sim$ defined by $xy \sim yx$, for $x, y \in V^*$ (see for example [31]). $V^\sim$ is the set of all such circular strings over V, i.e. the quotient of V^* with respect to $\sim$.

Given $L \in V^*$, $Cir(L)$ is the circularization of L, i.e. the set of all circular words corresponding to elements of L, while any language L such that $Cir(L) = C$, for a given circular language $C \subseteq V^\sim$, is called a linearization of C. The set of all strings in V^* corresponding to elements of C is the *full linearization* of C, written $Lin(C)$. Moreover, given a family of languages FA, $FA^{\sim}$ consists of all those circular languages C which have some linearization in FA. It has been proved that $L \in Reg^{\sim}$ if and only if its full linearization is regular [26].

`Head's definition [43].` Let $SC_H = (V, I, T, P)$ be a *splicing system*, where V is a finite alphabet, $I \subseteq V^{\sim}$ is the initial set of circular strings, $T \subseteq V^* \times V^* \times V^*$ and P is a binary relation on T, such that if $p, x, q, u, y, v \in V^*$, $(p, x, q), (u, y, v) \in T$ and $(u, x, v)P(u, y, v) \Rightarrow x = y$. Then, given SC_H and $^{\sim}hpxq, ^{\sim}wuxv \in V^{\sim}$ with $(p, x, q)P(u, x, v)$, the splicing produces $^{\sim}hpxvwuxq$. The *splicing language* $C = C(SC_H)$ generated by SC_H is obtained by iteration of the splicing applied to I. We will say that C is $CH-$generated.

`Paun's definition [25].` Given a splicing rule $r = u_1|u_2\$u_3|u_4$ applied to two circular strings $^{\sim}xu_1u_2, ^{\sim}yu_3u_4$, the rule cuts and linearizes the two strings obtaining u_2xu_1 and u_4yu_3, and pastes and circularizes them obtaining $^{\sim}u_2xu_1u_4yu_3$. A *splicing system* is $SC_{PA} = (V, I, R)$ where V is a finite alphabet, $I \subseteq V^{\sim}$ is the initial circular language, R is the set of rules. The *splicing language* $C = C(SC_{PA})$ generated by SC_{PA} is obtained by iteration of the splicing applied to the initial language using R. We will say that C is $CPA-$generated.

`Pixton's definition [39].` Given $r = (\alpha, \alpha'; \beta)$ and $\overline{r} = (\alpha', \alpha; \beta')$ applied to two circular strings $^{\sim}\alpha\epsilon, ^{\sim}\alpha'\epsilon'$, the splicing produces $^{\sim}\beta\epsilon'\beta'\epsilon$. A *splicing system* is $SC_{PI} = (V, I, R)$ where V is a finite alphabet, $I \subseteq V^{\sim}$ is the initial circular language, R is the set of rules. The *splicing language* $C = C(SC_{PI})$ generated by SC_{PI} is obtained by iteration of the splicing applied to the initial language using R. We will say that C is $CPI-$generated.

3.2 State of the Art

LINEAR CASE

The basic problem was the comparison with Chomsky hierarchy, namely with the classical models of computation [26]. This computational power obviously depends on I, R. Precisely, let us denote $H(F_1, F_2)$ the family of languages generated by S_{PA} with $I \in F_1, R \in F_2$, F_1, F_2 being classes of languages in the Chomsky hierarchy. In [25] there is a collection of known results in this sense (see also [33, 39]), some of which proved giving very useful and interesting properties of splicing languages [19, 40]. The following table collects the size of $H(F_1, F_2)$ (where I is the first column and R is the first row).

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 general result is that either $H(F_1, F_2)$ is a specific class of languages in the above hierarchy, or it is strictly intermediate between two of them. Let us see only some examples of $H(F_1, F_2)$ for specific values of F_1, F_2. For example, starting with a finite language, using $R \in Reg$ is possible to obtain some languages in RE, but not all; moreover, $Fin \subseteq H(Fin, Fin) \subseteq Reg$, Fin (resp. Reg, RE) being the class of finite (resp. regular, recursive enumerable) languages. There are regular languages that cannot be generated by splicing, and so a natural problem is the following.

Problem 3.1 *Characterize regular languages which belong to* $H(Fin, Fin)$*.*

Trying to give an answer to this problem, in [5] is shown how to obtain Reg using iterated splicing with $I \in Fin, R \in Fin$, but with "marked rules". In the same paper, it is given a characterization of the strictly locally testable (SLT) languages through simple properties of the automaton, proving that this class is generated by particular splicing systems (using Paun definition).

A different characterization of SLT languages was already presented. Indeed, in [22] the splicing operation is explicitly linked with the concept of *constant* introduced in [42]. In the same paper, Head proved that a significant subclass of splicing languages (persistent splicing languages) coincide with SLT languages and may be generated by $S = (V, I, V^P, V^P)$, for $P \geq 1$ (uniform splicing systems). To do this, he used a result in [16]. It is worth observing that $H(Fin, Fin) \not\subseteq LT$ and $LT \not\subseteq H(Fin, Fin)$ where LT denotes the class of locally testable languages, i.e. $H(Fin, Fin)$ is not comparable (with respect to the inclusion) with LT (see [7]).

It is natural to compare the computational power of the three types of splicing operation, which is the same as solving the problem below.

Problem 3.2 *Given* $L = L(S_X)$*, with* $X \in \{H, PA, PI\}$*, does* S_Y *exist, with* $Y \neq X, Y \in \{H, PA, PI\}$*, such that* $L = L(S_Y)$*?*

It is quite evident that $S_H \leq S_{PA} \leq S_{PI}$, i.e. a language generated with Head splicing rules may be rewritten using Paun splicing rules and the latter may be rewritten with Pixton splicing rules. The question is whether these inclusions are strict or not.

In the previous section, I have presented a particular type of splicing system. Let me sketch that there are other kinds of splicing systems as, for example, *extended splicing systems* [25, 34, 36]. These ones, using a finite initial set of words and a finite (resp. regular) set of rules, generate all Reg (resp. RE, so they have the same computational power of Turing Machines). A lot of variants of splicing were considered in [35], especially with the aim of building a system with the same power of a Turing Machine and as simple as possible.

Other models introduce multeplicity on the initial set. In this case, the membership problem suggested by Head [22] (given a DNA string and a splicing system, it is decidable whether the string is spliced by the system?) has been proved to be recursive unsolvable [17], with $I \in Reg, R \in Fin$.

Obviously, questions of finiteness, membership and equivalence are recursive when, without the multeplicity hypothesis, we consider the result $H(Reg, Fin) = Reg$. The above equality has been obtained after several intermediate results, presented in different papers. For instance, the membership problem was shown to be recursive firstly in [18], for a particular class of splicing systems S_H and with $I \in Reg, R \in Fin$, because the author shows that these particular splicing systems generate a subclass of regular languages. Moreover, in the same paper a universal algorithm is given, which produces a finite state automaton that accepts the splicing language, starting from an algorithmic reduction of the set of patterns. After, the regularity of $L(S_H)$ is shown in [29], by giving an algorithm which constructs a finite state automaton accepting the splicing language spliced by the system. Moreover, in the same paper a characterization of SLT languages is given, based on automata properties. The above equality was already given in [9], using dominoes languages to model double stranded DNA. In [29] we find contructive proofs for the regularity of languages considered in [9] and for the strict locality of the languages considered in [22]. Pixton gave a simpler proof of this regularity, with respect to his definition, building a graph capturing the splicing behaviour [39]. Head has given a constructive proof for SLT generated by splicing [24] and decidable properties for regular languages generated by particular splicing systems, using the concept of constant [23].

Circular case.

In [39, 43] there was noted the difference between linear and circular splicing, in other words, some regular languages not generated by finite linear splicing are generated by finite circular splicing, when they are taken in their circular form.

In the papers which have already been published on this topic, additional hypotheses are often considered for Paun's or Pixton's definitions [25, 39]. Precisely:

Hypothesis 1. R is a symmetric scheme, i.e. for any rule $r = u_1|u_2\$u_3|u_4$ (resp. $r = (\alpha, \alpha'; \beta)$) in a splicing system SC_{PA} (resp. SC_{PI}) there is the rule $\overline{r} = u_3|u_4\$u_1|u_2$ (resp. $\overline{r} = (\alpha', \alpha; \beta')$).

Hypothesis 2. R is a reflexive scheme, i.e. for any rule $u_1|u_2\$u_3|u_4$ (resp. $(\alpha, \alpha'; \beta)$) in a splicing system SC_{PA} (resp. SC_{PI}) there is the rule $u_1|u_2\$u_1|u_2$ (resp. $(\alpha, \alpha; \alpha)$).

Hypothesis 3. Self-splicing. Self-splicing is defined in a splicing system SC_{PA} (resp. SC_{PI}) producing $\sim u_4 x u_1$ and $\sim u_2 y u_3$ from $\sim x u_1 u_2 y u_3 u_4$ 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 $r, \overline{r}$ as above).

We note that any set R of rules in a splicing system SC_{PI} is implicitly supposed to be symmetric and that the original definition of circular splicing language given by Paun in [25] includes self-splicing to define a splicing language generated by a SC_{PA} system.

As we have already said, the investigation of circular splicing systems is not so developed as in the linear case. In particular, the analysis of their generative capacity remains an open problem even in a more restricted situation, i.e., with a finite set of rules. In the sequel, we suppose R finite.

It is already known that, in contrast with the linear case, $C(Fin, Fin)$ is neither intermediate between two classes of languages (in the Chomsky hierarchy) nor a specific class. Indeed, $\sim a^n b^n$ is a circular context-free language which is not a circular regular language (since it has no regular linearization), but $\sim a^n b^n$ is $CH-$generated with finite initial circular language [43]. On the other hand, there are regular circular languages that cannot be generated by using CH or CPA splicing systems (with finite initial circular language), e.g. $\{\sim a^n b | n \geq 1\}$, $\{\sim (aa)^n b | n \geq 0\}$ (see [43, 7]).

There are few results on the computational power of circular splicing. One concerns Paun's systems with all hypotheses and states that $C(F_1, Fin) \subseteq F_1$, for F_1 circular full AFL [25]. Moreover Pixton proved that, with hypotheses 1 and 2, $C(F_1, Reg) \subseteq F_1$, for F_1 circular full AFL [39, 40]. Another result is given in [43], where the authors show that using particular Head's rules, applied to a finite initial circular language, the generated splicing language is regular.

Finally, in a natural way, we can reformulate Problem 3.2 for the circular case.

3.3 Contributions

Linear case

I am interested in studing iterated splicing without multeplicity and this is why I have limited myself to tell this topic. Moreover, inside this area, until now, I have worked on the comparison between splicing languages and regular languages (or subclasses of regular languages [32]).

- Many papers concern the problem of passing from splicing systems to automata, also for proving that some questions are decidable (see Section 3.2). Instead, in [6] we have solved Problem 3.1 capturing the behaviour of $H(Fin, Fin)$ and providing the first characterization (effective procedure) of the class of languages generated by these systems in terms of properties of finite state automata. Moreover, we give an algorithm to construct the splicing system associated with the minimum finite state automaton $\mathcal{A}$ for the given regular (splicing) language. Since our aim is building a splicing system with finite rules and a finite initial language, we need to consider a number of conditions (on the state graph $G_{\mathcal{A}}$, associated to $\mathcal{A}$) which is finite, but sufficient to generate all the sequences accepted by $\mathcal{A}$. In order to do so, three properties that may be tested on $G_{\mathcal{A}}$ are given. These properties are necessary and sufficient for $G_{\mathcal{A}}$ to be associated with a regular language generated by splicing. Finally, an algorithm is given which starts from a finite initial set of "canonical" crossings of successful paths and produces the set of rules in order to generate $L(\mathcal{A})$. The first part of the paper has been a real crux of the matter. We propose a method that may be used for dissolving the cycles in a maximal strongly connected component of an automaton, in a canonical way.

- With respect to the problem 3.2, in [6] we have given the first example of regular language PA-generated, but not H-generated. This example has been given using strongly the characterization given above.

Circular case

- We have said that circular splicing implies investigations about the nature of circular languages. It is worth noting that formal languages closed under conjugacy which are also free monoids were already studied [3, 4, 31]. Indeed a characterization of the corresponding generated codes exists [3]. This characterization is strongly used in [8], for showing that the corresponding circularization of these languages is generated by splicing.

- The circular version of the Problem 3.2 has just been faced in [7], where we have given a regular circular language CPI-generated, but neither CH-generated nor CPA-generated. So the three definitions have different computational power.

- Let us denote $C(F_1, F_2)$ the family of languages generated by circular splicing using one of the three definitions given in Section 3.1, with $I \in F_1^{\sim}$ and $R \in F_2$, for F_1, F_2 classes of languages in the Chomsky hierarchy. Following general strategies, we have proposed the problem below.

Problem 3.3 *Can we characterize $F \cap C(Fin, Fin)$ (resp. $F \cap C(Reg, Fin)$) for each class F of languages in the Chomsky hierarchy, for every definition of $C(Fin, Fin)$ and for every possible combination of the three hypotheses above?*

Up to now, we have considered only circular splicing behaviour without hypotheses. Beside combinatorial properties for splicing languages and examples of circular splicing languages [7], we have obtained promising results towards the above problem, using links between syntactic monoid, conjugacy and codes (see [3, 4, 32]). It is known that if X is a code, X^* is closed under conjugacy if and only if the syntactic monoid of X^* is a group. Moreover, if X is finite then $X = A^d$, for d positive integer [3] and if X is rational, it may be proved that the syntactic monoid of X^* is a group if and only if X is a group code. Using these results, we have proved that $X^* \in C(Fin, Fin)$, for X rational group code [8]. We have found rational languages L, which are closed under conjugacy and generated by circular splicing, with $L \neq X^*$, for X code.

- I will review in the next section problems and future developments proposed in [8].

3.4 Future works

LINEAR CASE.

Following some partial result obtained in [24], we are concerned to give a syntactic characterization of $H(Fin, Fin)$. Moreover, it could be possible to try to characterize $H(Fin, Fin)$ by using closure properties of regular languages or subclasses of regular languages.

Since automata, grammars and splicing systems concepts are equivalent for languages in $Reg \cap H(Fin, Fin)$, we can investigate for a " minimal splicing system" with analogy to the notion of "minimal state automata" [26].

With respect to Problem 3.2, until now, we have not discovered regular languages PI-generated but not PA-generated. For solving this problem, it seems necessary to give an automata characterization of the class of languages $L(S_{PI})$, as done for Paun's definition in [6].

It may be nice to examine relations just found between splicing languages and picture languages (for this class of languages, see [20]).

CIRCULAR CASE.

The basic proposal is to answer to Problem 3.3. In [7] we have proposed a conjecture about the circular regular languages generated by finite splicing, but in [8] we have showed that this conjecture is false. Moreover, in [7] there is a conjecture about the same problem for context free languages. We have not yet discovered a context-sensitive or recursive enumerable language, which is not context-free, in $C(Fin, Fin)$. We think that the power of $C(Fin, Fin)$ is not more than NP.

To approach this problem, we can follow two ideas. Firstly, as we are doing, we can study circular languages as formal languages closed under conjugacy. The aim is to individuate rational languages closed under conjugacy and generated by splicing. Interesting links with the theory of codes have been used in this approach.

It could be important, in order to define this class of languages, to give closure properties; for example, as we have pointed out in [8], rational languages closed under conjugacy are closed under union and intersection. It is unknown whether this closure remains considering also the hypothesis of being in $C(Fin, Fin)$.

For the Problem 3.2 we must deeply investigate Pixton systems, that model a different behaviour of circular splicing, as we can see. We do not know whether Head's or Paun's systems are equivalent or not.

It is necessary to understand the role of the hypotheses introduced by the literature, which can increase the complexity our semplified model.

A second approach may be used to redefine splicing taking into account the redoundance. In other words, the splicing definition allows the generation of multiple copies of a circular word, i.e. we can produce different words whose circularization is the same circular word, having no control on this step. It should be interesting to give a new splicing definition such that splicing does not duplicate elements (circular words) previously obtained (like a multeplicity on the output of splicing operation). In this way, splicing produces only one element for each conjugacy class. This may be done by choosing a suitable element of the class. Then, given two classes, it is possible to define the concatenation of two circular words through these elements and so the splicing becomes a particular concatenation (or a particular "circular shuffle", constrained by a rule). Indeed it is evident that, while the linear splicing lose "pieces" of the input strings, the circular splicing does not. An effective strategy could be to apply the theory of Lyndon Words [31].

Appendix

A A flash of biochemical background

- **DNA (deoxyribonucleic acid).** It can abstractly be viewed as a string over a four-letter alphabet (A, C, G, T). It is a large molecule (polymer) composed by repeated four smaller units (monomers), known as nucleotides, consisting of a nitrogenous base (Adenine, Guanine, Cytosine, Thymine), a phosphate molecule and a sugar molecule, that link together in a linear structure forming DNA's bachbone. This has two distint ends, giving a polarity. DNA in a cell is normally double stranded, i.e. it forms a double helix of two intertwined strands, binded by hydrogen bounds (A-T and G-C are Watson-Crick complementary).

- **Polymerase Chain Reaction (PCR).** It is a reaction in which double-stranded sequences of DNA (template) are replicated, strarting from a short sequence that marks the piece to be replicated (primer). It is often used to amplify DNA sequences. Because the copies of templates can function as templates themselves, repeated PCR can be used to exponentially multiply the template. It is realized by thermic cycles.

- **Hybridization.** DNA sequences may be joined when they begin or end in short overhanging complementary single stranded DNA sequences. The forming of hydrogen bonds between such sequences is known as hybridization. It may happen even if the bases are not all perfectly complementary.

- **Ligase.** Ligase enzymes are proteins that paste together pieces of DNA whose sticky ends coincide.

- **Restriction.** Certain classes of enzymes (endonucleases) are capable of cutting DNA in a specific base sequence (site) producing strands with scattered or sticky ends. These enzymes, combined with the previous ones are the inspiration for the splicing operator.

- **Plasmides.** They are circular strand of DNA that reply themselves indipendently by bacterial genome. They are often used as vectors (particular DNA molecules) for replication. We observe that the replication happens inside the host cell only when DNA has a circular form.

B Abstract of contributions

- C. De Felice, R.Zizza, *Factorizing Codes and Krasner Factorizations*, Proc. of ICTCS98, (1998), pp. 347 – 358.
  Abstract: Motivated by Schützenberger's factorization conjecture, we investigate finite languages $P, S \subseteq A^*$ which can appear in a factorization $A^* = SC^*P$ of the free monoid, with the product being unambiguous and C being a factorizing code. In particular, we characterize all such P for $|S| = 3$ and all such S with $S \subseteq w^*, w \in A^*$. We propose conjectures on the structure of the left factors S in a factorization as above. The results which we obtain show a relationship with the Krasner factorizations.

- P. Bonizzoni, R. Zizza, *Deciding whether a regular language is a splicing language*, Tech. Rep., (1999), 17 pages.
  Abstract: The splicing operation has been introduced in [22] as a language generative mechanism to model the recombinant behaviour of double stranded DNA molecules under the influence of restriction enzymes and ligases. One of the main problems raised in [22] concerns the structure of the language which could be produced by a *splicing system* consisting of a finite set of initial DNA and a finite set of rules (enzymes). A partial answer to this question was given in [39] by showing that such splicing systems produce a restricted subclass of regular languages. Here we give the first characterization of the languages generated by these splicing systems answering the original question in [22] by giving the properties of finite state automata recognizing such languages. An effective procedure for deciding whether a regular language is a splicing language and an algorithm which produces the splicing system that generates it are given.

- P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza *Linear and circular splicing*, WORDS99, (1999), 13 pages.
  Abstract: In this paper we survey the general ideas underlying splicing, focusing on the three different definitions for both linear and circular case. We survey known results, restricting our investigation to regular splicing languages. We announce the result of [6] for linear case. We also give a survey on the state of the art for the circular case, showing some combinatorial results and examples of circular splicing languages. Moreover some conjectures for regular and context free circular languages are proposed.

- P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza *On the circular splicing*, in preparation.
  Abstract: In this paper we study circular splicing, carrying on the investigation of [7]. We review some partial results on splicing and conjugacy relation, giving a first class of rational languages generated by circular splicing. Some open questions and starting points for future works are presented, as well.

References

[1] L. M. Adleman, *Molecular computation of solutions to combinatorial problems*, Science, No. 226, 1994, pp. 1021 – 1024.

[2] E.B. Baum, *DNA sequences useful for computation*, Proc. of 2nd Annual Meeting on DNA Based Computers (DIMACS Workshop), (1996), pp. 122 – 126.

[3] J. Berstel, D. Perrin, *Theory of Codes*, Academic Press, (1985).

[4] J. Berstel, A. Restivo, *Codes et sousmonoides fermes par conjugaison*, Rapport LITP 81 – 45, (1981), 11 pages.

[5] P. Bonizzoni, C. Ferretti, G. Mauri, *Splicing systems with marked rules*, Romanian J. of Information Science and Technology, Vol. 1, No. 4, (1998), pp. 295 – 306.

[6] P. Bonizzoni, R. Zizza, *Deciding whether a regular language is a splicing language*, Tech. Rep., (1999), 17 pages.

[7] P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza, *On the linear and circular splicing*, WORDS99, (1999), 12 pages.

[8] P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza, *On the circular splicing*, in preparation.

[9] K. Culik II, T. Harju, *Splicing semigroups of dominoes and DNA*, Discr. Applied Math., 31, (1991), pp. 261 – 277.

[10] J.H.M. Dassen, *A bibliography of molecular computation and splicing systems*, HTML: http://www.wi.LeidenUniv.nl/~jdassen/dna.bib

[11] J.H.M. Dassen, *Molecular Computation and Splicing Systems*, MSc. Thesis, Leiden Univ. (1996).

[12] R. Deaton, R.C. Murphy, M. Garzon, D.R. Franceschetti, S.E. Stevens, *Good encoding for DNA-based solutions to combinatorial problems*, Proc. of 2nd Annual Meeting on DNA Based Computers (DIMACS Workshop), (1996), pp. 131 – 140.

[13] R. Deaton, R.C. Murphy, M. Garzon, D.R. Franceschetti, S.E. Stevens, *On the encoding problem for DNA Computing*, Proc. of 3rd Annual Meeting on DNA Based Computers (DIMACS Workshop), (1997), pp. 230 – 237.

[14] R. Deaton, R.C. Murphy, M. Garzon, D.R. Franceschetti, J.A. Rose, S.E. Stevens, *The effect of uniform melting temperatures on the efficiency of DNA Computing*, Proc. of 3rd Annual Meeting on DNA Based Computers (DIMACS Workshop), (1997), pp. 35 – 42.

[15] C. De Felice, R. Zizza, *Factorizing Codes and Krasner Factorizations*, Proc. of ICTCS98, (1998), pp.

[16] A. De Luca, A. Restivo, *A characterization of strictly locally testable languages and its application to subsemigroups*, Information and Control, 44 (1980), pp. 300 – 319.

[17] K.L. Denninghoff, R.W. Gatterdam, *On the undecidability of splicing systems*, Intern. J. Computer Math., Vol. 27, (1989), pp. 133 – 145.

[18] R.W. Gatterdam, *Algorithms for splicing systems*, SIAM Journal of Computing, 21, (1992), pp. 507 – 520.

[19] R. W. Gatterdam, *Splicing systems and regularity*, Intern. J. Computer Math., Vol. 31, (1989), pp. 63 – 67.

[20] D. Giammarresi, A. Restivo, *Two-dimensional Languages*, in: Handbook of Formal Languages, G. Rozenberg & A. Salomaa, Eds., Springer Verlag, Vol. 3, (1996), pp. 215 – 267.

[21] J. Hartmanis, *On the Weight of Computations*, Bull. of EATCS, No. 55, (1995), pp. 136 – 138.

[22] T. Head, *Formal Language Theory and DNA: an analysis of the generative capacity of specific recombinant behaviors*, Bull. Math. Biol., Vol. 49, No. 5, (1987), pp. 737 – 759.

[23] T. Head, *Splicing languages generated with one sided context*, Discrete Applied Math., Vol. 84, (1998), pp. 145 – 163.

[24] T. Head, *Splicing representations of strictly locally testable languages*, Discrete Applied Math., Vol. 82, (1998), pp. 139 – 147.

[25] T. Head, Gh. Paun, D. Pixton, *Language theory and molecular genetics: generative mechanisms suggested by DNA recombination*, in: Handbook of Formal Languages, G. Rozenberg & A. Salomaa, Eds., Springer Verlag, Vol. 2, (1996), pp. 295 – 360.

[26] J. Hopcroft, J. Ullmann, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, (1979).

[27] N. Jonoska, S. Karl, *A molecular computation of the road coloring problem*, Proc. of 2nd Annual Meeting on DNA Based Computers (DIMACS Workshop), (1996), pp. 148 – 158.

[28] P.D. Kaplan, G. Cecchi, A. Libchaber, *DNA based molecular computation: template-template interactions in PCR*, Proc. of 2nd Annual Meeting on DNA Based Computers (DIMACS Workshop), (1996), pp. 159 – 171.

[29] S.M. Kim, *Computational Modeling for genetic splicing systems*, SIAM Journal of Computing, 26, (1997), pp. 1284 – 1309.

[30] D. Lind, B. Marcus, *An Introduction to Symbolic Dynamics and Coding*, Cambridge Univ. Press, (1995).

[31] M. Lothaire, *Combinatorics on Words*, Encyclopedia of Math. and its Appl., Addison Wesley Publishing Company (1983).

[32] R. McNaughton, S. Papert, *Counter-Free Automata*, MIT Press, Cambridge, Mass., (1971).

[33] A. Mateescu, Gh. Paun, G. Rozenberg, A. Salomaa, *Simple splicing systems*, Discr. Appl. Math., 84, (1998), pp. 145 – 163.

[34] Gh. Paun, A. Salomaa, *DNA computing based on the splicing operation*, Math. Japonica, Vol. 43, No. 3, (1996), pp. 607 – 632

[35] Gh. Paun, G. Rozenberg, A. Salomaa, *Computing by splicing*, Theoretical Computer Science, 168, Vol. 2, (1996), pp. 321 – 336.

[36] Gh. Paun, G. Rozenberg, A. Salomaa, *Dna Computing: new computing paradigms*, Springer-Verlag, (1998).

[37] D. Perrin, *Symbolics Dynamics and Finite Automata*, Proc. of MFCS95, LNCS 969, (1995), pp.

[38] Dominique Perrin, M. P. Schützenberger, *Synchronizing prefix codes and automata and the road coloring problem.* In P. Walters editor, Symbolic Dynamics and its Applications, Vol. 135 of Contemporary Mathematics, Amer. Math. Soc., (1992), pp. 295 – 318.

[39] D. Pixton, *Regularity of splicing languages*, Discrete Applied Mathematics, 69, (1996), pp. 101 – 124.

[40] D. Pixton, *Linear and circular splicing*, Proc. of 1st Int. Symp. on Int. in Neural and Biological Systems, (1996), pp. 181 – 188.

[41] C. Reutenauer, *Non Communtative Factorization of variable length codes*, J. Pure and Applied Algebra, 36, (1985), pp. 157 – 186.

[42] M.-P. Schützenberger, *Sur certains operations de fermeture dans les langages rationnels*, Symposia Math., 15 (1975), pp. 245 – 253.

[43] R. Siromoney, K.G. Subramanian, V. Dare, *Circular DNA and Splicing Systems*, Proc. of ICPIA, LNCS 654, Springer-Verlag, (1992), pp.260 – 273.