Università degli Studi di Milano
Dipartimento Scienze dell'Informazione
Dottorato di Ricerca in Informatica (XIII ciclo)
a.a. 2000-2001

PhD Thesis
On the power of classes of splicing systems

Advisors

Prof. Giancarlo Mauri

Prof.ssa Clelia De Felice (Univ. di Salerno)

PhD Student

Rosalba Zizza

Coordinatore:

Prof. Gianni Degli Antoni

Contents

Abstract 4

Foreword 5

Introduction 7

I Linear Splicing 17

1 Definitions on languages, splicing systems and main results 20

1.1 Basics on languages 20

1.2 Linear splicing operation 22

1.3 H-systems: state of the art 26

1.4 Other models of linear splicing systems 33

1.5 Linear results: overview 35

2 A comparison of the three variants of finite linear splicing systems 36

2.1 Deeply remarks on linear splicing definitions 37

2.2 Inclusion results 38

3 Linear splicing via syntactic monoid 43

3.1 Preliminary definitions and properties of splicing rules 44

3.2 Other definitions on automata and representatives of classes . 47

3.3 Regular splicing languages 50

II Circular splicing **56**

4 Definitions on circular languages **59**

5 Circular splicing history: models and main results **62**
5.1 Circular splicing operation 62
5.2 Circular splicing hystory: other models 65
5.3 Circular results: overview 68

6 Computational power of the three finite splicing systems **70**
6.1 Circular regular splicing languages and inclusion results . . . 70
6.2 A circular regular language not generated 76

7 Circular regular Paun generated languages: star languages **81**
7.1 New definitions 81
7.2 Fingerprint closed star languages are Paun generated 85
7.3 Circular splicing and codes 91

8 The case of one-letter alphabet and uniform languages **96**
8.1 The characterization 97
8.2 The case of regular languages on one-letter alphabet is decidable 102
8.3 Uniform languages 111

9 Circular regular Paun generated languages: cyclic and weak cyclic languages **116**
9.1 Cyclic languages: definitions and results 117
9.2 Weak cyclic languages 126
9.3 From Paun splicing to Pixton splicing 128

10 Finite Pixton splicing systems **131**
10.1 A finite circular Pixton splicing system 133
10.2 Pixton splicing and codes 141
10.3 Examples 144

Conclusions **149**

Bibliography **150**

Appendix **158**

Splicing in cartoons **161**

Abstract

"It is hoped that the line of research initiated here will interest biologists who have not previously found reason to be concerned with formal language theory... The results discussed here are certainly only modest formal achievements. It is hoped that future developments will provide much stronger results following the given paradigm and yield wholly new paradigms as well (T. Head, 1987)"

My thesis provides an overview and investigations on Molecular Computing and Splicing Systems. Molecular Computing is a novel branch connecting Computer Science, Mathematics and Biology: computation is carried on by DNA molecules and the operations are performed by using enzymes and other biological processes (the "manifesto" is [1]). Splicing Systems are formal models of a recombinant generative process among molecules which are cut and pasted by a simultaneous action of restriction and ligase enzymes.

Splicing operation is introduced as an operator on strings and systems generating languages are defined, whose generating power is compared with classical results in Formal Languages (the initiator of Splicing Systems Theory is Tom Head [34]). My thesis concerns particular (finite) splicing systems for linear and circular splicing in a perspective which has never been explored so far. Decidability and descriptional complexity questions are answered, various results are provided by using combinatorics on words, classical issues in Formal Languages and novel ideas on automata. Deep connections with Theory of (variable length) Codes have been discovered. Problems and conjectures for future work are also proposed. A flash of biochemical background, thought out of our scope, is provided together with an extensive bibliography.

Foreword

"O tutto o nulla io voglio:
il sole d'oro o l'uragano nero...
e, dallo sguardo tuo, martirio o incanto"
(Antonio Nardacci)

To my Angels...
in Heaven and in Earth

I thank God and all those helped and stimulated me, during the PhD time. I know that no words are enough to express what I feel.

I am sincerely indebted with Clelia De Felice. She introduced me to Theory of Codes, teaching the passion for research, the sense of duty and honesty. During these years, she has continued on attending my studies, as critical proofreader, taking care of me much more than is customary for her role. I thank Giancarlo Mauri for introducing me to Molecular Computing and EMCC group, for his paternal guide, especially during the first times. I have been, and I am, glad to work with Paola Bonizzoni. She has made me welcome in Milano, encouraging and stimulating me during these years, sharing uncertainty and joys.

My work has been supported by DSI of the University of Milano, Disco of the University of Milano-Bicocca and DIA of the University of Salerno, which have given me the technical means during the research for and the writing of this thesis.

There is someone who has made my PhD possible. Dear Marilu', you have opened your home, like a sister, and only thanks to you Milano has been another home for me. I bless God for having met you, for your kindness, your friendship, your nice family, for Paola, who made us met.

Good luck to Brunella, Claudio and Marco, my collegues in PhD course; I thank my catechists and my community, who have prayed for me and have showed me what is important; my sincere friends, a God's present who have encouraged, making me laugh and think, without words... I thank my "big brothers - body guards", theirs wives, sons, my sweet treasues.

I thank Mauro, my blue prince, for his patience, his optimism, his love, for having listened my feelings, sharing these last years with me, together with neverending days in the university, anxiety, joys, successes and failures, for helping me to go on, always thoughtful with me.

Finally I dedicate this work to my parents. They have been patient and braves starting with me the PhD adventure and accepting, with me and more than me, the uncertainty of these years, being happy seeing me happy, worried seeing me tired, but they are always two little angels for me, everywhere I were.

Introduction

Automata Theory has always concerned a deep variety of areas, revealing connections with different topics, such as semigroups algebraic theory, algorithms structure, pattern manipulation. It is perhaps this diversity which has enriched the field to make it one with both interesting applications and significant mathematical problems.

This thesis provides results that can be analyzed from different points of view, since common techniques in Formal Languages together with Combinatorics on Words and properties of Finite Monoids are used.

The first historical reference in Automata Theory is a paper of S. Kleene (1956), in which the equivalence between the notion of regular event-language (recognized or accepted by a finite state automaton) and rational language (a language that belongs to the closure of finite languages with respect to rational operations) was given. Since then, the theory of Finite Automata has developed in order to investigate both languages properties (operations among languages, closure properties) and their interpretations on automata (see [61] for an introduction).

Very recently, Formal Languages Theory has been positively influenced by the introduction of new models of computation, which is contributing to its renewal. In particular, a new mathematical formalism for biological phenomena of recombinant processes has been proposed [34]. Indeed a DNA strand can be viewed as a string over a four letter alphabet and so a natural way to model DNA computation is within the framework of Formal Language Theory.

The DNA (deoxyribonucleic acid) is the storage medium for genetic in-

formations. It is composed by units, called *nucleotides*, distinguished by the chemical group (base) attached to them. The four bases are Adenine, Guanine, Cytosine, and Thymine (denoted respectively A, G, C, T). Each (single stranded) nucleotide is linked end-to-end to form DNA strands, called *oligonucleotides* if there are less than 30 nucleotides. The DNA sequence has a *polarity*, i.e., a sequence of DNA is distinct from its reverse. Thus the two distinct ends of a DNA sequence are known as $5'$ end and the $3'$ end. The four bases are related each other by Watson-Crick complementary rule (A=$\bar{T}$, G=$\bar{C}$) and two complementary (single) strands with opposite polarity will join together, forming a double helix (hybridation process).

Even if the possibility of using DNA and biological tools in order to formulate and solve computer science problems may surprise, the idea behind Molecular Computing is the observation that biological processes and mathematical processes are analogous. In fact, the complex structure of living cells is the result of applying *simple* opeations (recognition, copying, duplication...) to initial informations encoded in a DNA sequence. Moreover, each result of application of a computable function to a given argument can be obtained as the application of combined *simple* functions to the given argument [44]. This similarity is the key of Molecular Computing. Indeed, 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 Biology to approach computer science's problems, underlying idea of *Molecular Computing* (for a quite exhaustive bibliography, see [27]).

The rapidly increasing of interest in Molecular Computing started with Adleman's article [1], where the author showed how to solve (a small instance of) the Hamiltonian Path Problem, that is a NP-complete problem, by manipulating DNA sequences in a lab. A series of papers were soon written, generalizing his approach (for example, see [42] or DNAx Proceendings, $x \in \{1, \ldots, 7\}$). This is commonly called 'wet' molecular computing, to make evident how the computer scientists enter in the laboratory to make algorithms in test tubes.

Let me say that Wet Molecular Computing may enlighten unexpected

links with the classical Theory of Codes [3]. Indeed, beyond the real value of Adleman's experiment, his paper opened new interactions among different areas. As Hartmanis observed in [32], 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 greater input 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 lab-simulation, trying to avoid PCR, the chief agent of time and weight increasing and hybridization. In fact, PCR (see Appendix) may cause errors in computation, i.e., in a test tube wrong molecules are produced with respect to the problem encoding. 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. Giving constraints on the DNA sequences encoding the input, different techniques to avoid errors caused by unwanted hybridizations, a new distance more suitable than Hamming distance allowing the development of encodings to prevent errors are presented in [2, 20, 21, 22, 43] (some suggestions of research have been mentioned in [75]). To avoid unwanted hybridization, another approach, using (variable-length) codes, is initiated in [41], studying deeply the Watson-Crick complementary law. Also circular DNA may be useful. For example, the plasmids (see Appendix) have circular form and autonomously replicate without errors and exponential weight increasing. In fact, a new method of computing, applicable to a wide variety of algorithmic problems, has been very recently introduced by using DNA plasmids: a circular DNA plasmid is specifically constructed step by step, using a scheme of enzymatic treatments [37, 39]. Previously, a circular DNA restriction enzyme is used to encode the transition table of a Turing Machine, through insertion and deletion of fragments into the plasmid [73].

The *molecular algorithm* presented by Adleman has been generalized to other combinatorial problems. An interesting example is the molecular algorithm proposed in [42] for the Road Coloring Problem (RCP), a well-known problem on graphs and up to now, it is unknown whether RCP is decidible

(see, for example [60]). A partial answer is given in [62], where the problem has been positively solved with particular additional hypotheses, by using a formulation in Symbolic Dinamics of a theorem on the non commutative factorization of polynomials [68]. This contribution is known 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 [42], 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.

Thanks to this renewed interest in the Molecular Computing, new models of computation have been suggested and old ones have been rediscovered. Nevertheless, Formal Language Theory already studied a new manner of relating with the study of informational macromolecules. Indeed, in 1987 Tom Head revealed his idea in the paper: *Formal Language Theory and DNA: an analysis of the generative capacity of specific recombinant behaviours* [34]. In this work, he introduced the formalism of *splicing system* to observe the generative power of some recombination processes. So, abstract models for the generated languages are defined as a formal counterpart of the DNA recombination under the reaction of restriction and ligase enzymes. Nowadays, this is still one of the growing areas in Molecular Computing, as well 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. It should be said that splicing systems are only one of systems proposed from DNA to Formal Languages. Indeed in [59] other systems are presented, some of which oriented to double stranded DNA model, such us Sticker Systems, Watson-Crick Automata, Insertion and Deletion Systems ([45, 59]).

Since a DNA strand can be likened to a string consisting of a combination of the four bases A, C, G, T, this means that we have a four letter alphabet $\{A, C, G, T\}$ (the four deoxyribonucleotides) to encode information and so biological operations on DNA strands can be seen as operations on strings. Each DNA molecule is double stranded, but we can treat each such molecules

as two strings bounded together in parallel (i.e., our alphabet consists of four symbols $[A/T], [C/G], [G/C], [T/A]$). A nice description of biochemical introduction is given in [64], where a single molecule of double stranded DNA is visualized as a long "ladder", depicted horizontally below, whose steps are the vertical pairs $[A/T], [C/G], [G/C], [T/A]$.

As DNA occurs in both linear and circular form, two kinds of splicing operation were defined to model different biological phenomena, but since the underlying idea is the same, we only summed up the biological interpretation of linear splicing following [34, 44, 38] in this description, without facing here the difficulty of circular representation.

Consider the following two double-stranded DNA molecules:

$$\begin{array}{ll} 5'CCCCCTCGACCCCC3' & 5'AAAAAGCGCAAAAA3' \\ 3'GGGGGAGCTGGGGG5' & 3'TTTTTCGCGTTTTT5' \end{array}$$

Splicing occurs in two steps. First, restriction enzymes (proteins) recognize a specific pattern in a DNA sequence and then they cut the molecule inside the pattern, in a specific point inside the pattern. The site and the shape of the cut ends are specific to each enzyme. Consider the two restriction enzymes *Taq*I and *Sci*NI, whose recognition sites are, respectively:

$$\begin{array}{ll} 5'TCGA3' & 5'GCGC3' \\ 3'AGCT5' & 3'CGCG5' \end{array}$$

Since DNA is double stranded, the cutting phase may produce *blunt* or *staggered* ends. This is caused by cleavege patterns which are blunt or not. For example, *Taq*I and *Sci*NI have the following cleavege patterns:

$$\begin{array}{ll} 5'T \quad CGA3' & 5'G \quad CGC3' \\ 3'AGC \quad T5' & 3'CGC \quad G5' \end{array}$$

Other enzymes produce blunt ends: for example, we report below the pattern recognized by *Alu*I and its cleveage pattern

$$\begin{array}{ll} 5'AGCT3' & 5'AG \quad CT3' \\ 3'TCGA5' & 3'TC \quad GA5' \end{array}$$

The effect of TaqI's action on the first molecule and of SciNI's action on the second molecule is to produce the four fragments, with dangling single strands (in [64] and others, the remaining unsplit parts are called the left and the right contexts of the site):

$$5'CCCCCT \quad CGACCCCC3' \qquad 5'AAAAAG \quad CGCAAAAA3'$$
$$3'GGGGGAGC \quad TGGGGG5' \qquad 3'TTTTTCGC \quad GTTTTT5'$$

It is quite evident that no fragment previously cutted by AluI may be pasted with any of the four fragments written above (no sticky end match). Clearly, if such staggered ends are in the neighborhood of each other, the original two molecules may be reassociated, but this is not very interesting. Anyway, it also possible that two new hybrid molecules will be formed as below:

$$5'CCCCCTCGCAAAAA3' \qquad 5'AAAAAGCGACCCCC3'$$
$$3'GGGGGAGCGTTTTT5' \qquad 3'TTTTTCGCTGGGGG5'$$

Ligase enzymes perform this process, pasting together properly matched fragments, under chemical conditions (related to cleavege patterns structure and complementary lows). Thus, two DNA sequences are cut by two restriction enzymes and the fragments may be recombined so that new sequences are produced

On the other hand, the biological circular splicing occurs in a recombinant mechanism (transposition) between bacteria and plasmids. In this case, restriction enzymes cut two circular DNA sequences while ligase enzymes paste them together, producing a new circular DNA sequence. All details about the steps from the biochemical processes to Formal Languages may be found in [34, 38, 59], and some of them will be surveyed in Chapter 1.2 and in the Appendix.

After being investigated in a few papers as an operation with double stranded sequences [34, 33, 17], the splicing operation has been considered as an operation with single stranded strings (see Introduction of Part I). Even if different definitions of splicing have been given in the literature (see Chapter 1), 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. Splicing operation is applied to two words and clearly two strings may be generated, i.e., splicing performs a changing of prefixes and suffixes. Moreover in [57] it is explicitely noted that only one string is obtained by splicing through a rule, since the other one may be obtained by a simmetric rule. On the contrary, the circular splicing opens two circular strings and pastes them together, obtaining a circular string, under some conditions represented into a (splicing) rule. Thus the set of all possible molecules generated in the test tube is nothing but the language (called *splicing language*) of all possible strings that may be obtained by applying the rules to the strings, starting with an initial language. A splicing system is defined by giving the initial set of strings (or axioms) I and the set of rules R (or triples, according to which definition we use).

Based on this operation, the H-systems were introduced as unconventional models of computation and, more precisely, as a generative device of languages (see [58, 59] for a survey on the different models). An H-system is specified by I and R and, given two families of languages F_1, F_2 in Chomsky hierarchy, $H(F_1, F_2)$ denotes the class of languages generated by H-system where $I \in F_1$ and $R \in F_2$. Indeed, the set of rules, representation of splicing specification (cutting and rejoining sites) by using special symbols, may be seen as a language. The generative power of an H-system, i.e., the type of languages that are generated by it, has been deeply studied and will be surveyed in Chapter 1.3 as well as different variants of splicing systems (Chapter 1.4).

By iterating a finite set of splicing rules starting from a finite (even regular) language, only regular languages may be obtained [17, 66]. Hence the power of finite state automata cannot be increased. However, as soon as we use a regular set of rules (with an additional hypothesis on splicing operation that leads to the definition of Extended Splicing Systems), we reach the power of Turing Machines [56]. The jump from finite automata to Turing Machines has encouraged a lot of works and variants proposal, mainly oriented in how to get rid the infinity in the above-mentioned result, maintaining the universality. Several ideas are explored in the literature, new models are introduced (such as "counting" the number of copies of

the used strings, i.e., splicing with multeplicity, permitting or forbidding contexts) and in all cases splicing systems are computationally complete, i.e., every algorithm (effective procedure) can be carried out by a splicing system. Thus, the existence of universal programmable DNA computers based on splicing operation is *theoretically* possible. This may be done by showing that the action of any computable function can be realized by a splicing system [26]. In this sense suitable definition of splicing systems have been formalized, and by using extended splicing systems, it has been proved that every recursive enumerable language can be generated by a splicing system, i.e., the work of any computable function can be carried out by a splicing system, or, in other words, everything is Turing-computable can be computed by this DNA (finite) model of computation (see [38], and bibliography in [44] for more details). Decidability questions have been firstly tackled in [25].

In the Introduction of Part I we will enlighten better the hypotheses under which we work. My thesis concerns H-systems and finite state automaton, precisely regular languages and languages generated by H-systems with both finite initial language and set of rules. We would like to find another characterization of infinite (but regular) languages through the generative mechanism of splicing systems with finite objects, as well as automata. H-systems are extensively presented in Chapter 1 while some digressions on other models are glanced in Chapter 1.4.

We have also been involved in the study of (finite) circular splicing power and its relationship with regular languages. The framework of circular splicing is less developed than linear splicing, probably because circular splicing deal with the intricate circular languages, i.e., with languages composed by words in which final ends are pasted together. Each circular word, that can be written in a circle and may be read in clockwise orientation, is in bijection with all the possible words obtained by cutting it in all possible ways (full linearizations of circular words). Circular words have been already treated in Combinatorics on Words through conjugacy relation $\sim$ that is an equivalence relation among words [16, 51]. In short, two words are conjugated if they are the same word read along a circle, i.e., unless a cyclic permutation

of their letters. Since no concatenation may be defined (conjugacy relation is not a congruence) splicing operation seems to be like a suitable concatenation of circular words. Indeed, given two equivalence classes (two circular words) it is possible to define the concatenation of two circular words via splicing. Thus, the splicing becomes a particular concatenation (or a particular "circular shuffle" on circular words, constrained by a rule). While the linear splicing loses "pieces" of the input strings, the circular splicing does not (at least in Head and Paun's definitions).

Furthermore, since no characterization of circular regular languages is known, the comprehension of circular splicing power may be another approach to face this problem. Again, different variants of circular splicing have been mentioned by the literature with the aim of proving universality results and are collected in Chapter 5.2.

Let me say some final remaks, before sketching an overview of the contents of this thesis. In [57], the authors observed that splicing operation may be viewed as an alternative to the rewriting operation on which the present day computers are based. This is the reason for which the main problem in Splicing theory concerns its computational power and universality results inspired different proposal of splicing systems. Moreover, even if a lot of works remains to be done for its feasible realizations, experimental works on splicing sistems have been successfully done also by the Biomolecular Computing Group at Binghamton [5]. This group proved the vialability of finite Head's model using a single step procedure in lab, making clear the predective capability of splicing [48].

My thesis is purely a language-theoretic oriented investigation and it is organized in two parts. The present introduction is followed two parts dealing in turn with Linear Splicing Systems and Circular Splicing Systems, each handling the state of the art and personal contributions. The splicing operation for both linear and circular strings is formally defined. We will give the state of the art, paying particular attention to the relation existing between splicing systems and regular languages, this being my research area. As to the latter, we will also present contributions already obtained that close open questions. Finally, we will mention very recently discovered links

between splicing and codes, referring to a not much known and developed branch in Splicing Systems Theory. Indeed for the first time we approach circular splicing with finite objects and no additional hypothesis. All the results are presented at the beginning of each chapter regarding them.

Part I

Linear Splicing

In the introduction we have described the biological phenomenon underlying splicing systems, observed and formalized for the first time by Head [34]. DNA molecules are cut by restriction enzymes (proteins) leaving two pieces of DNA which are pasted, if some biological conditions on the sticky ends happen, by ligase enzymes, producing new molecules. We have already observed that the translation of this recombinant mechanism is natural in Formal Language Theory: splicing systems are generative mechanisms which start from a given set of strings and a given set of splicing rules in order to generate a language. Indeed, the phenomenon is nothing but a pattern recognition inside a string, cutting the string at a specified point inside the factor and then pasting a prefix of a word with a suffix of another word. The two words are related each other by a rule, representing the combined action of enzymes. As noted in [29], splicing exchanges left and right ends of words. Some considerations are necessary, before giving the overview of the contents of this part, since we choose splicing operation with a few of hypotheses (commonly used in the literature).

Firstly, sites can occur more than once and whenever this happens, the sites are chosen nondeterministically. The rules are applied in a maximal parallel manner: each rule which can be applied should be applied.

Moreover, it is evident that a recombinant mechanism happens with more probability if there is an high number of DNA molecule copies instead of a small number of it. If the number of copies is unbounded or, better, is high, then the probability of DNA molecules"meeting" is higher, since no topology is defined in the test tube. We assume that if we splice x, y obtaining w, w', we may use again x, y as terms of splicing, that is the strings are not consumed by splicing (using notation in Chapter 1.4, we work without multeplicity). Similarly, there is no restriction on (the number of the copies of) the newly obtained w, w'. In [25] a consideration is esplicitely done about the assumption that all of the initial strings or DNA strands are available in unbounded quantity. There are phisical reasons to consider this assumption reasonable for modeling purposes. The number of molecules in a few grams of most compounds is on the order of 10^{22} or more. In [25] is shown that this assumption is computationally needed since otherwise

there are specifically languages for which some problems are undecidable (such as the membership problem). At the opposite site, others consider this assumption not so realistic.

We consider strings, not double stranded structure, since Formal Languages Theory does not yet deal directly with "linked pairs" of strings. This is the reason why Head considered splicing on strings. Recently, some papers have intestigated bidimensional languages (see, for example [30]). Previously, splicing operation has been approached by using the bidimensional structure of DNA through dominoes languages, already defined in other contexts [17]. Even if this approach is more realistic for modelling splicing behaviour, it seems to be quite difficult. As observed in [34, 57], due to complementary lows ($A-T$, $C-G$), there is no loss of information if the double stranded DNA molecule is identified with one single stranded DNA sequence. Furthermore, we do not restrict the alphabet to four letters. Indeed it is well-known that the symbols of an arbitrary alphabet can be codified on two letters preserving the "place" of a language in the Chomsky hierarchy.

Finally, as done in a series of papers, the splicing rules are considered symmetric, i.e., starting from two words and by using a rule, two words are generated. Clearly, the simmetric case corresponds to considering each rule together with its simmetric rule. In [55], for example, symmetry is not automatically imposed, i.e., only one word is generated at each step.

Let us describe our work on linear splicing. After recalling some basics on ormal languages (Section 1.1), we give the three definitions of splicing, with a lot of old and new examples on $H-$systems (Sections 1.2 and 1.3). Thus, in Section 1.4 we survey some variants of linear splicing models and their power. We focus our investigation on regular languages that can be gerated by linear splicing by using both finite initial language and finite set of rules, since until now it is unknown a characterization of this subclass of regular languages (finite) splicing generated. In Chapter 3 we show a decidability property with respect to this problem. Still considering finite splicing, we compare the three splicing systems giving some considerations and inclusion results (Chapter 2).

Chapter 1

Definitions on languages, splicing systems and main results

In this chapter we will give all the basics on languages, used in the rest of the thesis. We suppose the reader familiar with elementary finite state automaton theory and so we fix here only few notions and notations (see [3, 40]), Then we formally define linear splicing operation and H-systems, with related state of the art. Finally, we glance the plan of linear splicing results.

1.1 Basics on languages

Let us fix some notations. We denote by A^* the free monoid over a finite alphabet A under the operation of concatenation. The elements of A^* are called *words* (or strings) and the elements in A are called *letters* (or *symbols*), while a *language* X is a subset of A^*. The empty string is denoted by 1. The set of nonempty strings over A, that is $A^* \setminus 1$, is denoted by A^+. Set $|w|$ the length of a string $w \in A^*$, i.e., the sum of the occurences of each letter of A in w and, for every $a \in A$, $w \in A^*$, we denote $|w|_a$ the number of occurrences of a in w. The cardinality $|X|$ of a language X of A^* is the

number of words in X.

Given a finite alphabet A, a *multiset* over A^* is a mapping $M : A^* \to \mathbb{N} \cup \{\infty\}$ such that $M(x)$ is the number of copies of $x \in A^*$ in M. The set $supp(M) = \{w \in A^* | M(w) > 0\}$ is called the *support* of M. Thus a set S is a multiset with $S(x) = 1$ for $x \in S$ and $S(x) = 0$ for $x \notin S$.

A word $x \in A^*$ is a *factor* (resp. *proper factor*) of a word $w \in A^*$ if there are words $u, u' \in A^*$ such that $w = uxu'$ (resp. $x \neq w$). If $u' = 1$ (resp. $u = 1$) then u (resp. u') is a *prefix* (*suffix*) of w. A prefix u (resp. suffix) of w is a *proper prefix* (resp. *proper suffix*) if $u \neq w$. Given a word $x \in A^*$ (resp. a language $L \subseteq A^*$) we denote $Fact(x)$ (resp. $Fact(L)$) the set of all factors of x (resp. of all factors of all words $w \in L$).

Let $\mathcal{A} = (A, Q, \delta, q_0, F)$ be a finite state automaton, where A is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of final states. The transition function δ is classically defined. The language accepted (recognized) by $\mathcal{A}$ is denoted $L(\mathcal{A})$. As usual, in the transition diagram of a finite state automaton $\mathcal{A}$, each circled vertex will be a final state and the initial state will be a vertex with an arrow (without a label) pointing towards it. Furthermore, a *path* is a transition sequence $q_1a_1q_2 \cdots a_{n-1}q_n$ such that $q_1, \ldots q_n \in Q$, $a_1, \ldots, a_{n-1} \in A$ and $\delta(q_i, a_i) = q_{i+1}$, for $i = 1, \ldots, n-1$. A *closed* path is a path $q_1a_1q_2 \cdots a_{n-1}q_n$ in which $q_1 = q_n$. A path is a *successful* path if q_1 is the initial state in $\mathcal{A}$ and $q_n \in F$.

A finite state automaton $\mathcal{A}$ is *trim* if each state is accessible and coaccessible, i.e., if for each state $q \in Q$ there is a transition sequence from q_0 to q and from q to a final state. Furthermore, $\mathcal{A}$ is *deterministic* if, for each $q \in Q$, $a \in A$, there exists at most one state $q' \in Q$ such that $\delta(q, a) = q'$.

It is well-known that a language $L \subseteq A^*$ is regular if and only if it is recognized by a (trim deterministic minimal) finite state automaton $\mathcal{A} = (A, Q, \delta, q_0, F)$, i.e., $L = L(\mathcal{A})$. An algebraic description of some subclasses of the class of the regular languages $L \subseteq A^*$ has been given by means of the *syntactic monoid* $\mathcal{M}(L)$ of L. This is the quotient of A^* with respect to the syntactic congruence $\equiv_L$, defined as follows: $w \equiv_L w'$, with $w, w' \in A^*$, if and only if $xwy \in L \Leftrightarrow xw'y \in L$, for all $x, y \in A^*$.

Given a regular language $L \subseteq A^*$, it is well known that there exists a *minimal* finite state automaton $\mathcal{A}$ recognizing it. This canonical automaton is unique up to a renaming of the states and (when we make it trim) has the minimal number of states. It is related to the syntactic monoid of L and can be obtained thanks to standard construction algorithms [40]. In the next we will always consider trim deterministic automata even if this is not explicitly stated. If L is regular, a well known result states that $\mathcal{M}(L)$ is a finite monoid, i.e., the number of equivalence classes is finite.

We denote Fin, Reg, CF, CS, RE the class of respectively finite, regular, context-free, context-sensitive, recursive enumerable languages. We suppose known all the properties of these classes of languages [40]. We only recall that a nontrivial family of languages, i.e., containing at least one language different from $\emptyset$ and $\{1\}$, is *full AFL* (full abstract family of languages) if it is closed under arbitrary morphisms, intersection with a regular set, union, concatenation and Kleene $*$. For example, Reg, CF, RE are full AFL.

1.2 Linear splicing operation

We present the three definitions of linear splicing operation and linear splicing systems. We point out again, that before using a splicing rule r, the two words, spliced by r, are copied in order to be still used.

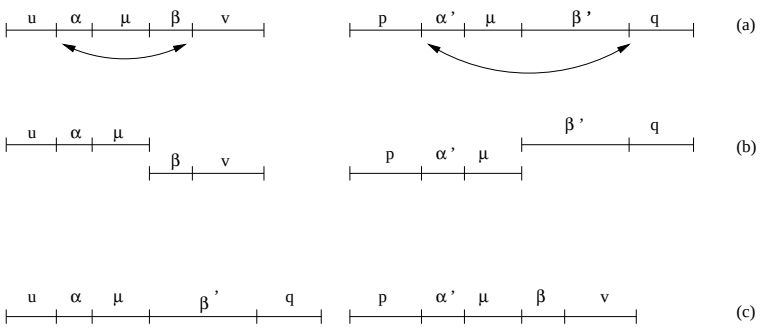

Figure 1.1: *Head's splicing: (a) patterns recognition, (b) cut, (c) paste.*

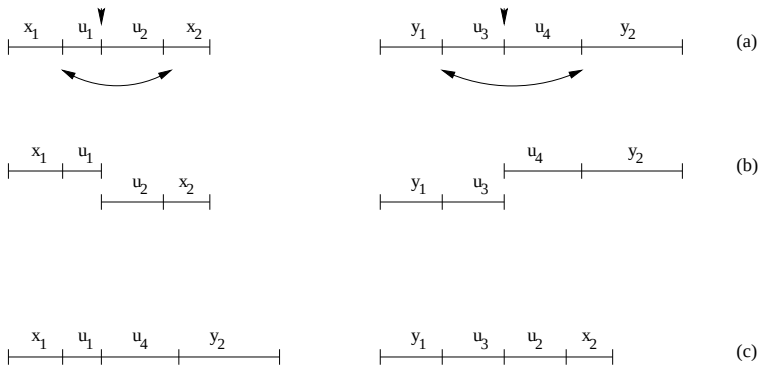

Figure 1.2: *Paun's splicing: (a) sites recognition, (b) cut, (c) paste.*

`Head's definition [34]`. A *Head splicing system* is a tuple $S_H = (A, I, B, C)$, 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 [55]`. Given a finite alphabet A and two special symbols $|, \$$ not in A, a *splicing rule* (over A) is $r = u_1|u_2\$u_3|u_4$, $(u_i \in A^*, i = 1, 2, 3, 4)$. For such a rule r, $x = x_1u_1u_2x_2, y = y_1u_3u_4y_2$, the splicing rule r applied to x, y produces $w = x_1u_1u_4y_2$ and $w' = y_1u_3u_2x_2$ with $x_1, x_2, y_1, y_2 \in A^*$. A *Paun splicing system* is a tuple $S_{PA} = (A, I, R)$ consisting of a set R of rules and $I \subseteq A^*$, called *initial language*.

`Pixton's definition [66]`. Given a finite alphabet A, a *splicing rule* is a triple $r = (\alpha, \alpha'; \beta)$ that applied to $x = \epsilon\alpha\eta$ and $y = \epsilon'\alpha'\eta'$ produces $w = \epsilon\beta\eta'$ and $w' = \epsilon'\beta\eta$. A *Pixton splicing system* is a tuple $S_{PI} = (A, I, R)$ where R is a finite collection of splicing rules and $I \subseteq A^*$, called *initial language*.

Notation. We will sometimes call "rules" R also the set of triples $B \cup C$ in Head's definition and "axioms" the words in I.

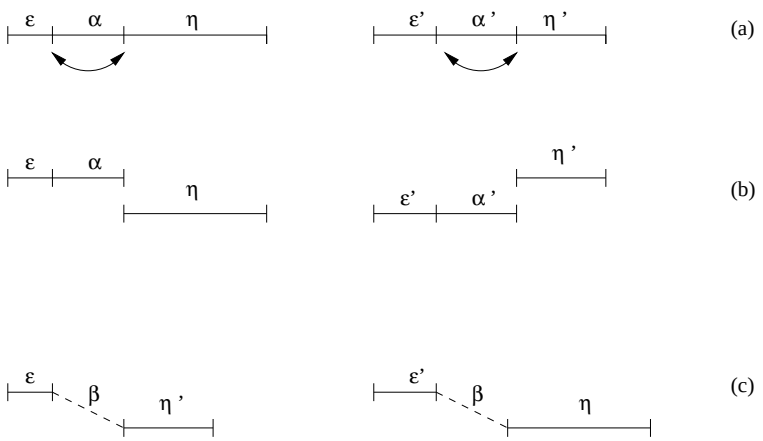

Figure 1.3: *Pixton's splicing: (a) sites recognition, (b) cut, (c) paste with substitution.*

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. As just described in the introduction, the enzymes' action on DNA produces blunt ends and staggered cleavages. In the recombinant processes of two segments the overhangs must match under some conditions. This is the reason why Head introduced two sets of patterns (this explains why pair B, C appears in his definition). On the contrary, Paun and Pixton work more abstractly on single strands of DNA.

In [64] a different definition of Paun splicing operation is used: two strings are spliced using a (Paun) rule $u_1|u_2\$u_3|u_4$, the substrings u_2, u_3 cannot appear in the final results, i.e., splicing produces only one word containing u_1u_4 as factor (thus simmetry of splicing operation is not imposed). For this reason, u_2, u_3 are called *invisible* sites. Some restrictions on this set of words is often used in order to obtain some results (see [64, 65, 66] for other comments).

Pixton declared in [65] he used a semplified formulation of splicing introduced in [64] with respect to the one of Head. In fact, α, α' are respectively

referred as the left and the right sites of the rule and β as the *glue*. Also Paun's definition may be translated in Pixton's formulation (translating the Paun's rule $u_1|u_2\$u_3|u_4$ into the Pixton rule $(u_1u_2, u_3u_4; u_1u_4)$). Thus, some results obtained by using on Pixton systems, such as finite splicing closure results (see Section 1.3) imply the same results using Paun's formulation. However, in general, regularity for not finite splicing systems is not preserved, by using the above mentioned translation [65]. Other remarks will be given in Chapter 2.

In the Splicing Systems framework, it was natural the possibility of defining the *minimal splicing system*, which is esplicitely introduced in [55, 59] and which is a typical research topic concerning *descriptional complexity*. It goes without saying that this notion is the counterpart of the minimal automaton for regular languages. Here, the minimality of the system could be referred to the cardinality of R or to the length of the rules in R. Some initial steps towards this notion were done implicitely in [29]. Superfluos patterns are individuated and so an algorithm reducing the size of R, without changing the generated language, is given. In Section 8, a complete answer to this question will be given for languages over a one-letter alphabet, by using Paun's definition.

Notation. Given splicing system S_X, with $X \in \{H, PA, PI\}$ and set of rules R, we denote by $(x, y)\vdash_r (w', w'')$ the fact that the two strings w', w'' are produced from (or spliced by) x, y by using a rule r (or triples). Note that x and y are not necessarily distinct and that $(x, y)\vdash_r$ is different from $\{x, y\}\vdash_r$ [38].

Let $L \subseteq A^*$ be a language and let S_X be a splicing system. We denote by

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

Thus, we define

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

Let us now give the formal definition of (iterated) splicing language.

Definition 1.1 (Splicing language) *Given a splicing system S_X, with $X \in \{H, PA, PI\}$, the language $L(S_X) = \sigma^*(I)$ is the language generated by the splicing system S_X, I being the initial language in S_X. A language L is X generated (or L is a X* splicing language*) if a splicing system S_X exists such that $L = L(S_X)$.*

Roughly speaking, a splicing language is the smallest language obtained by iterated application (with respect to a definition) of the rules, starting with I. We will see in Chapter 2 that linear splicing is not closed under union.

Notation. More in general, we will sometimes denote $L(S)$ the splicing language generated by a given splicing system S.

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 1.1 *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 $L = L(S_Y)$?*

When I, R are both finite we are able to solve this problem (see Chapter 2).

1.3 H-systems: state of the art

Splicing systems have been proposed as an unconventional model of computation and so a basic problem was the comparison with Chomsky hierarchy, namely with the classical models of computation [40]. Since I, R

are languages, the computational power of splicing systems obviously depends on which level of Chomsky hierarchy they belong to. Precisely, let F_1, F_2 be two families of languages in the Chomsky hierarchy, i.e., $F_1, F_2 \in \{Fin, Reg, LIN, CF, CS, RE\}$. We denote

$$H(F_1, F_2) = \{\sigma^*(I) \mid S_{PA} = (A, I, R) \text{ with } I \in F_1, R \in F_2\}$$

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. Clearly, in a similar way, we can define the class of languages generated by Head or Pixton splicing systems. Since we focus our attention deeply on Paun splicing systems, we have defined only them, as generally done by the literature.

A collection of known results about $H(F_1, F_2)$ is contained in in [38] and presented in different papers (starting with [55], where H-systems are defined) by giving very useful and interesting properties of splicing languages. 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

Many of the relations in the Table above, obtained by several intermediate results, presented in different papers, are of interest. They have been obtained by several intermediate results, presented in different papers and will be explained in details. 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. For example, $Fin \subseteq H(Fin, Reg) \subseteq RE$ stands for starting with a finite language, using regular rules is possible to obtain some

languages in RE, but not all; moreover, $Fin \subseteq H(Fin, Fin) \subseteq Reg$, while, $H(Reg, Fin) = Reg$.

Notation. In the following, with *finite splicing systems* we mean splicing systems with both I, R finite.

There are regular languages that cannot be generated by finite splicing. The following example is one of them, firstly given in [29], and others are contained in [8].

Example 1.1 The regular language $(aa)^*$ cannot be generated by finite splicing.

Thus a natural problem is the following.

Problem 1.2 *Characterize regular languages which belong to* $H(Fin, Fin)$.

We have studied this problem and so we will survey the state of the art with respect to it. We stress that our aim is to find an alternative characterization of those regular languages generated by splicing, but by using only finite objects (initial language and set of rules). Trying to give an answer to this problem, in [7] is shown how to obtain Reg using iterated splicing with $I \in Fin, R \in Fin$, but with "marked rules".

Theorem 1.1 *[7] A language is regular if and only if it is generated by a finite marked splicing system.*

Obviously, classical questions about finiteness, membership and equivalence naturally arise. Clearly, when we consider the class of languages generated by iterating a finite set of rules starting with a finite or regular initial language, since $H(Reg, Fin) = Reg$ and or $H(Fin, Fin) \subseteq Reg$, all those questions are recursive (i.e., decidable), since they are decidable for Reg. For instance, the splicing membership problem, suggested by Head [34], may be formulated as follows: given a splicing system and a word, device an algorithm (if possible) to answer the question of whether or not the word is spliced by the given splicing system. This problem was proved

to be recursive firstly in [29], for a particular class of (finite) Head splicing systems S_H: the author proved that these particular (finite) splicing systems generate a proper subclass of regular languages. The regularity of the (Head) splicing system S is given by associating an automaton $\mathcal{A}$ to S. The constructive algorithm is organized as follows. Given a (finite) Head splicing system, firstly determine whether B, C are crossing disjoint, i.e., there do not exist two patterns, one in B, the other one in C with the same crossing. This condition, not assumed by the definition given by Head, is evaluated to be necessary for stating the result, otherwise illegal spliced words may arise. Thus, a reduction procedure of the cardinality of B and C, is performed by eliminating those patterns which are unuseful (with respect to some criteria). This step is related to descriptional complexity issues later defined and already mentioned in the previous section. Moreover, an algorithm is given deciding whether the reduced and crossing disjoint splicing system is *permanent*. Roughly speaking, a splicing system is permanent if, for each pair of strings containing patterns, any factor which is a crossing of a site is still a factor of the spliced word. Then, Gatterdam stated the following result.

Theorem 1.2 *[29] For each crossing disjoint, reduced and permanent finite Head splicing system S, a nondeterministic finite state automaton $\mathcal{A}$ may be constructed such that $L(S) = L(\mathcal{A})$.*

The permanent notion was introduced by Head in [34], together with the concept of *persistent* splicing system. This definition is the same of permanent, where any factor which is a crossing of a site contains a factor of the spliced word. In particular, in the next theorem Head proved that a remarkable subclass or regular languages is generated by splicing.

Theorem 1.3 *[34] A language L is a SLT (strictly locally testable) language if and only if L is generated by a (finite) persistent splicing system.*

Since permanence implies persistence, even if in some cases (full contexts splicing systems) the definitions coincide, Theorem 1.3 is stronger than Theorem 1.2. Nevertheless, no algorithm is given, because the proof of Theorem

1.3 is not constructive. Thus, Gatterdam's algorithm may be considered a realization of Head's result.

In [34] the splicing operation is explicitly linked with the concept of *constant* introduced in [69]. Given a language $L \subseteq A^*$, a word $c \in A^*$ is a constant if, whenever $ucv, pcq \in L$ then $ucq, pcv \in L$. It is evident the similarity between a constant and a crossing in splicing definition. Head proved that persistent splicing languages, that coincide with SLT languages for Theorem 1.3, may be generated by $S = (A, I, A^P, A^P)$, for $P \geq 1$ (*uniform splicing systems* or Null Context H-systems). To do this, he used a result in [24], where it is shown that a language L is SLT if and only if there is an integer k such that all strings in A^k are constants with respect to L.

If S is not persistent, regularity proofs were given by Culik and Harju [17], through dominoes languages to model double stranded DNA and finite Head splicing systems are simulated.

Kim showed the regularity of $L(S_H)$ by giving an algorithm which constructs a finite state automaton accepting the splicing language spliced by the system, when the initial language is regular [47]. Moreover, in the same paper a characterization of SLT languages is given, based on automata properties. We find contructive proofs for the regularity of languages considered in [17] and for the strict locality of the languages considered in [34].

It is worth of note that $H(Fin, Fin)$ is not comparable with LT (locally testable) languages. Indeed there exist a regular language not in LT that can be generated by finite (Paun) splicing, and there is a locally testable language that cannot be generated by finite (Paun) splicing [6].

Pixton gave a simpler proof of the regularity of finite splicing systems, with respect to his definition, building a graph capturing the splicing's behaviour [66]. The idea of the proof is to record the sequence of splicing steps in a single string, by using suitable brackets and to process it.

Theorem 1.4 *[66] (Regularity preserving theorem) Given a Pixton splicing system $SC_{PI} = (A, I, R)$, where $I \in Reg$ and $R \in Fin$, we have $L(S_{PI}) \in Reg$.*

The same result is extended to full AFL languages [65] and also by using

Paun's definition, translating splicing into two alternative forms (cutting and pasting scheme - CH schemes, matrices of languages).

Theorem 1.5 *[64] Given a Pixton splicing system $S_{PI} = (A, I, R)$, where $I \in FA$ a full AFL and $R \in Reg$ with finitely many visible sites, we have $L(S_{PI}) \in FA$.*

Some papers directly concern with subclasses of splicing systems. One, already mentioned and called null context splicing system (NCH) in [34], is defined by restricting the triples to the form $(1, x, 1)$ and so each crossing is a costant for $L(S)$. In [53] *simple H-systems* (SH) were introduced as a subclass of NCH systems, where each crossing is a letter, called *marker*, $a \in M \subseteq A$. In the same paper is shown that only regular languages, but not all SLT, may be generated. As an example, $(aabba)^* \in SLT$, but $(aabba)^*$ is not generated by a SH system. We can decide whether a regular language is generated by these systems and closure properties of SH splicing languages are provided. Moreover, each regular language $L \subseteq \{a\}^*$ is generated by SH if and only if $L = a^*$ or $L = a^+$. An extension of this result is provided by proving that w^* is SH if and only if $|w|_a = 1$, where a is the crossing of the rules in SH. In [53] investigations on SH where the length of the crossing is more than one are questioned.

As a consequence of a theorem in [34], the family of SLT languages conicides with the family of NCH languages, but in [36] Head has given a decision procedure which determines for any regular language L, whether L is SLT, using a hierachy of SH splicing systems. He proved that SLT is the union of the families of languages generated by it, answering the questions proposed in [53] and above-mentioned.

In [7] is shown that regular languages satisfying a condition form a proper subclass of Reg which properly contains SLT. Thus, the authors give an alternative characterization of SLT languages through simple properties of the automaton proving that this class is generated by finite marked splicing systems (using Paun definition). This characterization is larger than the one contained in [35], being the latter coincident with SLT.

The concept of constant is yet strongly used in [35], where decidable

properties for regular languages and a characterization of regular languages generated by particular splicing systems are given. The paper faces the generating power of (Paun) finite splicing systems where each rule is *one-sided context*, i.e., each rule is of the form $u|1\$v|1$ (or $1|u\$1|v$) and the reflexivity hypothesis is used, i.e., rules $u|1\$u|1$ (or $1|u\$1|u$) are also in R. In this case, the words appearing in each rules are constant for the generated language. Interesting properties relating constants and paths in the automaton are also given.

In [31] simple splicing systems are generalized, by considering all rules of the form $a|1\$b|1$, $a, b \in A$. In terms of certain directed graphs, the so-called *semi-simple splicing languages* are characterized also interpreted by using constants. Indeed, by using Head's results, semi-simple splicing languages are SLT and can be generated by reflexive splicing systems (not all splicing languages can be generated by reflexive systems, as asserted in this paper).

All the above mentioned contributions state the possibility of going from splicing systems to automata or, in some special cases, algorithms deciding whether regular languages are generated by special splicing systems. Thus, we have focused our investigation on Problem 1.2, since, up to now, no answer is given to it.

We conclude this section by reporting that Kim has solved a variation of Problem 1.2 for Head splicing system. In [46] he presented an algorithm for the problem of deciding whether or not a regular language can be generated by a (Head) finite splicing system using a given set of rules. In other words, given a regular language L and a (finite) set of triples X, we can decide whether there exists a language (initial language) $I \subseteq L$ such that by using only rules in X we generate L. The idea is to investigate the structure of the (trim deterministic finite state) automaton recognizing L, in order to find suitable properties of the cycle and estabilishing a correspondence between cyclic strings spliced by a system S, by using X, and cyclic paths of the state automaton whose language is $L(S)$.

1.4 Other models of linear splicing systems

In the previous section, we have presented the state of the art with respect to a particular type of splicing system. There are other kinds of splicing systems, inspired by old notions, such as Post normal systems, and new ones. Firstly, for the sake of completeness, we specify that non iterated splicing systems S may be defined when only one application of the rules is allowed starting form I. It is evident that the iterated case is more realistic, but, however, a complete table is given, similar to the one depicted above for H systems. We report only that $Fin = S(Fin, FA)$, for every FA, while $RE = S(FA_1, FA_2)$, for every FA_2 and $FA_1 \in \{CS, RE\}$, or $FA_1, FA_2 \in \{LIN, CF, CS, RE\}$ [55, 38].

In the framework of descriptional complexity, the *radius* of a rule is defined in [55] as the maximal length of a site u_i. Thus $H(FA, p)$ denote the class of languages generated starting with $I \in FA$ and a finite set of rules, where each rule has radius less that or equal to p. Some interesting results are provided, and we only mention that, even if $H(Fin, 1)$ and Reg are different, they are equal modulo a coding [55, 38].

A lot of variants of splicing were considered in [58], especially with the aim of building a system with the same power of a Turing Machine and as simple as possible. Indeed, a series of new charcterizations of RE are obtained, starting from "simple" families FA_1, FA_2, by fixing suitable constraints on splicing rules.

The aim of this section is to describe a few of them, collected in [38, 57, 59] without going into details and without thinking to be exhaustive.

Extended splicing systems EH are defined by means of an auxiliary alphabet T (*filter*) of terminal symbols, that allows to overcome the limitations of H-systems (the splicing language generated by EH systems is $L(S) \cap T^*$). Indeed, by using a finite initial set of words and a finite (resp. regular) set of rules, EH generate all Reg (resp. RE), so they have the same computational power of Turing Machines. The proof of universality are given by simulating the grammar type-0 productions.

Other models introduce multeplicity on the initial set. The idea of using

multeplicity in controlling splicing operation was firstly used in [25]. This assumption allows to consum the terms of splicing x, y when producing new strings. Thus, it may be realized by considering that there are infinitely many copies of each string. We are leading to multisets, sets with multeplicity associated to their elements (see Section 1.1). In the general case, we have multeplicity worked with multisets M such that $M(x) = \infty$, for each $x \in Supp(M)$. Thus, splicing operation and splicing systems may be suitable defined taking into account the disappering of one occurrence x, y when producing w, w'. As enlightened in [38], the case of multisets has surprisingly strong influence on the power of iterated splicing, as reported below.

Theorem 1.6 *[38] Each RE language is generated by using an EH splicing system with both I and R finite sets, where multeplicity is defined on I.*

Multeplicity plays a crucial role, for example, with respect to splicing membership problem, previously defined. Indeed, the concept of *sequential (Head) splicing system* is introduced in [25], differing from Head splicing system in that it restrict the initial number of copies of some strings. The main result is reported below.

Theorem 1.7 *[25] The splicing membership problem is recursive unsolvable with respect to sequential splicing systems.*

This theorem is proven by showing that the range of any Turing Machine can be encoded in the language generated by an associated sequential splicing system. The same result has been strenghtened and proved in a simpler way by others and on this basis characterizations of RE languages are obtained in [26].

Universality for finite sets of rules is proven also for EH systems with checking context conditions (permitting, forbidding), or defining a communicating distributed H system, where several grammars are put to work together, according to a specified protocol, for generating a common language [57].

1.5 Linear results: overview

We stress here that we have faced with Problem 1.2. Firstly, we have compared the power of the three finite splicing systems, showing strict inclusions among them (Chapter 2). A preliminary version of this result is [14] and published in [15]. The complete solution of Problem 1.2 is a work in progress. We give a result toward this characterization, by using syntactc monoid in Chapter 3.

Chapter 2

A comparison of the three variants of finite linear splicing systems

In Chapter 1.2 we have given the three definition of splicing operation and we have posed the problem of comparing these definitions. Indeed, it is quite evident that, in the finite case, $S_H \leq S_{PA} \leq S_{PI}$, i.e., a language generated with (finite) Head splicing rules may be rewritten using (finite) Paun splicing rules and the latter may be rewritten with (finite) Pixton splicing rules [38, 64] (see Chapter 1.2).

The question is whether these inclusions are strict or not. Here we show that the three main definitions of linear splicing operation give rise to different subclasses of regular languages, when a finite set of rules is iterated on a finite set of axioms [1]. More precisely, we show that the family of regular languages generated by finite splicing, as defined in the early paper by Head [34] is strictly included in the family defined later by Paun [55], which is in turn strictly included in the splicing family defined by Pixton [66]. We describe instance languages in the difference sets, and we prove how they cannot be generated by the smaller families.

Some considerations on the three definitions of linear splicing are followed

[1] The results presented in this chapter are contained in [14] and [15].

by our result.

2.1 Deeply remarks on linear splicing definitions

In 1992 Head [33] redefined his model of splicing, no more considering patterns defined by triples, and rules for pasting dictated by the patterns as done in Chapter 1.2. He defined a (reflexive and symmetric) relation between patterns, which was encoded by pairs of strings, such that strings are allowed to tie together only if they are originated by strings in which patterns are in relation.

We suggest that a common framework to understand hierarchy among three definitions of splicing can be based on the binary relation on strings defined by the rules: as pointed out above, in [33] rules in splicing systems are described by a binary relation between pairs of strings which must be factors of those sequences that can be tied together by splicing. It turns out that this relation in Head's definition is an equivalence relation, since only strings (called patterns) having the same crossing μ can be bound together. Thus, the string μ in the pattern directly creates the relation allowing to paste the strings. For instance, one could not design two distinct groups of patterns, whose generated strings must not be cross-pasted, if all these patterns need to have the identical empty crossing μ. Clearly, this fact restricts the power of the splicing operation, as we will see by giving a language that cannot be generated under this definition of splicing since it needs distinct groups of patterns with empty crossing.

Differently, as shown in Example 2.2, the binary relation generating a Paun splicing language is not necessarily an equivalence one, thus making Paun's definition more general than the one of Head. Note that by Paun's formalism for splicing systems patterns are pairs which define a binary relation; strings are allowed to tie together if they contain patterns which are in this relation. The relation among patterns could be, generally, any binary relation. This fact effectively links in a very strong way the two steps of cutting and pasting, since now the splicing system could be defined in such a way that the relation between two patterns means that when they are cut,

the generated strings are immediately pasted together with no interference from strings cut from some other patterns. We just relate each pattern with only one pattern, independently from their content. This approach is very powerful, and it has often been chosen in the literature on splicing systems.

Moreover we stress that Pixton splicing operation is not really a formal description of the biological cut and paste recombinant behaviour of DNA molecules, as splicing operation defined by Head and Paun are, but it describes also a substitution operation related to splicing. In Pixton's definition the cutting step and the pasting step are again operated one immediately after the other, but it is even allowed to *substitute* part of the pattern before pasting the strings originated by the cut.

2.2 Inclusion results

We have just said that the regularity of finite splicing operation holds for each of the three definitions, since it is quite simple to check that given their respective definitions, (finite) Head splicing languages are always (finite) Paun splicing languages, and these are always (finite) Pixton splicing languages. Until now, it was unknown whether the inclusions among the above classes of languages were strict or not. In this chapter we give a positive answer to this question, providing regular languages that separate the three classes. We firstly present a nonempty class of regular languages that are Paun splicing languages, but that can be generated by Head splicing systems only if the crossing of some triples is the empty word. By this fact, it follows that this class is not closed under union by using Head's definition, while we can show that it is closed under union with respect to Paun's definition. From now on, we will only deal with finite splicing.

Example 2.1 Let $A_1 = \{a, b, c, h\}$. We define $L_{abch} = \{a^2c^*a, ca^2c^*ac, bac^*a, ah, bh\}$, depicted in Figure 2.1. Similarly, we can define L_{defg}, L_{mnop} regular languages on $A_2 = \{d, e, f, g\}$ and $A_3 = \{m, n, o, p\}$, respectively, where L_{abch}, L_{defg}, L_{mnop} are three isomorphic languages. Let us consider the following Paun splicing system $S_{PA} = (A_1, I, R)$, for $I = \{a^2c^na,\ bh, bac^na,\ ca^2c^nac, ah\}$, where $0 \leq n \leq 2$, and $R = \{ba|c\$bac|c,\ ca^2|c\$ca^2c|c,\ b|a\$a|h\}$.

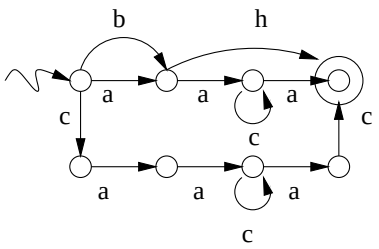

Figure 2.1: *The Paun splicing language* L_{abch}.

It is quite easy to show that S_{PA} generates L_{abch}. In fact it is immediate to verify that the rule $ba|c\$bac|c$ applies only to sequences of the form $bac^m a$, thus allowing to generate $bac^* a$, as well as the rule $ca^2|c\$ca^2c|c$ generates ca^2c^*ac when iterated starting from ca^2c^nac. Note that a^2c^*a cannot be generated by using a rule containing one pattern that is a substring of a^2c^*a, since this rule can be applied also to strings in ca^2c^*ac, thus producing splicing sequences which are not in L_{abch}. Actually, a^2c^*a is obtained by crossing the strings in the set bac^*a with ah, using the rule $b|a\$a|h$, whose effect is to replace the prefix b in the strings in bac^*a with a, avoiding the development of strings not in L. Similarly, we can define the Paun splicing system generating L_{defg} and L_{mnop}.

Let $L = L_{abch} \cup L_{defg} \cup L_{mnop}$ be a regular language over the alphabet $A_1 \cup A_2 \cup A_3$, as defined above. We have the following result.

Lemma 2.1 *The regular language* $L = L_{abch} \cup L_{defg} \cup L_{mnop}$ *is generated by a Paun splicing system and cannot be generated by a Head splicing system.*

Proof. We first give the Paun splicing system generating L. Since L_{abch}, L_{defg}, L_{mnop} are over distinct alphabets, it is immediate to obtain the Paun splicing system for generating L by union of the three Paun splicing systems given in example above. Let us show that L is not generated by a Head splicing system. The proof is by contradiction. Thus, let us assume that $L = L(S_H)$ for some splicing system $S_H = (A, I, B, C)$. Now, given an arbitrary string $s_j = a^2c^ja \in L$, with j arbitrarily large, there are a pair

p_1, p_2 of patterns and two strings w, u from which s_j is derived. We will assume that j is larger than any pattern in $B \cup C$. Assume that $p_1 = (\alpha, \mu, \beta)$ and $p_2 = (\delta, \mu, \gamma)$. We show in the following that μ cannot be a nonempty crossing. In fact, assume that $\mu \neq 1$. It is not restrictive to assume that $s_j = u_1 \alpha \mu \gamma v_2$. Since $|\mu| < j$, otherwise j is not larger than any pattern in $B \cup C$, it follows that $\mu = a^i c^k$ or $\mu = c^k a^t$ for $0 \leq i \leq 2$, $0 \leq k < j$ and $0 \leq t \leq 1$, being $s_j \in L$. Thus the following two cases must be considered.

Case i: Let μ contain a c letter. It follows that p_1 is also a substring of a string x, where $x = ca^2c^mac$; thus, by splicing with patterns p_1 and p_2, a prefix of x can be combined with a suffix of a^2c^*a. But, then p_1 and p_2 generate a string not in L, a contradiction.

Case ii: Assume that μ contains only a letters. Again, if p_1 is a substring of string ca^2c^nac, we obtain as in case i a contradiction. Thus, let us assume that p_1 is not a substring of any string in $\{ca^2c^*ac\}$. Then it must be that $p_1 = (1, a, h)$. If $p_1 = (1, a, h)$, then $p_2 = (\delta, a, \gamma)$, where $a\gamma w = a^2c^na$ and γ is a prefix (eventually empty) of ac^na. But γ is also a factor of ca^2c^*ac, thus p_1 and p_2 produce strings in a^2c^*ac which are not in L, a contradiction.

Case i and ii together show that μ must be empty. Similarly, as above, we can show that given an arbitrary string $z_j = d^2f^jd \in L$, with j arbitrarily large, then z_j is derived by using a pair of patterns p_3, p_4 whose crossing must be empty. Clearly, by definition of L, we need to keep the patterns with empty μ, but we can only separate two groups: one in B and one in C. Two patterns with the same μ, but one from B and the other from C, will not interfere pasting together the strings they cut. We can suppose to put patterns over alphabets A_1 and A_2 in B, and patterns over A_3 in C. Consequently, for the strings in L considered above pattern p_1 can be used with pattern p_3 or p_4 to derive strings which contain letters from both alphabets A_1 and A_2, and hence which are not in L. ■

In the following example we show that in general, the binary relation used to specify rules in Paun systems is not an equivalence relation. Indeed the Paun language described below cannot be generated by any Paun splicing system whose binary relation is an equivalence one.

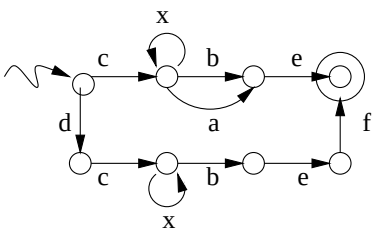

Figure 2.2: *A Pixton splicing language which is not a Paun splicing language.*

Example 2.2 Let us consider $L = L_{abch} \cup \{bu\} \cup \{aef^*e, faef^*ef, def^*e, dh, du\}$, where L_{abch} is the language described in the Example 2.1. Now, it is possible to construct a rule generating $faef^*ef$ and another rule generating def^*e, without ambiguity, but in order to generate aef^*e, we are forced to use the rule $a|h\$d|e$, that can be applied only to strings in ah and def^*e. In fact, if a rule that generates the strings in aef^*e has one of the two patterns as a substring of aef^*e, then this rule could be also applied to strings in $faef^*ef$, thus producing strings not in L. Consequently, a rule for aef^*e must be applied only to ah and def^*e, which implies that such a rule consists of the two patterns $a|h$ and $d|e$ (note that since $du \in L$, it is not possible to have a rule with site $d|1$). Similarly, it is easy to verify that every Paun splicing system generating L must necessarily contain the rule $b|a\$a|h$. Now, suppose that the binary relation between patterns is an equivalence relation. Thus the system contains also the rule $b|a\$d|e$. But this rule can be applied to strings in L (i.e., strings in bac^*a and def^*e), thus producing strings with be and da as factors, which cannot be in L, a contradiction with the fact that L is generated.

Now, we consider a regular language which is generated by a Pixton splicing system and, in the Lemma below, we prove that it cannot be generated by any Paun splicing system.

Example 2.3 Consider the regular language $L' = \{cx^*ae, cx^*be, dcx^*bef\}$, depicted in Figure 2.2. Let us consider the following Pixton splicing system

$S_{PI} = (A, I, R)$, where $I = \{cae, cxae, dcbef, dcxbef\}$, and $R = \{r_1 = (a, xa; xa), r_2 = (dcx, dc; dcx), r_3 = (a, a; b)\}$. We can sketch that $L' = L(S_{PI})$. Indeed, by the first rule r_1 applied to two copies of the initial string $cxae$, and by repeated applications of it, it is possible to generate the language $L'' = \{cxx^*ae\}$. Similarly, rule r_2 is used to generate the strings in $\{dcx^*xbef\}$, starting from the initial string $dcxbef$. Now, the last group of strings, $\{cx^*be\}$ is generated by the application of the rule r_3 to strings in $L'' \cup \{cae\}$, substituting letters a with letters b. The letter a, in the first and third rule, and the string dc in the second rule, forbid the use of the rules to mix strings from the three different families in L': no other string is generated.

We now prove that the language L' considered in the previous example cannot be a Paun splicing language.

Lemma 2.2 *The regular Pixton splicing language $L' = \{cx^*ae, cx^*be, dcx^*bef\}$ cannot be generated by a Paun splicing system.*

Proof. Let us make the following observation. Any Paun splicing system, with finite sets of axioms and rules, could only develop the x^* loops putting in the rules some substrings leading to or trailing from x^*. But there is no way to choose those substrings and avoid the cross-over between strings in $\{cx^*be\}$ and strings in $\{dcx^*bef\}$. The trick of first developing x^*a loops, and then *substituting* the a's with b's, which avoids such cross-overs, is feasible only in Pixton splicing systems. ■

Lemmas 2.1 and 2.2 prove the strict inclusion between Head splicing languages, Paun splicing languages and Pixton splicing languages: these are the main classes of languages studied in the framework of splicing systems. It could be quite interesting and useful to know whether an analogous result between splicing languages generated by using an initial language and a set of rules that are not finite still holds.

Chapter 3

Linear splicing via syntactic monoid

In this chapter we face the problem of deciding whether or not a regular language $L \subseteq A^*$ (not finite) is generated by finite Paun splicing, i.e., there exists $S_{PA} = (A, I, R)$ such that $L = L(S_{PA})$. Here we are able to give a sufficient condition, by means of the syntactic monoid of L, such that if L verifies it, then L is generated by finite Paun splicing.

The technique used is based on syntactic congruence and we argue that this may be applied to characterize regular (finite) splicing languages. Our approach uses two basic ideas: (1) the first one consists of representing the language L by means of sequences of congruence classes in the syntactic monoid, (2) properties of the congruence classes are investigated in order to relate splicing rules to congruence classes.

Given a representation of L in terms of congruence classes, we prove that a certain condition on congruence classes allows us to show that L is generated by finite splicing, by giving a system $S_{PA} = (A, I, R)$ such that $L = L(\mathcal{A}) = L(S)$, being $\mathcal{A}$ the trim deterministic finite state automaton recognizing L.. Clearly, since I and R are both finite, rules in R must allow us to generate by cut and paste the sequences that label transitions obtained by extending or reducing the traversal of closed paths in $\mathcal{A}$. Intuitively, the above-mentioned condition is verified if there exists a congruence class (recall

that the number of congruence classes is finite, since L is regular) such that this class may be uniquely identified by means of a state in $\mathcal{A}$. This class will be as a marker, used to construct I and R. Indeed, by using the marker, we will show how to augment the traversal of every closed path that *follows* it or is *contained* in it. Moreover, we construct the *Reversed Automaton.* Indeed, by the uniqueness of the marker, we will show how to augment the traversal of the closed path that *precedes* it, since they *follows* the marker in the reversed automaton.

3.1 Preliminary definitions and properties of splicing rules

We will always suppose $\mathcal{A} = (A, Q, \delta, q_0, F)$ be a minimal trim deterministic finite state automaton. Given $w \in A^*$ and $q \in Q$, then

$$L_q(w) = \{x | \delta(q, wx) \in F\}$$

$$Q_w = \{q | L_q(w) \neq \emptyset\}$$

Notation. When the automaton $\mathcal{A}$ is not known from the context, we use the notation $Q_w(\mathcal{A})$, instead of Q_w.

Definition 3.1 *Given $w, w' \in A^*$, we define $w \equiv w'$, if and only if $Q_w = Q_{w'}$ and $L_q(w) = L_q(w')$, for each $q \in Q_w$.*

Let $\alpha \in A^*$ and $q \in Q_\alpha$. The transition over α from q, denoted $t_{\alpha,q}$, is the sequence $(q_0, a_1)(q_1, a_2)\cdots(q_n, a_n)$ such that for every $0 \leq i \leq n-1$, $(q_i a_{i+1}, q_{i+1}) \in \delta$, $q_0 = q$ and $\alpha = a_1 a_2 \cdots a_n$. If $\delta(q, w) = q$, then q is called a *cyclic state.*

Notation. In the following we associate a t-uple (u_1, u_2, u_3, u_4) to a rule $r = u_1 | u_2 \$ u_3 | u_4$.

Given a regular language L, we are interested in those t-uple which represent rules that applied to words in L produce words in L. The definition

will be given by means of the trim deterministic finite state automaton recognizing L.

Definition 3.2 (splicing t-uple) *Let $\mathcal{A}$ be a reduced finite state automata. Given a t-uple α, with $\alpha = (x, y, z, v) \in A^* \times A^* \times A^* \times A^*$, the language $L(\mathcal{A})$ is closed with respect to α if and only if:*

(1) for each pair $(q, p) \in Q_{xy} \times Q_{zv}$, then $L_q(xy) \subseteq L_p(zy)$, and

(2) for each pair $(q', p') \in Q_{zv} \times Q_{xy}$, then $L_{q'}(zv) \subseteq L_{p'}(xv)$.

The set of all t-uples with respect to which $L(\mathcal{A})$ is closed will be denoted by $C(\mathcal{A})$. We prove a fundamental property of finite state automata that relates a splicing system to the language accepted by the automata.

Lemma 3.1 *Let $S = (I, R)$ be a splicing system and $\mathcal{A}$ a reduced finite state automata, with $I \subseteq L(\mathcal{A})$. The following two statements are equivalent:*

(1) $R \subseteq C(\mathcal{A})$,

(2)$L(S) \subseteq L(\mathcal{A})$.

Proof. Assume that (1) holds. Given $w \in L(S)$ such that w is derived from $u, v \in L(S)$ by applying rule r, we show that $w \in L(\mathcal{A})$.

Let us prove this fact by induction on the number k of steps to obtain w by splicing. Clearly, if $k = 0$, then $w \in I$ and hence $w \in L(\mathcal{A})$. Assume now that $k = n + 1$. By induction $u, v \in L(\mathcal{A})$. Let $r = (u_1, u_2, u_3, u_4)$ be such that $u = u_i u_1 u_2 u_j$, $v = v_i u_3 u_4 v_j$ and $w = u_i u_1 u_4 v_j$. Since $r \in C(\mathcal{A})$, given $q \in Q_{u_3 u_4}$ and $p \in Q_{u_1 u_2}$, then $L_q(u_3 u_4) \subseteq L_p(u_1 u_4)$. But, this fact implies that $w \in L(\mathcal{A})$, as required.

Assume that (2) holds. Let $q \in Q_{u_3 u_4}$ and $p \in Q_{u_1 u_2}$, then we show that $L_q(u_3 u_4) \subseteq L_p(u_1 u_4)$. In fact, given r applied to $x = \alpha u_1 u_2 \beta$ and $y = \sigma u_3 u_4 \gamma$, where $\delta(q_0, \alpha) = p$, that is $p \in Q_{u_1 u_2}$ and $\delta(q_0, \sigma) = q$, since $\alpha u_1 u_4 \gamma \in L(A)$, it must be that $\delta(q_0, \alpha u_1 u_4 \gamma) \in F$. But, since $\mathcal{A}$ is reduced, this fact implies that $\gamma \in L_p(u_1 u_4)$, as required. By symmetry of rule r, similarly as above we can show that $L_p(u_1 u_2) \subseteq L_q(u_3 u_2)$. ■

We relate the syntactic equivalence with the equivalence relation given in Definition 3.1 by using Lemma 3.2. We denote $[w]$ the (syntactic) congruence

class of $w \in A^*$, i.e., the set of words (syntactic) equivalent to w. Observe that, in general, $[x] \cdot [y] \subseteq [xy]$ [52].

Lemma 3.2 *[52] Given $w, w' \in A^*$, $w \equiv_L w'$ iff $\delta(q, w) = \delta(q, w')$ for every $q \in Q$.*

Proposition 3.1 *The following conditions are equivalent:*

(1) $w \equiv_L w'$

(2) $w \equiv w'$

Proof. It is easy to prove that (2) $\Rightarrow$ (1), by using definitions and Lemma 3.2.

Vice versa, let us prove that (1) $\Rightarrow$ (2). By contradiction, let us suppose that there exists $q' \in Q_{w'}$ such that $q' \notin Q_w$, i.e. $L_{q'}(w') \neq \emptyset$ and $L_{q'}(w) = \emptyset$. Thus $\delta(q', w)$ is defined, but $\delta(q', w')$ is not defined, a contradiction with the hypothesis that $w \equiv_L w'$. Now, let us prove that $L_p(w) = L_p(w')$, for any $p \in Q_w$. By contradiction, let us suppose that there exists $p \in Q_w$ such that $L_p(w) \neq L_p(w')$ and let $x \in A^*$ such that $x \in L_p(w)$ and $x \notin L_p(w')$, i.e., $\delta(p, wx) \in F$ and $\delta(p, w'x) \notin F$, a contradiction with Lemma 3.2. ■

Remark 3.1 In the next proposition we collect two properties of $\equiv$. Since they are known to be true when $\equiv$ is substituted with $\equiv_L$, Proposition 3.1 allows us to omit the proof. Observe that these properties may be proved without using Proposition 3.1 and syntactic congruence definition.

Proposition 3.2 *If $\alpha_1 \equiv \beta_1$ and $\alpha_2 \equiv \beta_2$, then $\delta(q, \alpha_1) = \delta(q, \beta_1)$, $\delta(q', \alpha_2) = \delta(q', \beta_2)$ and $\alpha_1\alpha_2 \equiv \beta_1\beta_2$.*

Now we will give another property of splicing t-uples, that may be used also to reduce the length of splicing rules. Moreover, the property explained below shows a deep connection between splicing operation and classes in a syntactic monoid. Indeed, Lemma 3.3 together with Lemma 3.1 allows us to pass from a splicing system to another one, preserving the consistence of the system.

Lemma 3.3 *Let* $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) \in C(\mathcal{A})$ *and* $\beta_i \equiv \alpha_i$, *for* $i \in \{1,2,3,4\}$. *Then* $(\beta_1, \beta_2, \beta_3, \beta_4) \in C(\mathcal{A})$.

Proof. By hypothesis, $L(\mathcal{A})$ is closed with respect to $r = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)$. We must prove that $L(\mathcal{A})$ is closed with respect to $r' = (\beta_1, \beta_2, \beta_3, \beta_4)$, i.e. for each pair $(q,p) \in Q_{\beta_1\beta_2} \times Q_{\beta_3\beta_4}$, then $L_q(\beta_1\beta_2) \subseteq L_p(\beta_3\beta_2)$ (and for each pair $(q',p') \in Q_{\beta_3\beta_4} \times Q_{\beta_1\beta_2}$, then $L_{q'}(\beta_3\beta_4) \subseteq L_{p'}(\beta_1\beta_4)$).

By contradiction, suppose that there exists $w' \in L_q(\beta_1\beta_2)$ such that $w' \notin L_p(\beta_3\beta_2)$, with $q \in Q_{\beta_1\beta_2}$ and $p \in Q_{\beta_3\beta_4}$. Since $w' \in L_q(\beta_1\beta_2)$, it means that $\delta(q, \beta_1\beta_2 w') \in F$. Let w be such that $\delta(q_0, w) = q$. Since $\beta_i \equiv \alpha_i$, for $i = 1,2,3,4$, $\alpha_1\alpha_2 \equiv \beta_1\beta_2$ (Lemma 3.2). So, by definition, $Q_{\alpha_1\alpha_2} = Q_{\beta_1\beta_2}$ and $L_{p'}(\alpha_1\alpha_2) = L_{p'}(\beta_1\beta_2)$, for each $p' \in Q_{\alpha_1\alpha_2}$. Thus $w' \in L_q(\beta_1\beta_2)$ implies $w' \in L_q(\alpha_1\alpha_2)$. Moreover, $\beta_3\beta_2 \equiv \alpha_3\alpha_2$, and so $Q_{\alpha_3\alpha_2} = Q_{\beta_3\beta_2}$ and $L_{p''}(\alpha_3\alpha_2) = L_{p''}(\beta_3\beta_2)$. Since $w' \notin L_p(\beta_3\beta_2)$, for $p \in Q_{\beta_3\beta_4}$, we have that $w' \notin L_p(\alpha_3\alpha_2)$, since $\beta_3\beta_2 \equiv \alpha_3\alpha_2$. But this is a contradiction with respect to the fact that $L(\mathcal{A})$ closed with respect to r, i.e. $w' \in L_q(\beta_1\beta_2) = L_q(\alpha_1\alpha_2)$ implies $w' \in L_p(\alpha_3\alpha_2) = L_p(\beta_3\beta_2)$. Similarly we obtain the second condition. ■

3.2 Other definitions on automata and representatives of classes

We give the definition of cycle slightly modified with respect to [12] and of a minimal element in a congruence class. A condition on this element will be given in order to show that some languages are generated by finite splicing.

Definition 3.3 (cycle) *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. each other and with respect to* q*). Then, if* c *is a cycle in* $\mathcal{A}$ *and* c *is not simple, there exist* $u_1, \ldots, u_{k+1} \in A^*$, *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 transitions 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* $\delta(q', c_i) = \delta(q', c_j)$ *then* $u_{i+1} \cdots u_j \neq 1$.

Remark 3.2 This definition of cycle allows us to state that $u_i \neq 1$ in $c = u_1 c_1^{p_1} u_2 \cdots u_k c_k^{p_k} u_{k+1}$. Roughly speaking, if a state q exists such that two closed path c, c' are defined on q, then no closed path crossing q containing both c and c' can be a cycle.

By $Occ(\alpha, q)$ we denote the set of all distinct states that occur in $t_{\alpha,q}$ for $q \in Q_\alpha$. Clearly $Occ(\alpha, q) = \emptyset$ if $q \notin Q_\alpha$.

Definition 3.4 (Minimum representative of a class) *Given the congruence class* $[\alpha]$*, a* minimum representative *of* $[\alpha]$ *is a sequence* $y \in [\alpha]$ *such that if* $|z| < |y|$*, for* $z \in [\alpha]$*, then* $Occ(y, q) \neq Occ(z, q)$*, for each* $q \in Q_y$*.*

Given a class, it is quite evident that the number of minimum representatives of a class is finite.

The following definitions allow us to define a finite set of sequences of a congruence class that can be used to relate splicing rules to congruence classes in a syntactic monoid.

Definition 3.5 (Representative of a sequence in a class) *Given the congruence class* $[\alpha]$ *and* $x \in [\alpha]$*, a* representative *of* x *in* $[\alpha]$ *is a shortest sequence* $y \in [\alpha]$ *such that* $Occ(y, q) = Occ(x, q)$*, for each* $q \in Q_x$*.*

Clearly, a representative of a sequence x is not necessarily a minimum representative in $[x]$.

Notation. Given a class $[x_i] \in \mathcal{M}(L)$, we denote by $R[x_i]$ the set of all representatives in the class $[x_i]$. Clearly, $R[x_i]$ is a finite language.

The above Definition 3.5 can be extended (see the following definition) in order to enlarge the set of representative sequences in a congruence class, still maintaining the finiteness of such set.

Definition 3.6 (Q-representative of a sequence in a class) *Let* $[x]$ *be a class of* $\mathcal{M}(L)$*. Then the* Q*-representative language associated to* $[x]$*, denoted* $R_Q[x]$ *is defined as:*

$R_Q[x] = \{y \in [x] :$ *given* $q \in Q_y$*, each state in* $t_{q,y}$ *occurs less than* $|Q|$*-times* $\}$.

Note that the language $R_Q([u_i])$ includes the class $R([u_i])$ as well as all minimum representatives of $[u_i]$.

Definition 3.7 (Accepting factorization) *Let $[x_1],\ldots[x_k]$ be classes of $\mathcal{M}(L)$. Then a sequence $s =< [x_1],\ldots,[x_k] >$ of elements of $\mathcal{M}(L)$ is an* accepting factorization *of $\mathcal{M}(L)$ if $\delta(q_0,x_1) = q_1$, $\delta(q_1,x_2) = q_2,\ldots,$ $\delta(q_{k-1},x_k) = q_k \in F$, for any $x_i \in [x_i]$, and if $\delta(q, x_i x_{i+1}\cdots x_j) = q$, then $[x_t] \neq [x_r]$, for $i \leq r, t \leq j$.*

Lemma 3.4 *Given the set $\mathcal{M}(L)$, then the set $seq(M)$ of all accepting factorizations of $\mathcal{M}(L)$ is finite.*

The following notion of partial accepting factorization is obtained from the above Definition 3.7 by allowing that in an accepting factorization a representative sequence (with respect to Definition 3.6) in a class of the factorization can substitute the class itself.

Definition 3.8 (Partial accepting factorization) *Let $s =< [x_1],\ldots[x_k] >$ be an accepting factorization and let us denote by a_i an element in $\alpha \in A^* \cup \mathcal{M}(L)$, such that $a_i = [x_i]$ or $a_i \in R_Q[x_i]$, that is a_i is either a congruence class of s or a representative sequence of a class in s. Then the sequence $s =< a_1,\ldots,a_k >$ is a* partial accepting factorization *of $\mathcal{M}(L)$.*

Since the set $R_Q[x]$ is finite as well as the set of all accepting factorizations, it is quite immediate to show that:

Lemma 3.5 *Given the set $\mathcal{M}(L)$, then the subset of $seq(M)$ of all partial accepting factorizations of $\mathcal{M}(L)$ is finite.*

Given $s =< a_1,\ldots,a_k >$ a partial accepting factorization, by $L^*(s)$ we denote the language $\{x'_1\cdots x'_{i-1}\cdots x'_k \mid x'_l \in [x_l] = a_l$ or $x'_l = a_l$, whenever $a_l \in A^*, 1 \leq l \leq k\}$ associated to s.

Observe that, by definition, $L^*(s) \subseteq L$, i.e. it consists of successful transition sequences for the automaton for L.

Definition 3.9 *A class in $\mathcal{M}(L)$ is called finite if it consists of a finite language, otherwise it is called infinite.*

Definition 3.10 (simple class) *A class* $[x] \in \mathcal{M}(L)$ *is called* simple *if it is finite or whenever it is infinite then there exists a representative* $y \in R[x]$ *that consists of a cycle in* $\mathcal{A}$ *for* $\mathcal{A}$ *the minimum finite state automaton for* L.

Roughly speaking, if a class in not finite and is a simple class, we have that this class can contain a cycle, i.e., it is the label of a transition in which no more one closed path is crossed.

Lemma 3.6 (property A) *Let* $[x]$ *be a simple class, such that* $|Q_x| = 1$. *Then for every* $y, z \in [x]$, *then* $yz \in [x]$.

Proof. By definition of simple class, there is an element x in $[x]$ such that $\delta(q, x) = q$, for $q \in Q_x$. But, then given y an element in $[x]$, $\delta(q, y) = q$, which implies that $yz \in [x]$, for $z \in [x]$. ■

3.3 Regular splicing languages

In this section we state a condition on congruence classes that allows us to state that the language associated to a partial accepting factorization is generated by splicing. Note that a regular language can be seen as the finite union of languages associated to a finite set of partial accepting factorizations. In order to prove the main result of the chapter, we need to introduce the notion of reverse automaton and illustrate some properties of the reverse operation applied to sequences.

We recall that the *reverse* $Rev : A^* \to A^*$ may be inductively define as follows: $Rev(1) = 1$, $Rev(a) = a$, for any $a \in A$, $Rev(a_1 \cdots a_k) = Rev(a_2 \cdots a_k)a_1$.

Let us consider a rule $r \in R$ for a splicing system $S = (I, R)$. Clearly r may be considered a word, then the reverse of r may be also defined. Indeed, given $r = u_1|u_2\$u_3|u_4$, then $Rev(r) = Rev(u_4)|Rev(u_3)\$Rev(u_2)|Rev(u_1)$. Moreover, observe that $u_1|u_2\$u_3|u_4$ and $u_3|u_4\$u_1|u_2$ have the same behaviour. Thus $Rev(r) = Rev(u_2)|Rev(u_1)\$Rev(u_4)|Rev(u_3)$. We can prove the following proposition, that is reported here for the sake of completeness.

Proposition 3.3 *Given a splicing system $S = (I_1, R_1)$, then*

$$L(Rev(I_1), Rev(R_1)) = Rev(L(I_1, R_1))$$

.

In the next we will show how to generate $L^*(s)$ when the following condition (on a class of s) is verified.

Definition 3.11 (0-generated) *A partial accepting factorization s with $s =< a_1, a_2, \ldots, a_k] >$ is 0-generated with respect to $\mathcal{A}$ if and only if there exists $a_i a_{i+1} \ldots a_j$ such that $a_i a_{i+1} \ldots a_j = [x_i]\alpha$, or $a_i a_{i+1} \ldots a_j = \alpha[x_j]$, with $\alpha \in A^*$, $[x_i]$, $[x_j]$ simple classes, and $|Q_{a_i a_{i+1} \ldots a_j}| = 1$.*

Observe that, given a regular language, 0-generated property may be decided, since, the syntactic monoid is finite and, by Lemma 3.4 the set of all partial accepting factorizations is finite.

Definition 3.12 (left/right-marker) *Given a sequence s that is 0-generated, then we call* left-marker *the sequence $< [x_i], \alpha >$ for which $|Q_{x_i \alpha}| = 1$, and* right-marker *the sequence $< \alpha, [x_j] >$ for which $|Q_{\alpha x_j}| = 1$.*

We now prove our main result. Let $\mathcal{A}_s$ be the deterministic finite state automaton recognizing $L^*(s)$. Clearly, a right or left marker denotes the language $\alpha\ [x_j]$ or $[x_i]\alpha$.

Theorem 3.1 *Given a partial accepting factorization s of $\mathcal{M}(L)$ that is 0-generated then there exists a splicing system $S_s = (I, R)$ such that $L^*(s) = L(S_s)$.*

Proof. Let $\mathcal{A}_s$ be the deterministic finite state automaton recognizing $L^*(s)$. Let $\mathcal{A}_r = \{A, Q, \delta_r, q_r, F_r\}$ be the minimal finite state automaton recognizing $Rev(L^*(s))$. We construct $S_s = (I, R)$ where $I = I_r \cup I_l$ and $R = R_r \cup R_l$ are defined as follows. We firstly give the system $S_r = (I_r, R_r)$.

It is not restrictive to assume that s is of the form $s =< [x_1], [x_2], \ldots, [x_i], \alpha, [x_j], \ldots, [x_k] >$, $\alpha \in A^*$, i.e. α is the only representative that substitutes the congruence classes $[x_{i-1}] \ldots [x_{j-1}]$.

If s is 0-generated, then define the language $L_1 = \{x_1\, x_2 \cdots x_i \alpha[x_j][x_{j+1}] \cdots [x_k] \mid \ x_l \in [x_l],\ 1 \leq l \leq i$, such that if $Rev(x_1 \cdots x_i) = uc^t\, v$, with $\delta_r(q,c) = q$, and $\delta_r(q_r, u) = q$, then c is a cycle in $\mathcal{A}_r$ and $t \leq 1\}$. Observe that the condition $Rev(x_1 \cdots x_i) = uc^t v$ must be checked on the left of the simple class, since the cycles on the left of $[x_i]$ will be enlarged by the Reverse and so they must be cycles in the reversed automaton.

Then, define $I_r = \{w \in L_1 \mid$ if $w = w_1 \alpha u c^t v$, $uc^t v \in [x_j][x_{j+1}] \cdots [x_k]$, with $\delta(q, c^t) = q$, for $q = \delta(q_0, w_1 \alpha u)$, then c is a cycle in $\mathcal{A}_s$, and $t \leq 1\}$. Clearly, I_r is finite (we can use the same argument of the proof of Lemma 7.1 in Part II).

Given $\alpha[x_j]$ (resp. or $[x_i]\alpha$) the right-marker (resp. the left-marker) of s, we define the *representative marker language* $M(s)$ *for* s as follows:

$M(s) = \{y : y \in \alpha[x_j]$, for $\alpha[x_j]$ right-marker in s, (resp. $y \in [x_i]\alpha$ for $[x_i]\alpha$ left-marker in s) such that if $y = uc^t v$ with $\delta(q,c), q \in Q_c(\mathcal{A}_s)$, (resp. $Rev(y) = uc^t v$ with $\delta_r(q,c), q \in Q_c(\mathcal{A}_r)$) then c is a cycle in $\mathcal{A}_s$ (resp. in $\mathcal{A}_r$) and $t \leq 1\}$. In the rest of proof, assume that $y_1 \in [x_i]$ or $y_1 \in [x_j]$.

By the above definition of the representative marker language (which is finite) the following property holds: for each word $w \in I_r$ we have that $w = w_1 y w_2$, $y \in M(s)$. Indeed, it must be that $w = w_1 y w_2$, with $y \in \alpha[x_j]$ if s has a right-marker or $y \in [x_i]\alpha$ if s has a left-marker. Thus, by definition of I_r as consisting of strings in L_1, the required property is immediate.

Then define R_r as follows: $R_r = \cup_{y \in M(s)} R_y$ where

(1) for every cycle c_j in $\mathcal{A}_s$ that follows y, i.e., $w = w_1 y w' c_j w''$, $w \in I_r$,

$$r_{c_j} = \quad yw' \mid 1 \ \$ \ yw'c_j \mid 1$$

(2) if s has a right-marker, for every cycle c inside y, i.e. given $y = \alpha ucv = \alpha y_1$ (we recall that α does not contain any cycle) then,

$$r_c = \quad yu \mid 1 \ \$ \ yuc \mid 1,$$

and

$$r_y = \quad \alpha y_1 \mid 1 \ \$ \ \alpha \mid y_1,$$

r_{c_j}, r_c and $r_y \in R_y$.
Since I_r is finite, there is a finite number of cycles at the right of y and so also R_r is finite.

Now, let us construct $S_l = (I_l, R_l)$. Consider the language $L_2 = \{[x_1][x_2]\cdots[x_i]$ $\alpha x_j x_{j+1}\cdots x_k \mid , x_l \in [x_l],\ j \leq l \leq k$, and if $x_j\cdots x_k = uc^tv$, with $\delta(q,c) = q = \delta(q_0, x_1x_2\cdots x_i\alpha u)$, then c is a cycle in $\mathcal{A}_s$ and $t \leq 1\}$.

Given $s_1 = Rev(s)$, then s_1 is 0-generated, as $|Q_y(\mathcal{A}_s)| = 1$ implies that $|Q_{Rev(y)}(\mathcal{A}_r)| = 1$ (see property B in Lemma 3.8).

Then construct as above the system $S_r = (I_r, R_r)$ for the automaton $\mathcal{A}_r$, which we denote as $S_1 = (I_1, R_1)$, where note that for the construction of I_1, L_1 is given by $L_l = Rev(L_2) \subseteq Rev(L^*(s))$.
Thus, pose

$$Rev(I_1) = I_l, Rev(R_1) = R_l, S_l = (I_l, R_l)$$

.
Let us show that $L^*(s) = L(S_l \cup S_r) = L(S_s)$. By using Lemma 3.1, we can prove that $L(S_l \cup S_r) \subseteq L^*(s)$. In fact, we can prove by Lemma 3.8 that $R_r \cup R_l \in C(\mathcal{A}_s)$; thus since $I_l \cup I_r \subseteq L^*(s)$, clearly Lemma 3.1 holds.

Let us now show that $L^*(s) \subseteq L(S_l \cup S_r)$ by induction on the length of a string $w \in L^*(s)$. Clearly, this fact holds for $w \in I_l \cup I_r$, thus assume that $w \in L^*(s) - (I_l \cup I_r)$. Then it must be that

(*) $w = w_1\alpha u_1 {c_2}^{t_2} u_2$, or $w = v_1 {c_1}^{t_1} v_2 \alpha w_2$,

where the following cases are given:

(i) c_1 or c_2 is a cycle, and $t_1, t_2 > 1$,

(ii) c_1 or c_2 is not a cycle, but $\delta(q, c_1) = q$ or $\delta(q_1, c_2) = q_1$.

Let u_1c_2 be the shortest string for which condition (*) is verified (whenever c_1v_2 is such shortest string the proof is similar).

Case (i) Assume that c_2 is a cycle. We must consider two cases: $w = w_1\alpha u_1 {c_2}^{t_2} u_2$ where $u_1 = y_1u_1'$, that is c_2 follows y, or c_2 is inside y whenever s has a right-marker $\alpha[x_j]$ and $u_1c_2u_2' \in [x_j]$. We first consider the first case and prove by induction that the string $w = w_1\alpha y_1 u_1' {c_2}^{t_2} u_2$ is in $L(S_s)$, By induction, the string $w' = w_1\alpha y_1 u_1' {c_2}^{t_2-1} u_2 \in L(S_s)$. We claim that rule

r_{c_2} is in R_r, $r_{c_2} = \quad \alpha y_1 u_1' \mid 1 \ \$ \ \alpha y_1 u_1' c_2 \mid 1$, otherwise we contradict the minimality of $u_1 c_2$. Applying rule r_{c_2} to w' we can obtain w.

Now, we consider the second case and prove that given $w = w_1 \alpha u_1 {c_2}^{t_2} u_2$, where c_2 is inside y, $w \in L(S_s)$. Since $y_1 \in [x_j]$, where $[x_j]$ is a simple class, by Property A, then $y_1 u_1 {c_2}^{t_2} u_2' \in [x_j]$. By the first case it follows that $w' = w_1 \alpha y_1 u_1 {c_2}^{t_2} u_2$ is in $L(S_s)$. Since $r_y = \quad \alpha \mid y_1 \ \$ \ \alpha y_1 \mid 1 \in R_r$, it is immediate to obtain w from w'.

Case (ii) Assume that c_2 is not a cycle. Then we claim that $c_2 = c'c''$ such that $\delta(q, c') = \delta(q, c'')$ for $q \in Q_{c_2}$, $q = \delta(q_0, w_1 \alpha u_1)$ and q does not occur inside the transition $t_{q,c'}$. In fact otherwise $c_2 = vcz$ where c is not a cycle by $\delta(q, v) = q_1$, and $\delta(q_1, c) = q_1$. Then $w = w_1 \alpha u_1 (vcz) {c_2}^{t_2 - 1} u_2$, but $|u_1 vc| < |u_1 c_2|$, a contradiction with the minimality of $|u_1 c_2|$. Again, we consider the case that $w = w_1 \alpha y_1 u_1 {c_2}^{t_2} u_2$, that is c_2 follows y_1. In fact, by induction the string $w' = w_1 \alpha y_1 u_1 c''(c_2^{t-1}) u_2 \in L(S_s)$ and also $\bar{w} = w_1 \alpha y_1 u_1 c'(c_2^{t-1}) u_2 \in L(S_s)$. Since the rule $r_{c'} = \quad \alpha y_1 u_1 \mid 1 \ \$ \ \alpha y_1 u_1 c' \mid 1$ is in R_r, by applying it to w' and $\bar{w}$ it is immediate to obtain w.

Now, let us prove that $w = w_1 \alpha u_1 {c_2}^{t_2} u_2$ is also in $L(S_s)$ in the case c_2 is inside y_1, that is whenever s has a right-marker $\alpha[x_j]$ and $u_1 c_2^{t_2} u_2' \in [x_j]$. By the fact that $y_1 \in [x_j]$, where $[x_j]$ is a simple class, it follows (see Property A, 3.6) that also $w' = w_1 \alpha y_1 u_1 {c_2}^{t_2} u_2$ is in $L^*(s)$, and hence $w' \in L(S_s)$, by the above proof. Since $r_y = \quad \alpha \mid y_1 \ \$ \ \alpha y_1 \mid 1 \in R_r$, it is immediate to obtain w from w'. ■

By using the determinism of $\mathcal{A}_s$, the proof of the following lemma is quite immediate.

Lemma 3.7 *Let $\mathcal{A}_s$ be a minimal deterministic automaton and $\mathcal{A}_r$ be the minimal deterministic automaton for $Rev(L(\mathcal{A}_s))$. Let $y \in A^*$ such that $|Q_y(\mathcal{A}_s)| = 1$, then $|Q_{Rev(y)}(\mathcal{A}_r)| = 1$.*

Observe that the use of the Reverse automaton to construct rules that are used to enlarge cycles to the left of the marker is essential in order to obtain a set of rules R_l such that $R_l \in C(\mathcal{A}_s)$. This property allows to apply Lemma 3.1 in the proof of Theorem 3.1.

Lemma 3.8 *Let $R = R_r \cup R_l$ be the set of rules constructed in Theorem 3.1, then $R \in C(\mathcal{A}_s)$.*

Proof. Let us first show that $R_r \in C(\mathcal{A}_s)$, where $\mathcal{A}_s$ is the minimal finite state automaton recognizing $L^*(s)$. By using the definition 3.2, we have that, for every $p \in Q_{u_1u_2}$ and $q \in Q_{u_3u_4}$, $L_p(u_1u_2) \subseteq L_q(u_3u_2)$. Now, given the rule in case (1), then $u_1u_2 = yw'$, while $u_3u_2 = u_3u_4 = yw'c$. Since $|Q_y| = 1$, it must be that $p = q$. Now, $\delta(p, yw') = \delta(p, yw'c) = q_1$, as by construction c is a cycle, that is $\delta(q_1, c) = q_1$. Thus, $L_p(u_1u_2) = L_p(u_3u_2)$. Moreover $L_p(u_1u_2) = L_p(u_1u_4)$, thus also proving that $L_p(u_3u_4) \subseteq L_q(u_1u_4)$, as required. Along the same lines, we can prove that the rules in cases (2) are in $C(\mathcal{A}_s)$ and that $Rev(R_1) \in C(\mathcal{A}_r)$, as in fact the system S_1 is constructed as the system S_r.

Given the sites u_1u_2 and u_3u_4 of a rule in R_1, it is $|Q_{u_1u_2}| = 1$ as well as $|Q_{u_3u_4}| = 1$. But, since $\mathcal{A}_r$ is deterministic, then also $|Q_{Rev(u_1u_2)}| = 1$ and $|Q_{Rev(u_3u_4)}| = 1$ with respect to the automaton $\mathcal{A}_s$ (property B). This fact allows to show that $Rev(R_1) \in C(\mathcal{A}_s)$. ■

Final remark. We point out that this chapter is part of a work in progress. Thus, some definitions are given here in such a way they may be used also for the complete characterization of $H(Fin, Fin) \cap Reg$.

Part II

Circular splicing

Circular splicing has been defined in different ways, but only for single stranded DNA molecules. Let us focus on the behaviour of circular splicing. A restriction enzyme recognizes a pattern inside a circular DNA molecule and then it opens the molecule inside the pattern, i.e., it linearizes a circular DNA molecule at a specific point (linearization). Now two different processes may happen: either ligase enzyme pastes together the (linear) fragments and circularizes the newly obtained molecule (Head and Paun's definition), or before the pasting action, ligase enzyme substitutes the recognized pattern (Pixton's definition). Our results and investigations deal with all these definitions.

As in the linear case, we suppose to manage a nonbounded number of copies of a molecule, or, in other words, before performing splicing, we preliminary copy the terms of splicing.

Moreover, we will only deal with finite splicing systems, i.e., the (circular) initial language and the set of rules are both finite sets. Thus, if a language is "generated", we mean that it is generated by a finite splicing system. Moreover, even if the literature consider hypotheses on splicing systems, we do not take them into account.

In Nature circular splicing occurs in a recombinant mechanism (transposition) between bacterial chromosomes and F plasmids in *E.coli*. In [74] we read that a molecular biological observation reports that *lambda phage* DNA can easily change its form from linear to circular and vice versa. When *lambda phage* DNA is transposed in a bacterial DNA like *E.coli* DNA, it forms a circular that is incorporated into a host DNA by splicing (crossing over). Moreover, the replication process is possible only if DNA has a circular form in the host cell. In the Introduction we have already mentioned the possibility of using circular DNA to perform Adleman's experiment, since plasmids autonomously replicate without errors or exponential weight increase, which are weak points in Adleman's approach [1]. Indeed, in [18] a laboratory experiment implementing a new model for DNA biomolecular computing, previously introduced in [41], involves circular and linear molecules with insertions and deletions operations. These systems are capable of universal computation.

As in the linear case, we have investigated the relationship between finite circular splicing and circular regular languages. This aim leads us to approach circular languages, or, analogously, as we will see later, with formal languages closed under conjugacy relation, a well-known equivalence relation between words (Chapter 4). All the definitions about circular splicing (Chapter 5.1) are followed by chapters containing our results toward a characterization of circular regular languages generated by finite splicing. For the sake of clarity, we have preferred to split in different chapters the presentation of the classes of languages proved to be generated.

A comparison among the three finite splicing systems is provided in Chapter 6. We focus our attention firstly on Paun's definition, we define a class of regular languages (*star languages*) and we prove that for a subclass of them the corresponding circular languages are Paun generated. Restricting ourselves to regular languages on *one-letter alphabet*, we prove that we can decide whether such a regular language is Paun generated, by using a characterization we have found. We extend this characterization, by considering *uniform languages*. *Cyclic* and *weak cyclic languages*, which will be introduced in Chapter 9, show that this result does not hold when we increase the size of alphabets, even if we restrict ourselves to regular languages. We conclude our studies on (circular) splicing with the definition of a class of circular regular languages which we prove to be Pixton generated. As a conclusion, no regular circular language generated by finite splicing was found not to belong to this class.

Chapter 4

Definitions on circular languages

We will give here all the necessary definitions to treat circular splicing. We start with the definition of a circular word. As we have already said in the introduction of Part II, circular words have already been intensively examined in Formal Language Theory and Combinatorics on Words (see, for example, Lyndon words definition [51, 3]). Intuitively, a circular word over A is a sequence of symbols of A with the understanding that the first symbol follows the last symbol. Thus, $a_1 \cdots a_n$ is the same as $a_k a_{k+1} \cdots a_1 \cdots a_{k-1}$, for $1 < k < n$, $a_1, \ldots, a_n \in A$. Precisely, we define the *conjugacy relation* $\sim$ on words in A^* by $xy \sim yx$, for $x, y \in A^*$. It is easy to prove that $\sim$ is an equivalence relation. Thus we have the following definition.

Definition 4.1 (Circular word) *[51] Given word $w \in A^*$, $^\sim w$ is the equivalence class of w with respect to conjugacy relation, or* circular word. *The set of all circular words over A, i.e., the quotient of A^* with respect to $\sim$, will be denoted as $^\sim A^*$.*

Notations. Clearly, $^\sim 1 = 1$. Moreover, for every $a \in A$, $|^\sim w|$ and $|^\sim w|_a$ will be defined respectively as $|w|$ and $|w|_a$, for any representative w of $^\sim w$. Furthermore, when the context does not make it ambiguous, we will use the notation w for a circular word $^\sim w$.

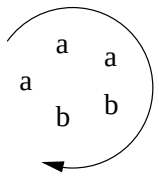

Figure 4.1: *A graphical representation of the circular word* $\sim abbaa$.

We use the notion of factors of a circular word: for any $x \in A^*$, we say that x is a factor of a circular word w when a representative w' of w exists such that x is a factor of w'.

Definition 4.2 (Circularization and Linearization) *Given* $L \subseteq A^*$, *the set* $\sim L = \{\sim w \mid w \in L\}$ *is the* circularization *of* L, *i.e., the set of all circular words corresponding to elements of* L. *Given a circular language* $C \subseteq \sim A^*$, *every language* L *such that* $\sim L = C$ *is called a* linearization *of* C. *The set of all strings in* A^* *corresponding to elements of* C, *is the* full linearization *of* C, *written* $Lin(C)$.

Example 4.1 For $w = abbaa$, we have $Lin(\sim abbaa) = \{abbaa, bbaaa, baaab, aaabb, aabba\}$. Every subset of $Lin(\sim abbaa)$ is a linearization of $\sim abbaa$. The circularization of $\{abbaa, bbaaa\}$ is $\sim abbaa$.

Circular splicing deals with circular strings and circular languages and as a result, with formal languages which are full linearizations of circular languages. The following proposition states an obvious property of full linearizations.

Proposition 4.1 *[12] A language* L *is the full linearization* $L = Lin(C)$ *of a circular language* C *if and only if* L *is closed under the conjugacy relation, i.e., for all* $l, l' \in A^*$ *if* $l \in L$ *and* $l' \in \sim l$, *we have* $l' \in L$.

Proof. Let $L = Lin(C)$ be the full linearization of a circular language C. Let $l \in L$. Then, $\sim l \in C$ and, by using the definition of full linearization, for every $l' \in \sim l$, we have $l' \in L$. Then, L is closed under the conjugacy relation.

Conversely, if L is closed under the conjugacy relation, then $Lin(^{\sim}L) = L$ holds. ■

In the next, we will consider languages closed under the conjugacy relation and the action of circular splicing over their circularization.

Some closure properties of conjugacy relations are evident. Indeed, given $L, L' \in A^*$, $^{\sim}(L \cup L') = {}^{\sim}L \cup {}^{\sim}L'$ and $^{\sim}(L \cap L') = {}^{\sim}L \cap {}^{\sim}L'$. This means that the application associating to each language its circularization is a morphism with respect to union and intersection.

Notation. Given a family FA in the Chomsky hierarchy, $FA^{\sim}$ consists of all those circular languages C which have some linearization in FA.

Observe that, given a regular language $L \subseteq A^*$, it is classically known that the set $CYCLE(L) = \{w' \in A^* \mid \exists\ w \in L.\ w' \sim w\}$ (cyclic permutation of L) is also regular ([40], p. 72). Thus, we have the following result.

Theorem 4.1 *[40] $C \in Reg^{\sim}$ if and only if its full linearization is regular.*

The same holds for $CF^{\sim}$ and $RE^{\sim}$ (since Reg, CF, RE are closed under cyclic permutation). We mainly deal with $Reg^{\sim}$ and with treat circular regular languages as well as regular languages closed under conjugacy relation.

Final remark. Up to now, no characterization of $Reg^{\sim}$ is known. Indeed, conjugacy equivalence relation is not a congruence and so no concatenation may be defined on circular words. Moreover, no syntactic monoid is defined and so circular splicing may be a new approach to face the characterization of $Reg^{\sim}$. As we will see in the next chapters, circular (Head or Paun) splicing may be seen as a suitable concatenation (or shuffle of suitable linearizations) of circular words.

Chapter 5

Circular splicing history: models and main results

The splicing operation in the circular case deals with biological phenomena which are different from the ones modelled by the linear case (Section 1.2). As well as for linear splicing, three definitions of circular splicing are known and given in Section 5.1. Additional hypotheses that can be added or not to R lead to new models as well as others, presented in Section 5.2. An overview of our contributions to circular splicing is collected in Section 5.3.

5.1 Circular splicing operation

We recall that Head's and Paun's definitions of circular splicing model a biological phenomenon different from Pixton's one. Indeed, in the former, two patterns are recognized in two circular words which are cutted (linearized), pasted and the obtained string is circularized. In the latter, a substitution of the recognized patterns, before putting the fragments together, is allowed.

Formally, the circular splicing opens two circular strings and pastes them together (eventually substituting the rcognized patterns), obtaining a circular string, under some conditions represented into a (splicing) rule.

`Head's definition [71]`. A *Head circular splicing system* is a 4-tuple $SC_H = (A, I, T, P)$, 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 produces $^\sim hpxvkuxq$.

`Paun's definition [64, 38].` A *Paun circular splicing system* is a 3-tuple $SC_{PA} = (A, I, R)$, 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 [66].` A *Pixton circular splicing system* is a 3-tuple $SC_{PI} = (A, I, R)$ 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 $^\sim\epsilon\beta\epsilon'\beta'$.

We point out that we restrict ourselves to circular splicing systems with a (circular) finite or regular initial language. Furthermore, it goes without saying that T and R are finite sets in the three definitions of circular splicing given above.

Remark 5.1 We notice that in a Pixton circular splicing system, for $r, \overline{r} \in R$, we can have $r = \overline{r}$. Furthermore, for given rules $r_i = (\alpha, \alpha'; \beta_i) \in R$, $1 \leq i \leq v$, a same rule $\overline{r}_i = \overline{r} = (\alpha', \alpha; \beta') \in R$ can be associated with all the r_i's.

We must note that in the original definition of circular splicing language given by Paun in [38], rules in R can be used in two different ways: one way has been described above, the other, called *self-splicing*, will be defined in Section 5.2. In this paper, in order to make the three definitions uniform, we have preferred to delete self-splicing from Paun's definition.

We now give the definition of circular splicing languages, where $(w', w'') \vdash_r z$ denote the fact that z is spliced by w', w, by using a rule r, with respect to

one of the three definitions. We explicitly note that for Pixton's systems, the splicing operation is the combined action of a pair of rules, r and $\bar{r}$ and we will use the notation $\vdash_{r,\bar{r}}$ to make this behaviour evident.

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

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

Thus, we define

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

Definition 5.1 (Circular splicing generated language) *Given a splicing system SC_X, with $X \in \{H, PA, PI\}$, the circular language $C(SC_X) = \sigma^*(I)$ is the language generated by the system SC_X, $I \subseteq {}^{\sim}A^*$ being the initial language in 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 $C = C(SC_X)$.

As in the linear case, we denote

$$C(F_1, F_2) = \{\sigma^*(I) \mid C_{PA} = (A, I, R) \text{ with } I \in F_1, R \in F_2\}$$

where F_1, F_2 are families of circular languages in Chomsky hierarchy. Clearly, the same definition may be given for the other two definitions of circular splicing systems.

Closure properties of circular splicing languages are provided in the next chapters (Theorem 5.2). Furthermore, we have proven that circular splicing is not closed under union (Proposition 6.4).

It is already 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 it has no regular linearization), but $\sim a^n b^n$ is C_H generated with a finite initial circular language [71]. (As a matter of fact, in [71] the authors show that $\sim (ca)^n (cb)^n$ is C_H generated.) On the other hand, we will see examples of regular circular languages that cannot be generated by using C_{PA} splicing systems with a finite initial circular language (Examples 6.1 and 6.3). So far, we have not yet discovered whether $C(Fin, Fin)$ contains any context-sensitive or recursive enumerable language which is not context-free. We believe that $C(Fin, Fin) \subseteq NP$. Nevertheless, in the rest of the thesis, we restrict our investigation to the following problem.

Problem 5.1 *Characterize* $Reg^{\sim} \cap C(Fin, Fin)$.

5.2 Circular splicing hystory: other models

Relatively few works on circular splicing systems and their languages have been published and they use only additional hypotheses, reported below. This is the reason why the state of the art reported in this work about "tout court" circular splicing (namely, the Problem 5.1) is so short.

In [33], Tom Head reviewed a previous work on splicing systems and strengthened the significance of studying circular splicing and mixed splicing (both linear and circular words are allowed). In [71], the authors proposed several splicing operations involving circular strings, some of which acting either only on circular strings (Head's definition of splicing operation), or on both linear and circular strings (mixed splicing), or on one circular word producing two circular words starting with a circular one (self-splicing).

In [71, 66] it was underlined 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.

Example 5.1 The regular language $(aa)^*$ is not generated by linear splicing (Example 1.1), but the corresponding circular language $\sim (aa)^*$ is C_H

generated (Example 6.1). Moreover, the regular language $(ab)^+ \cup (ba)^+$ is not generated by linear splicing [55], but the corresponding circular language $^\sim((ab)^+ \cup (ba)^+)$ is C_{PA} generated (Example 7.4).

The following is one of the first result in circular splicing.

Theorem 5.1 *[71] Circular languages generated by null context rules, whose crossing has length* 1*, are regular.*

In the papers which have already been published on this topic, additional hypotheses are often considered for Paun's or Pixton's definitions [38, 66, 64, 65]. Precisely:

Hypothesis 1. R is a *symmetric* scheme, 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}) there is the rule $\overline{r} = u_3|u_4\$u_1|u_2$ (resp. $\overline{r} = (\alpha', \alpha; \beta')$).

Remark 5.2 Any set R of rules in a splicing system SC_{PI} is implicitly symmetric (by definition). In [38] it is explicitely observed the simmetric behaviour of Paun's rules. Thus, any set R of rules in a splicing system SC_{PA} is implicitly symmetric.

Hypothesis 2. 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}) there are the rules $u_1|u_2\$u_1|u_2$ and $u_3|u_4\$u_3|u_4$ (resp. $(\alpha, \alpha; \alpha)$ and $(\alpha', \alpha'; \alpha')$).

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 the rules $(\alpha, \alpha'; \beta)$, $(\alpha', \alpha; \beta')$).

In Problem 5.2, which generalizes Problem 5.1, we also take into account these additional hypotheses.

Problem 5.2 *Can we characterize $FA^\sim \cap C(Fin, Fin)$ (resp. $FA^\sim \cap C(Reg, Fin)$) for each class FA of languages in the Chomsky hierarchy, for every definition of $C(Fin, Fin)$ (resp. $C(Reg, Fin)$) and for every possible combination of the three hypotheses given above?*

Pixton proved closure properties of circular splicing by using two different definitions of splicing (Paun's and Pixton's definitions) [64, 65]. In particular, Pixton proved that a circular regular language and a finite set of rules generate a regular language if the set of rules satisfies some additional hypotheses.

Theorem 5.2 *[66] Let $SC_{PI} = (A, I, R)$ be a circular splicing system with I a circular regular language and R (finite) reflexive and symmetric. Then $C = C(SC_{PI})$ is regular.*

A similar closure property of splicing has been proved under a more general hypothesis on I (full AFL) and for Paun systems [38, 64, 65], which generalizes a similar theorem proved for linear splicing (Theorem 1.5).

A characterization of the circular language accepted by an automaton is given together with examples of non regularity of circular splicing languages, leading on the hypotheses $1, 2, 3$. Regularity is guarenteed if reflexivity and self-splicing are incorporeted (this is the reason why, in some papers, the definition of circular splicing language include the closure under self-splicing operation). Regularity of Theorem 5.2 is shown by construction of an automaton accepting the circular splicing language. A simpler proof is in [65].

In the same paper the author treats mixed splicing, proving the regularity of languages generated starting with a regular mixed (initial) language by iterating a reflexive and simmetric set of rules.

In [74] a new type of circular (extended) H-system (CH systems) is proposed and the universality of this model is provided. Here, linear and circular strings are involved, without multeplicity, and with I, R both finite sets.

Starting from [74], CH systems are considered in [70] as components in a *Communicated Distributed H-system*, i.e., the circular splicing counterpart of the parallel communicating grammar systems. In this model, the components work by splicing and communicate by sending each other linear strings which pass certain filters, specified in advance. The characterization of RE languages by distributed circular systems amounts to looking for reducing the number of rules in each component the number of components itself.

Learning algorithm for circular molecules have been very recently presented [72].

5.3 Circular results: overview

Classical results in Formal Language Theory and combinatorial tools have been of great use in obtaining our results. The research about circular splicing is less developed than the linear case, probably because of the circularity of the words. Nevertheless, 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.

We have restricted ourselves to the relationship between circular regular languages and languages generated by finite circular splicing systems (in short, generated languages). We have provided some results towards a characterization of the intersection between these two classes.

Chapter 6 introduces the general problem we are attacking. In the same section we explicitly observe that the computational power of the three circular splicing systems is already known to be different. We also present a circular regular language which is neither Paun nor Pixton (finite) generated, proving that not all circular regular languages can be generated by (finite) circular splicing.

In Chapter 7 we consider a class of regular languages X^*, called *star languages*, which are closed under conjugacy relation and with X being a regular language. Using automata theory and combinatorial techniques on words, for two subclasses of star languages the corresponding circular languages are (Paun) circular splicing languages. The first class is defined as the family of star languages X^* with X being a finite set. The second class, whose languages are called *fingerprint closed* star languages, contains all star languages X^* with X^* being a free monoid (as a matter of fact we do not yet know of any examples of fingerprint closed star languages which do not satisfy this hypothesis). We work on X^* since some properties of this kind of languages may be interpretated in terms of splicing.

We prove that each (Paun) circular splicing language L over a one-letter

alphabet has the form $L = X^+ \cup Y$, with X, Y finite sets satisfying particular hypotheses. We prove that we can decide whether a regular language on one-letter alphabet is generated by a finite Paun circular splicing system. In Section 8 we generalize this result, characterizing circular regular languages $L = A^J = \bigcup_{j \in J} A^j$, J a subset of natural numbers $\boldsymbol{N}$.

Furthermore, *Cyclic* and *weak cyclic languages*, have been introduced. These Paun generated languages show that the characterization above does not hold when we increase the size of alphabets, even if we restrict ourselves to regular languages.

Finally, we introduce a class of circular regular languages which are generated by circular Pixton splicing, finding connection with subclasses of regular languages.

Our investigation on circular splicing started with [13]. The results presented here are partially contained in [11] and submitted to an international journal [12]. Answering some questions posed in [12], we have also presented developments on circular splicing at WORDS01 [9], whose extended version [10] is completely reported here.

Chapter 6

Computational power of the three finite splicing systems

In this chapter we discuss the computational power of (finite) circular splicing. We recall that we are looking for a characterization of the regular circular languages which are generated by a finite circular splicing system (Problem 5.1).

6.1 Circular regular splicing languages and inclusion results

Before going on, we specify which of the three above-mentioned definitions will be referred to in our investigation. This observation naturally leads to the question posed in Problem 6.1 which has a negative answer, as Examples 6.1 and 6.2 show. Proposition 6.1 is an attempt to give a more precise answer showing that the computational power of circular splicing systems increases when we substitute Head systems with Paun systems and Paun systems with Pixton systems. Nevertheless, we mainly deal with finite Paun systems.

Below we formalize the problem of comparing the computational power of the three definitions of circular splicing given in Chapter 5.

Problem 6.1 *Given $C = C(SC_X)$, with $X \in \{H, PA, PI\}$, does SC_Y exist, with $Y \neq X$, $Y \in \{H, PA, PI\}$, such that $C = C(SC_Y)$?*

Proposition 6.1 gives a partial answer to Problem 6.1. This proposition and Example 6.1 show that Pixton systems have a greater computational power than the other two.

Proposition 6.1 *If $C \subseteq^{\sim} A^*$ is C_H generated, then C is C_{PA} generated. Moreover, if $C \subseteq^{\sim} A^*$ is C_{PA} generated, then C is C_{PI} generated.*

Proof. Given $SC_H = (A, I, T, P)$ and $C = C(SC_H)$, let us consider the system $SC_{PA} = (A, I, R)$, where $R = \{r \mid r = px|q\$ux|v,\ (p, x, q), (u, x, v) \in T\}$, and $C' = C(SC_{PA})$. We prove that $C = C'$, by induction on $|w|$, for $w \in C \cup C'$. Indeed, if $w \in I$, then obviously $w \in C \cap C'$. On the other hand $w \in C \setminus I$ if and only if there exist $w' =^{\sim} hpxq, w'' =^{\sim} kuxv \in C$, with $(p, x, q), (u, x, v) \in T$, $w =^{\sim} hpxvkuxq$, $|w'| < |w|$, $|w''| < |w|$. In turn, this is equivalent to the existence of $w', w'' \in C'$ (induction hypothesis) and to a rule $r = px|q\$ux|v$ in R' (by construction of R') such that w', w'' generate w through r, i.e., $w \in C'$.

Given $SC_{PA} = (A, I, R)$ and $C = C(SC_{PA})$, let us consider the system $SC_{PI} = (A, I, R')$, where $R' = \{(u_1u_2, u_3u_4; u_1u_4), (u_3u_4, u_1u_2; u_3u_2) \mid u_1|u_2\$u_3|u_4 \in R\}$, and $C' = C(SC_{PI})$. We prove that $C = C'$, by induction on $|w|$, for $w \in C \cup C'$. Indeed, if $w \in I$ then obviously $w \in C \cap C'$. On the other hand $w \in C \setminus I$ if and only if there exists a rule $u_1|u_2\$u_3|u_4 \in R$, $w' =^{\sim} hu_1u_2, w'' =^{\sim} ku_3u_4 \in C$, $|w'| < |w|$, $|w''| < |w|$, such that w', w'' generate w through $u_1|u_2\$u_3|u_4$. In turn, this is equivalent to the existence of $w', w'' \in C'$ (induction hypothesis) and to a pair of rules $r = (u_1u_2, u_3u_4; u_1u_4)$, $\bar{r} = (u_3u_4, u_1u_2; u_3u_2)$ in R' (by construction of R') such that w', w'' generate w through $r, \bar{r}$, i.e., $w \in C'$. ■

Proposition 6.2 allows us to state that some circular languages are not generated by Head or Paun splicing systems.

Proposition 6.2 *Let $L = Lin(C)$ be an infinite language which is the full linearization of a circular language C. Suppose that L satisfies the following condition*

$$\forall l, m \in L: \quad lm \notin L.$$

Then $^{\sim}L$ *is not* C_{PA} *generated.*

Proof. By contradiction, suppose that $^{\sim}L$ is C_{PA} generated. Let w be a word of L with length greater than the maximal length of the words in I. Since $w \notin I$, there exist $^{\sim}hu_1u_2$, $^{\sim}ku_3u_4 \in {}^{\sim}L$ such that $w \sim u_2hu_1u_4ku_3$. Now, L being closed under the conjugacy relation (Proposition 4.1) and w being in L, we have $u_2hu_1u_4ku_3 \in L$. Thus, $u_2hu_1 \in {}^{\sim}hu_1u_2$ and $^{\sim}hu_1u_2 \in {}^{\sim}L$ and hence $l = u_2hu_1 \in L$. By the same argument, $m = u_4ku_3 \in L$ and $lm = u_2hu_1u_4ku_3 \in L$ which is a contradiction. ■

The following examples give results concerning the computational power of splicing systems. Namely, Example 6.1 shows the difference between linear and circular splicing. The same Example 6.1 and Example 6.3 show that circular Paun splicing systems cannot generate all circular regular languages. As we have already said, Pixton systems are more powerful and this is emphasized in Examples 6.2 and 6.4.

Example 6.1 In [71] it is shown that $(aa)^*$ is not a linear splicing language, while its circularization $\{^{\sim}(aa)^n | n \geq 0\}$ is C_H generated by $SC_H = (A, I, T, P)$ with $A = \{a\}, I = \{^{\sim}1, ^{\sim}aa\}, T = \{(1, a, 1)\}$, and $(1, a, 1)P(1, a, 1)$. We can observe that $\{^{\sim}(aa)^n | n \geq 0\}$ is also C_{PA} generated, by choosing $SC_{PA} = (A, I, R)$, with $A = \{a\}, I = \{^{\sim}aa\} \cup 1$, $R = \{aa|1\$1|aa\}$, as shown in Section 8. On the contrary, let us consider the regular language $(aa)^*b$. Using results in [8], we know that $(aa)^*b$ is generated by a finite Paun linear system. Take its circularization $C = {}^{\sim}(aa)^*b$. Notice that there is a linearization of C, $\{a^kba^k : k \in \boldsymbol{N}\}$ which is not regular. However, the full linearization $Lin(C) = (aa)^*\{b, aba\}(aa)^*$, given in [66], is regular. Thus, $C \in Reg^{\sim}$ (Chapter 4). Furthermore, for all $l, m \in Lin(C)$, we have $|lm|_b = 2$ and so $lm \notin Lin(C)$. Using Proposition 6.2, we have that C cannot be C_H or C_{PA} generated.

Example 6.2 Let us consider again $C = {}^{\sim}(aa)^*b$. We know that C cannot be C_H or C_{PA} generated (Example 6.1). On the contrary, we prove that C is C_{PI} generated by choosing $SC_{PI} = (A, I, R)$, with $A = \{a, b\}, I =$

$\{^{\sim}b, ^{\sim}a^2b, ^{\sim}a^4b\}$, $R = \{(a^2, a^2b; a^2), (a^2b, a^2; 1)\}$. Firstly let us show that $C(SC_{PI}) \subseteq C$, by induction on the length of $w \in C(SC_{PI})$. If $w \in I$, then $w \in C$. Now, let us suppose that there exist $w_1, w_2 \in C(SC_{PI})$ and $r, \bar{r} \in R$ such that $(w_1, w_2) \vdash_{r,\bar{r}} w$ and let us prove that $w \in C$. Since $w_1, w_2 \in C(SC_{PI})$, then, by induction hypothesis, $w_1, w_2 \in C$. Thus, $k', h' \in \boldsymbol{N}$ exist such that $|w_1|_a = 2k', |w_2|_a = 2h'$ and $|w_1|_b = 1 = |w_2|_b$, i.e., $w_1 = {}^{\sim}(aa)^{k'}b$, $w_2 = {}^{\sim}(aa)^{h'}b$, $k', h' \geq 1$. Given that $R = \{(a^2, a^2b; a^2), (a^2b, a^2; 1)\}$, looking at Pixton's definition of splicing, with the same notations as in this definition, we see that $w_1 = {}^{\sim}aa\epsilon$ with $\epsilon = {}^{\sim}(aa)^{k'-1}b$, $w_2 = {}^{\sim}aab\epsilon'$ with $\epsilon' = {}^{\sim}(aa)^{h'-1}$ and $w = {}^{\sim}\epsilon aa\epsilon'$. Thus, $w = {}^{\sim}(aa)^{k'}b(aa)^{h'-1} \in C$.

Vice versa, let us show that $C \subseteq C(SC_{PI})$, i.e., by using induction over n, we prove that for every $n \geq 0$, we have $w = {}^{\sim}(aa)^n b \in C(SC_{PI})$. If $n = 0, 1, 2$, then $w \in I$. Let us suppose $n > 2$ and ${}^{\sim}(aa)^i b \in C(SC_{PI})$, for $0 \leq i < n$. We prove that there exist $w_1, w_2 \in C$ and $r, \bar{r} \in R$ such that $(w_1, w_2) \vdash_{r,\bar{r}} {}^{\sim}(aa)^n b$. Take $w_1 = {}^{\sim}(aa)^s b$ and $w_2 = {}^{\sim}(aa)^t b$ with $0 < s < n$, $0 < t < n$ such that $s + (t-1) = n$. By induction hypothesis, $w_1, w_2 \in C(SC_{PI})$. Furthermore, by setting ${}^{\sim}(aa)^{s-1}b = \epsilon$, ${}^{\sim}(aa)^{t-1} = \epsilon'$ and by using the rules $r, \bar{r} \in R$, we get $(w_1, w_2) \vdash_{r,\bar{r}} {}^{\sim}(aa)^{s-1}baa(aa)^{t-1} = {}^{\sim}(aa)^n b$.

Example 6.3 As we have already seen in Example 6.1, Proposition 6.2 allows us to state that some circular regular languages cannot be generated by circular Paun splicing systems. As another example, consider $L_1 = \{w \in A^* \mid \exists h, k \in \boldsymbol{N} \quad |w|_a = 2k, |w|_b = 2h\}$ and $L_2 = \{w \in A^* \mid \exists h, k \in \boldsymbol{N} \quad |w|_a = 2k+1, |w|_b = 2h+1\}$. It is easy to see that L_1 and L_2 are languages closed under conjugacy. Moreover, L_1 and L_2 are regular languages since L_1 is the shuffle of $(aa)^*$ and $(bb)^*$ and L_2 is the shuffle of $(aa)^*a$ and $(bb)^*b$ (see [40] for the definition of the shuffle operation between languages). Thus, ${}^{\sim}L_1$ and ${}^{\sim}L_2$ are circular regular languages. This example will be considered again in Section 7, where we will show that ${}^{\sim}L_1$ is C_{PA} generated. On the other hand, for all $l, m \in L_2$, since $|lm|_a$ and $|lm|_b$ are both even, then $lm \notin L_2$. So, thanks to Proposition 6.2, we have that ${}^{\sim}L_2$ is not C_H or C_{PA} generated. In Example 6.4 we show how ${}^{\sim}L_2$ can be generated by a Pixton splicing system.

Example 6.4 Let us prove that $^{\sim}L_2$ is C_{PI} generated by using induction over $|w|$ and by choosing $SC_{PI} = (A, I, R)$, with $A = \{a, b\}$, $I = \{^{\sim}ab, ^{\sim}aaab, ^{\sim}bbba\}$, $R = \{r_0, r_1, r_2, r_3\}$, $r_1 = (ab, 1; aa), r_2 = (ab, 1; bb), r_3 = (ab, 1; abab), \overline{r}_1 = \overline{r}_2 = \overline{r}_3 = r_0 = (1, ab; 1)$ (see Remark 5.1).
Firstly, let us show that $C(SC_{PI}) \subseteq ^{\sim}L_2$. Clearly, $I \subseteq ^{\sim}L_2$. Let $w \in C(SC_{PI})$ with $|w| = n$. Thus, $w_1, w_2 \in C(SC_{PI})$ exist and $r, \overline{r} \in R$ also exist such that $(w_1, w_2) \vdash_{r,\overline{r}} w$. We observe that the application of a pair of rules in R produces a word the length of which is greater than the lengths of the spliced words. So, $|w_1| < n, |w_2| < n$ and, by induction hypothesis, $w_1, w_2 \in ^{\sim}L_2$. We can distinguish three cases.

Case 1. Let $\overline{r} = r_0 = (1, ab; 1)$ and $r = r_1 = (ab, 1; aa)$. Thus $w_1 = ^{\sim}\epsilon$, $w_2 = ^{\sim}ab\epsilon'$ and so $w = ^{\sim}\epsilon\epsilon' aa$. We have $w \in ^{\sim}L_2$. Indeed, $|w|_a = |\epsilon\epsilon' aa|_a = |w_1|_a + |\epsilon'|_a + 2 = |w_1|_a + |w_2|_a + 1$. By induction hypothesis, $|w_1|_a$ and $|w_2|_a$ are both odd, so $|w|_a$ is odd. Moreover, $|w|_b = |\epsilon\epsilon' aa|_b = |\epsilon\epsilon'|_b = |w_1|_b + |\epsilon'|_b = |w_1|_b + |w_2|_b - 1$. By induction hypothesis, $|w_1|_b$ and $|w_2|_b$ are both odd, so $|w|_b$ is odd.

Case 2. Let $\overline{r} = r_0 = (1, ab; 1)$ and $r = r_2 = (ab, 1; bb)$. This case is symmetric to case 1, so we can use the same argument as in case 1 by substituting a with b and vice versa.

Case 3. Let $\overline{r} = r_0 = (1, ab; 1)$ and $r = r_3 = (ab, 1; abab)$. Thus, $w_1 = ^{\sim}\epsilon$, $w_2 = ^{\sim}ab\epsilon'$ and so $w = ^{\sim}\epsilon\epsilon' abab$. We have $w \in ^{\sim}L_2$. Indeed, $|w|_a = |\epsilon\epsilon' abab|_a = |\epsilon\epsilon'|_a + 2 = |\epsilon|_a + |\epsilon'|_a + 2 = |w_1|_a + |w_2|_a + 1$. By induction hypothesis, $|w_1|_a$ and $|w_2|_a$ are both odd, so $|w|_a$ is odd. Moreover, $|w|_b = |\epsilon\epsilon' abab|_b = |\epsilon\epsilon'|_b + 2 = |w_1|_b + |w_2|_b + 1$. Once again, by induction hypothesis, $|w_1|_b$ and $|w_2|_b$ are both odd, so $|w|_b$ is odd.

Vice versa, let us show that $^{\sim}L_2 \subseteq C(SC_{PI})$. If $w \in ^{\sim}L_2$ and $|w| < 5$, then $w \in I$. Let $w \in ^{\sim}L_2$ with $|w| = n \geq 5$. Once again, we can distinguish three cases.

Case 1. Suppose that aa is a factor of w, i.e., $w_1 aa w_2 \in Fact(w)$. Take $x = w_2 w_1$ and $y = ab$. We have $|x| < |w|, |y| < 5 \leq |w|$. In addition, $^{\sim}y \in I \subseteq C(SC_{PI})$ and $^{\sim}x \in ^{\sim}L_2$ ($|x|_a = |w|_a - 2$ and $|x|_b = |w|_b$ are both odd numbers). Thus, by induction hypothesis, $^{\sim}x \in C(SC_{PI})$. Furthermore, $(^{\sim}x, ^{\sim}y) \vdash_{r_0,r_1} ^{\sim}w$. Indeed, when we set $\epsilon = w_2 w_1$, $\beta = 1$, $\epsilon' = 1$, $\beta' = aa$,

we get $(x,y)\vdash_{r_0,r_1} {}^{\sim}\epsilon\beta\epsilon'\beta' = {}^{\sim}w_2w_1aa = {}^{\sim}w$.

Case 2. Suppose that bb is a factor of w, i.e., $w_1bbw_2 \in Fact(w)$. This case is symmetric to case 1, so by using the same argument as in case 1, we can prove that $w = {}^{\sim}w_2w_1bb \in C(SC_{PI})$ by using r_0, r_2 (applied to ${}^{\sim}x = {}^{\sim}w_2w_1$ and ${}^{\sim}y = {}^{\sim}ab$).

Case 3. Suppose that neither aa nor bb are factors of w. Thus, $w = {}^{\sim}(ab)^t$, where t is an odd number and $t > 1$. Consider ${}^{\sim}x = {}^{\sim}(ab)^{t-2}$ and ${}^{\sim}ab \in {}^{\sim}L_2$. Since t is odd, $t-2$ is odd too, i.e. ${}^{\sim}x = {}^{\sim}(ab)^{t-2} \in {}^{\sim}L_2$. In addition, we have $|x| < |w|, |ab| < 5 \leq |w|$. Thus, ${}^{\sim}ab \in I \subseteq C(SC_{PI})$ and, by induction hypothesis, ${}^{\sim}x \in C(SC_{PI})$. Finally, when we set $\alpha = 1, \epsilon = x, \beta = 1$ and $\alpha' = ab$, $\epsilon' = 1, \beta' = abab$, we have $({}^{\sim}x, {}^{\sim}ab)\vdash_{r_0,r_3} {}^{\sim}\epsilon\beta\epsilon'\beta' = {}^{\sim}xababab = {}^{\sim}(ab)^t$.

In the next chapters we approach Problem 5.1 by dealing with different classes of languages. The last one is devoted to investigations on Pixton splicing systems.

We conclude this section with some remarks related to Problem 5.1, i.e., the characterization of the circular regular languages generated by finite circular splicing and with some problems which are still open. In all of them, we will refer to Paun's definition of circular splicing.

First of all, note that the class of regular languages closed under conjugacy relation is closed under union. However, while ${}^{\sim}(A^2)^*$, ${}^{\sim}(A^3)^* \in C(Fin, Fin)$, as shown in Proposition 7.3, we prove in Proposition 6.3 that ${}^{\sim}(A^2)^* \cup {}^{\sim}(A^3)^* \notin C(Fin, Fin)$. This means that $Reg^{\sim} \cap C(Fin, Fin)$ is not closed with respect to union. One could ask for additional hypotheses to be added so that the union of circular regular splicing languages will still be a circular splicing language. Proposition 7.3 gives a partial answer to this question. This research line could bring us to a complete characterization of the class of circular regular languages that are generated by finite circular splicing. In the next proposition, as usual, for positive integers s, t we say that s divides t if $t = sv$ for a positive integer v.

Proposition 6.3 ${}^{\sim}(A^2)^* \cup {}^{\sim}(A^3)^* \notin C(Fin, Fin)$.

Proof. By contradiction, suppose that $^{\sim}(A^2)^* \cup {}^{\sim}(A^3)^* = C(SC_{PA})$, for a circular splicing system $SC_{PA} = (A, I, R)$. Let $r = u_1|u_2\$u_3|u_4$, with $r \in R$. Then, there exists $h \in A^*$ such that 3 does not divide $s = |hu_1u_2|$ (take $h = 1$ if $|u_1u_2| \not\equiv 0 \pmod 3$, $h \in A$ otherwise). Analogously, there exists $k \in A^*$ such that 2 does not divide $t = |ku_3u_4|$ (take $k = 1$ if $|u_3u_4| \not\equiv 0 \pmod 2$, $k \in A$ otherwise). Thus, consider the two words $w = (hu_1u_2)^2$ and $z = (ku_3u_4)^3$. We have $^{\sim}w \in {}^{\sim}(A^2)^*$, $^{\sim}z \in {}^{\sim}(A^3)^*$ and so $^{\sim}w$, $^{\sim}z \in C(SC_{PA})$. Furthermore, rule r can be applied to $^{\sim}w$,$^{\sim}z$. By definition, this rule r generates a circular word $^{\sim}y$ which should be in $C(SC_{PA})$ and $|y| = 2s+3t$. Now, $^{\sim}y \notin {}^{\sim}(A^2)^*$ (since 2 does not divide $2s + 3t$) and $^{\sim}y \notin {}^{\sim}(A^3)^*$ (since 3 does not divide $2s + 3t$), and this is a contradiction. ■

6.2 A circular regular language not generated

In this section we prove that the circular regular language $^{\sim}((A^2)^* \cup (A^3)^*)$ cannot be generated by finite splicing, even in the case of the more powerful definition given by Pixton.

We preliminary observe that $Lin(^{\sim}(A^n)^*) = (A^n)^*$ and $^{\sim}(A^n)^*$ is a regular circular language, since its full linearization is regular (as a matter of fact this full linearization belongs to the class of star languages which are defined in the next section). Furthermore, if we take into account the observation made in the previous section, we have that $L = Lin(^{\sim}((A^2)^* \cup (A^3)^*)) = (A^2)^* \cup (A^3)^*$, and $^{\sim}((A^2)^* \cup (A^3)^*)$ is also a circular regular language.

Let us sketch the proof of this result. By contradiction, suppose that there exists $SC_{PI} = (A, I, R)$ such that $^{\sim}L = {}^{\sim}((A^2)^* \cup (A^3)^*) = C(SC_{PI})$. Let us consider $(\alpha, \alpha'; \beta), (\alpha', \alpha; \beta') \in R$ and $^{\sim}\alpha\epsilon$, $^{\sim}\alpha'\epsilon' \in {}^{\sim}L$. Observe that $|\epsilon\beta\epsilon'\beta'| = |\epsilon\alpha| + |\epsilon'\alpha'| + |\beta\beta'| - |\alpha\alpha'|$. Since $^{\sim}\alpha\epsilon$, $^{\sim}\alpha'\epsilon' \in {}^{\sim}L$, we must have that $^{\sim}\epsilon\beta\epsilon'\beta' \in {}^{\sim}L$, i.e., either $|\epsilon\beta\epsilon'\beta'| \equiv 0 (\mathrm{mod}\ 2)$, or $|\epsilon\beta\epsilon'\beta'| \equiv 0 (\mathrm{mod}\ 3)$.

We will prove that, for any pair of triples $(\alpha, \alpha'; \beta), (\alpha', \alpha; \beta') \in R$, there exists a pair of words $^{\sim}\alpha\epsilon$, $^{\sim}\alpha'\epsilon' \in {}^{\sim}L$ such that the triples may be applied to them and we can generate a word that does not belong to $^{\sim}L$ (that is a contradiction).

We will distinguish the following four different cases with respect to the length of $|\beta\beta'|$. Precisely (we denote $(*)$ these cases):

1. $|\beta\beta'| \equiv 0(\text{mod } 2)$, $|\beta\beta'| \equiv 0(\text{mod } 3)$;
2. $|\beta\beta'| \equiv 0(\text{mod } 2)$, $|\beta\beta'| \not\equiv 0(\text{mod } 3)$,
3. $|\beta\beta'| \not\equiv 0(\text{mod } 2)$, $|\beta\beta'| \equiv 0(\text{mod } 3)$;
4. $|\beta\beta'| \not\equiv 0(\text{mod } 2)$, $|\beta\beta'| \not\equiv 0(\text{mod } 3)$

We will prove that for each of the above four cases, there exist $\epsilon, \epsilon' \in A^*$ such that the triples may be applied, generating a word not in $^{\sim}L$. More precisely, we will consider that for each of the above four cases in $(*)$, there is a choice of $\epsilon, \epsilon' \in A^*$ such that (denote $(**)$ these cases)

1'. $|\epsilon\epsilon'| \not\equiv 0(\text{mod } 2)$, $|\epsilon\epsilon'| \not\equiv 0(\text{mod } 3)$;
2'. $|\epsilon\epsilon'| \not\equiv 0(\text{mod } 2)$, $|\epsilon\epsilon'| \equiv 0(\text{mod } 3)$,
3'. $|\epsilon\epsilon'| \equiv 0(\text{mod } 2)$, $|\epsilon\epsilon'| \not\equiv 0(\text{mod } 3)$;
4'. $|\epsilon\epsilon'| \equiv 0(\text{mod } 2)$, $|\epsilon\epsilon'| \equiv 0(\text{mod } 3)$

Thus, let us consider the following statement $(+)$, for each $(\alpha, \alpha'; \beta), (\alpha', \alpha; \beta') \in R$, for any pairs of words $^{\sim}\alpha\epsilon$, $^{\sim}\alpha'\epsilon' \in {}^{\sim}L$, for $i = \{1, \ldots, 4\}$

$$\text{if} \quad \text{case } i \quad \text{holds} \quad \text{then} \quad \text{case } i' \quad \text{holds} \qquad (+)$$

Now, it is evident that, if we can prove that $(+)$ holds, for $i = \{1, \ldots, 4\}$, then the conclusion easily follows, since $|\epsilon\beta\epsilon'\beta'| = |\epsilon\epsilon'| + |\beta\beta'|$ and, for each of the four expressions of $(+)$ we have $|\epsilon\epsilon'| + |\beta\beta'|$ is an odd number not multiple of 3.

Thus, in the next claims we will suppose by contradiction, that there exists $SC_{PI} = (A, I, R)$ such that $L = {}^{\sim}((A^2)^* \cup (A^3)^*) = C(SC_{PI})$ and consider $(\alpha, \alpha'; \beta), (\alpha', \alpha; \beta') \in R$. Let us prove that if case 1 (resp. 2, 3, 4) holds, then case 1' (resp. 2', 3', 4') holds.

Claim 1. If $|\beta\beta'| \equiv 0(\text{mod } 2)$ and $|\beta\beta'| \equiv 0(\text{mod } 3)$, then there exist $\epsilon, \epsilon' \in A^*$ such that $^{\sim}\alpha\epsilon$, $^{\sim}\alpha'\epsilon' \in {}^{\sim}L$, $|\epsilon\epsilon'| \not\equiv 0(\text{mod } 2)$, $|\epsilon\epsilon'| \not\equiv 0(\text{mod } 3)$.

Proof. If $\alpha \equiv 0(\text{mod } 3)$, then we can choose ϵ such that $|\epsilon| \equiv 0(\text{mod } 3)$ (i.e., $^{\sim}\alpha\epsilon \in {}^{\sim}L$) and so ϵ' such that $|\epsilon'| \not\equiv 0(\text{mod } 3)$ (i.e., $|\epsilon'\epsilon| \not\equiv 0(\text{mod } 3)$). Otherwise, if $\alpha \not\equiv 0(\text{mod } 3)$, then we can choose ϵ such that $|\epsilon\alpha| \equiv 0(\text{mod } 3)$ (i.e., $^{\sim}\alpha\epsilon \in {}^{\sim}L$) and so ϵ' such that $|\epsilon'| \equiv 0(\text{mod } 3)$ (i.e., $|\epsilon'\epsilon| \not\equiv 0(\text{mod } 3)$).

Moreover, if $\alpha' \equiv 0(\text{mod } 2)$, then ϵ' must be also such that $|\epsilon'| \equiv 0$ (mod 2) (i.e., $^{\sim}\ \alpha'\epsilon' \in {}^{\sim}L$) and so ϵ such that $|\epsilon| \not\equiv 0(\text{mod } 2)$ (i.e., $|\epsilon'\epsilon| \not\equiv 0$ (mod 2)). Otherwise, if $\alpha' \not\equiv 0(\text{mod } 2)$, then ϵ' must be also such that $|\epsilon'| \not\equiv 0(\text{mod } 2)$ (i.e., $^{\sim}\ \alpha'\epsilon' \in {}^{\sim}L$) and so ϵ such that $|\epsilon| \equiv 0(\text{mod } 2)$ (i.e., $|\epsilon'\epsilon| \not\equiv 0(\text{mod } 2)$).

Claim 2. If $|\beta\beta'| \equiv 0(\text{mod } 2)$ and $|\beta\beta'| \not\equiv 0(\text{mod } 3)$, then there exist $\epsilon, \epsilon' \in A^*$ such that $^{\sim}\alpha\epsilon$, $^{\sim}\alpha'\epsilon' \in {}^{\sim}L$, $|\epsilon\epsilon'| \not\equiv 0(\text{mod } 2)$, $|\epsilon\epsilon'| \equiv 0(\text{mod } 3)$.

Proof. If $\alpha \equiv 0(\text{mod } 3)$, then we can choose ϵ such that $|\epsilon| \equiv 0(\text{mod } 3)$ (i.e., $^{\sim}\alpha\epsilon \in {}^{\sim}L$) and so ϵ' such that $|\epsilon'| \equiv 0(\text{mod } 3)$ (i.e., $|\epsilon'\epsilon| \equiv 0(\text{mod } 3)$). Otherwise, if $\alpha \not\equiv 0(\text{mod } 3)$, then we can choose ϵ such that $|\epsilon\alpha| \equiv 0(\text{mod } 3)$ (i.e., $^{\sim}\alpha\epsilon \in {}^{\sim}L$) and so ϵ' such that $|\epsilon'| \equiv |\alpha|(\text{mod } 3)$ (i.e., $|\epsilon'\epsilon| \equiv 0(\text{mod } 3)$).

Moreover, if $\alpha' \equiv 0(\text{mod } 2)$, then ϵ' must be also such that $|\epsilon'| \equiv 0$ (mod 2) (i.e., $^{\sim}\ \alpha'\epsilon' \in {}^{\sim}L$) and so ϵ such that $|\epsilon| \not\equiv 0(\text{mod } 2)$ (i.e., $|\epsilon'\epsilon| \not\equiv 0$ (mod 2)). Otherwise, if $\alpha' \not\equiv 0(\text{mod } 2)$, then ϵ' must be also such that $|\epsilon'| \not\equiv 0(\text{mod } 2)$ (i.e., $^{\sim}\ \alpha'\epsilon' \in {}^{\sim}L$) and so ϵ such that $|\epsilon| \equiv 0(\text{mod } 2)$ (i.e., $|\epsilon'\epsilon| \not\equiv 0(\text{mod } 2)$).

Claim 3. If $|\beta\beta'| \not\equiv 0(\text{mod } 2)$ and $|\beta\beta'| \equiv 0(\text{mod } 3)$, then there exist $\epsilon, \epsilon' \in A^*$ such that $^{\sim}\alpha\epsilon$, $^{\sim}\alpha'\epsilon' \in {}^{\sim}L$, $|\epsilon\epsilon'| \equiv 0(\text{mod } 2)$, $|\epsilon\epsilon'| \not\equiv 0(\text{mod } 3)$.

Proof. If $\alpha \equiv 0(\text{mod } 3)$, then we can choose ϵ such that $|\epsilon| \equiv 0(\text{mod } 3)$ (i.e., $^{\sim}\alpha\epsilon \in {}^{\sim}L$) and so ϵ' such that $|\epsilon'| \not\equiv 0(\text{mod } 3)$ (i.e., $|\epsilon\epsilon'| \not\equiv 0(\text{mod } 3)$). Otherwise, if $\alpha \not\equiv 0(\text{mod } 3)$, then we can choose ϵ such that $|\epsilon\alpha| \equiv 0(\text{mod } 3)$ (i.e., $^{\sim}\alpha\epsilon \in {}^{\sim}L$) and so ϵ' such that $|\epsilon'| \equiv 0(\text{mod } 3)$ (i.e., $|\epsilon\epsilon'| \equiv 0(\text{mod } 3)$).

Moreover, if $\alpha' \equiv 0(\text{mod } 2)$, then ϵ' must be also such that $|\epsilon'| \equiv 0$ (mod 2) (i.e., $^{\sim}\ \alpha'\epsilon' \in {}^{\sim}L$) and so ϵ such that $|\epsilon| \equiv 0(\text{mod } 2)$ (i.e., $|\epsilon\epsilon'| \equiv 0$ (mod 2)). Otherwise, if $\alpha' \not\equiv 0(\text{mod } 2)$, then ϵ' must be also such that $|\epsilon'| \not\equiv 0(\text{mod } 2)$ (i.e., $^{\sim}\ \alpha'\epsilon' \in {}^{\sim}L$) and so ϵ such that $|\epsilon| \not\equiv 0(\text{mod } 2)$ (i.e., $|\epsilon\epsilon'| \equiv 0(\text{mod } 2)$).

Claim 4. If $|\beta\beta'| \not\equiv 0(\text{mod } 2)$ and $|\beta\beta'| \not\equiv 0(\text{mod } 3)$, then there exist $\epsilon, \epsilon' \in A^*$ such that ${}^\sim\alpha\epsilon, {}^\sim\alpha'\epsilon' \in {}^\sim L$, $|\epsilon\epsilon'| \equiv 0(\text{mod } 2)$, $|\epsilon\epsilon'| \equiv 0(\text{mod } 3)$.

Proof. If $\alpha \equiv 0(\text{mod } 3)$, then we can choose ϵ such that $|\epsilon| \equiv 0(\text{mod } 3)$ (i.e., ${}^\sim\alpha\epsilon \in {}^\sim L$) and so ϵ' such that $|\epsilon'| \equiv 0(\text{mod } 3)$ (i.e., $|\epsilon\epsilon'| \equiv 0(\text{mod } 3)$). Otherwise, if $\alpha \not\equiv 0(\text{mod } 3)$, then we can choose ϵ such that $|\epsilon\alpha| \equiv 0(\text{mod } 3)$ (i.e., ${}^\sim\alpha\epsilon \in {}^\sim L$) and so ϵ' such that $|\epsilon'| \equiv |\alpha|(\text{mod } 3)$ (i.e., $|\epsilon\epsilon'| \equiv 0(\text{mod } 3)$).

Moreover, if $\alpha' \equiv 0(\text{mod } 2)$, then ϵ' must be also such that $|\epsilon'| \equiv 0 (\text{mod } 2)$ (i.e., ${}^\sim \alpha'\epsilon' \in {}^\sim L$) and so ϵ such that $|\epsilon| \equiv 0(\text{mod } 2)$ (i.e., $|\epsilon\epsilon'| \equiv 0 (\text{mod } 2)$). Otherwise, if $\alpha' \not\equiv 0(\text{mod } 2)$, then ϵ' must be also such that $|\epsilon'| \not\equiv 0(\text{mod } 2)$ (i.e., ${}^\sim \alpha'\epsilon' \in {}^\sim L$) and so ϵ such that $|\epsilon| \not\equiv 0(\text{mod } 2)$ (i.e., $|\epsilon\epsilon'| \equiv 0(\text{mod } 2)$).

Proposition 6.4 *For every circular Pixton splicing system SC_{PI}, the circular regular language ${}^\sim((A^2)^* \cup (A^3)^*) \neq C(SC_{PI})$.*

As we have already said, Proposition 6.4 proves the following result.

Theorem 6.1 *There exists at least one circular regular language which is not generated by finite circular splicing.*

Finally we suppose the following conjecture true, and this implies that the inclusions in Proposition 6.1 are strict.

Conjecture 6.1 *There exist circular regular languages which are Paun generated, but not Head generated.*

Final remarks. We close this chapter by introducing the notion of *splicing sub-system* of a given circular splicing system, related to descriptional complexity. For example, ${}^\sim(A^2)^*$, for $A = \{a, b\}$, is C_{PA} generated by $I = {}^\sim A^2$ and $R = \{w_1|1\$1|\ w_2 \mid\ \ w_1, w_2 \in A^2\}$ (Proposition 7.3). Thus, with the same A and I, by considering $R_1 = \{aa|1\$1|aa\} \subseteq R$, we obtain ${}^\sim(aa)^*$, with $R_2 = \{bb|1\$1|bb\} \subseteq R$ we generate ${}^\sim(bb)^*$ and with $R_3 = \{ab|1\$1|ab\} \subseteq R$ we obtain ${}^\sim(ab)^*$. It would be interesting to prove or to disprove that all circular languages can be obtained as described before. Let us take $L_1 = \{w \in A^* \mid \exists h, k \in \boldsymbol{N} \ \ |w|_a = 2k, |w|_b = 2h\}$

as in Example 6.3. In Section 7, we will see that the splicing system $SC_{PA} = (A, I, R)$ generates $\sim L_1$, when we choose $I = \{\sim aa, \sim bb, \sim abab\} \cup 1$ and $R = \{aa|1\$1|1, bb|1\$1|1, ab|1\$1|1\}$. On the other hand, one can see that $\sim L_1$ cannot be generated by choosing a subset of R.

Chapter 7

Circular regular Paun generated languages: star languages

In this chapter, we will introduce a class of regular languages which we prove to be C_{PA} generated. As we have already said, we will consider languages L closed under conjugacy relation, and the action of circular splicing over their circularization $^{\sim}L = C$. Furthermore, since we will be referring to Problem 5.1, L will be regular. Here, we also consider the particular case in which $L = X^*$ is the Kleene closure of a regular language (Definition 7.1). We give some additional hypotheses under which these languages are C_{PA} generated (Theorem 7.1, Proposition 7.3). Several examples are given comparing this class of languages with both star languages and circular regular languages generated by (Paun or Pixton) splicing (Examples 7.4, 7.5, 7.6) and some questions are posed (Problems 7.1, 7.2).

7.1 New definitions

We define the class of languages that will be analyzed in this chapter. We will show some interesting properties of this class, that can be interpretated from a splicing theory point of view.

Definition 7.1 *A* star language *is a language $L \subseteq A^*$ that satisfies the following conditions:*

(1) $L = X^*$, with X a regular language,

(2) L is closed under conjugacy relation.

Under an additional hypothesis, a star language has its circularization in $C(Fin, Fin)$, as shown in Theorem 7.1, the proof of which can be obtained thanks to the following proposition.

Proposition 7.1 *Let X^* be a star language and let $SC_{PA} = (A, I, R)$ be a splicing system. If $I \subseteq {}^\sim X^*$ then $C(SC_{PA}) \subseteq {}^\sim X^*$.*

Proof. Let us prove that $C(SC_{PA}) \subseteq {}^\sim X^*$ holds, by using induction over the length $|y|$ of ${}^\sim y \in C(SC_{PA})$. If ${}^\sim y \in I$ then ${}^\sim y \in {}^\sim X^*$, since $I \subseteq {}^\sim X^*$. Otherwise, $y' \in {}^\sim y$ exists such that $y' = u_2hu_1u_4ku_3$ with ${}^\sim u_2hu_1, {}^\sim u_4ku_3 \in C(SC_{PA})$ and ${}^\sim u_2hu_1 \neq {}^\sim y, {}^\sim u_4ku_3 \neq {}^\sim y$. Then $|u_2hu_1| < |y|, |u_4ku_3| < |y|$. By induction hypothesis, ${}^\sim u_2hu_1, {}^\sim u_4ku_3 \in {}^\sim X^*$ and, X^* being closed under conjugacy, $u_2hu_1, u_4ku_3 \in X^*$. Because X^* is a submonoid, $u_2hu_1u_4ku_3 = y' \in X^*$, and so ${}^\sim y \in {}^\sim X^*$. ■

Proposition 7.1 reveals the reason why we have firstly approached C_{PA} generated languages starting with star languages. Indeed, if I is a finite set of words in a star language ${}^\sim X^*$, by Proposition 7.1 we have that for every R, the language generated by the splicing system is contained in ${}^\sim X^*$. This is the same as saying that each subset of X^* is closed under the action of an arbitrary set of rules. Thus, in order to prove the equivalence between ${}^\sim X^*$ and a circular (Paun) splicing system (such that $I \subseteq {}^\sim X^*$), we must only exibit rules generating it, when this is possible. Indeed, intuitively, if a rational not finite language L is generated by (finite) splicing, the set of rules is applied producing words with a non bounded number of occurrences of such labels as factors. Then, the definition below is crucial for our aim. We stress that the definition of cycle given below is different from the one given in Chapter 3, being different the approach used here.

Definition 7.2 (Cycle) *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. each other and with respect to q).*

Then, if c is a cycle in $\mathcal{A}$ and c is not simple, there exist $u_1, \ldots, u_{k+1} \in A^*$, 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 transitions 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 $c_i = c_j$ then $u_{i+1} \cdots u_j \neq 1$.

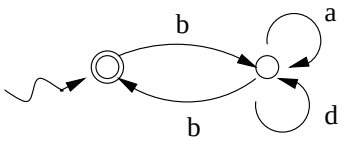

Figure 7.1: *Automaton in Example 7.1.*

Example 7.1 Let us consider the automaton $\mathcal{A}$ depicted in Figure 7.1. A word $w \in \{a, b, d\}^*$ is the label of a closed path in $\mathcal{A}$ if and only if $w \in (b(d^*a^*)^*b)^* \cup ((d^*a^*)^*bb)^*$. The set of the cycles in $\mathcal{A}$ is a proper subset of this set of words. Indeed, these cycles are: a, d, ba^pd^rb, bd^ra^pb, for each $p, r \geq 0$. Hence, as an example, $baddab$ is the label of a closed path which is not a cycle.

In the proof of Theorem 7.1, we will use a finite set of representatives of the cycles in a trim deterministic automaton and a particular class of languages. This is the reason why we will give Definitions 7.3 and 7.4. Furthermore, we notice that the property for c of being a cycle in $\mathcal{A}$ strictly depends on the automaton $\mathcal{A}$. Even if we restrict 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 $\mathcal{A}$. We will overcome this difficulty since the condition given in Definition 7.4 allows us to go from labels in a fixed automaton $\mathcal{A}$ to factors in the words of the recognized language $L(\mathcal{A})$.

Given a regular language L and a finite state automaton $\mathcal{A}$ recognizing L, we inductively define the *fingerprint* $F(c)$ of a cycle c in $\mathcal{A}$ and the class of *fingerprint closed languages* L with respect to $\mathcal{A}$.

Definition 7.3 (Fingerprint of a cycle) *Let $\mathcal{A}$ be a finite state automaton recognizing a regular language L, i.e., $L = L(\mathcal{A})$. The set $\mathcal{C}(L) = \bigcup_{c\ cycle} F(c)$ of the fingerprints $F(c)$ for L associated with c, for every cycle c in $\mathcal{A}$, is inductively defined as follows:*

- *if c is simple, 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 $u_{i+1} \cdots u_j \neq 1$ if $c_i = c_j$), $F(c_1), \cdots, F(c_k)$ are fingerprints of $c_1, \ldots, c_k$, then $F(c) = u_1 F(c_1) u_2 F(c_2) \cdots u_k F(c_k) u_{k+1}$.*

Example 7.1 *(continued).* Let $\mathcal{A}$ be the finite state automaton depicted in Figure 7.1. The set $\mathcal{C}(L)$ of the fingerprints for $L = L(\mathcal{A})$ is $\mathcal{C}(L) = \{F(a), F(d), F(ba^p d^r b), F(bd^r a^p b) \mid p, r \geq 0\} = \{a, d, bb, bab, bdb, badb, bdab\}$.

Definition 7.4 (Fingerprint closed language) [1] *For a given finite state automaton $\mathcal{A}$, we say that $L = L(\mathcal{A})$ is a fingerprint closed language with respect to $\mathcal{A}$ whenever $\mathcal{C}(L) = \bigcup_{c\ cycle} F(c) \subseteq L$.*

Example 7.2 Let $L = L(\mathcal{A})$ be a language which is fingerprint closed with respect to a finite state automaton $\mathcal{A}$. Let c_1 and c_2 be two simples cycles. Observe that for all positive integers n, n', m, m' and for all $u_1, u_3 \in A^+$, $u_2 \in A^*$ such that $c = u_1 c_1^n u_2 c_2^{n'} u_3$, $c' = u_1 c_1^m u_2 c_2^{m'} u_3$ are cycles, we have $F(c) = F(c')$. Furthermore, $F(c) = u_1 c_1 u_2 c_2 u_3$.

Loosely speaking, if L is a fingerprint closed language, we are sure that for every closed path, the label of a finite crossing of this closed path is in L. A language which is not fingerprint closed is given in Example 7.3. Another

[1] It is interesting that we have found that fingerprinting is a real biological phenomenon, reported in the Appendix, very close to our definition.

example of a language which is not fingerprint closed is given in Example 7.5, whereas Proposition 7.2 provides a class of fingerprint closed languages L which are also star languages.

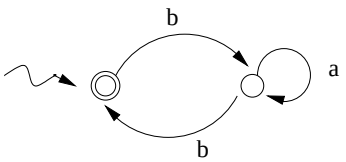

Figure 7.2: *Automaton for* $(ba^*b)^*$.

Example 7.3 Let us consider the regular language $L = (ba^*b)^*$. $L = (ba^*b)^*$ is not a fingerprint closed language with respect to the finite automaton which recognizes it and is depicted in Figure 7.2. Indeed, for the cycle $c_1 = a$, we have $F(c_1) = a \notin L$. Furthermore, let $\mathcal{A}$ be a trim deterministic automaton recognizing X^*. Let k be a positive integer greater than the number of the states in $\mathcal{A}$ and consider $ba^kb \in L$. It is easy to see that there exists j such that $j \leq k$ and a^j is a cycle in $\mathcal{A}$. Thus, since $L \cap a^+ = \emptyset$, L is not a fingerprint closed language with respect to $\mathcal{A}$. Observe that L is not a star language.

7.2 Fingerprint closed star languages are Paun generated

Lemma 7.1 is a preliminary step in the proof of our main result in this section. It allows us to state the finiteness of our splicing system. Indeed, in this lemma we will define a (finite) set of words whose circularization will be the initial set of the splicing system associated with a regular fingerprint closed star language L (Theorem 7.1). Informally, the initial language will be the set of the labels w of the successful paths in $\mathcal{A}$ such that, for each state q crossed by the transitions associated with w, there exists at most one closed path which starts and ends in q. Observe that the lemma does not use the fingerprint closure hypothesis.

Lemma 7.1 *Let $\mathcal{A} = (Q, A, \delta, q_0, F)$ be a trim deterministic finite state automaton recognizing a regular language $L = L(\mathcal{A})$ closed under conjugacy relation. Let $\mathcal{C}(L) = \bigcup_{c\ cycle} F(c)$ be the set of the fingerprints $F(c)$ for L associated with c, for every cycle c in $\mathcal{A}$. Thus, $\mathcal{C} = C(L)$ is finite.*
Let I_2 be the set containing $\mathcal{C}$, along with labels of special successful paths in $\mathcal{A}$: $I_2 = \mathcal{C} \cup I_1'$, $I_1' = \{w \in A^* \mid \delta(q_0, w) \in F$ and $[\forall x, z \in A^*, c \in A^+, q \in Q$ if $w = xc^kz$ with $\delta(q_0, x) = q$, $\delta(q, c) = q$, then $c = F(c)$ is a cycle and $k = 1]\}$. Thus, I_2 is also finite.

Proof. Let $\mathcal{A}$ be a finite state automaton recognizing a regular fingerprint closed language L. Let $\mathcal{C}(L) = \bigcup_{c\ cycle} F(c)$ be the set of the fingerprints $F(c)$ for L associated with c, for every cycle c in $\mathcal{A}$. Let us consider the set $\mathcal{I} = \{w \in A^* \mid [\forall x, z \in A^*, c \in A^+, q, q' \in Q$ if $w = xc^kz$ with $\delta(q', x) = q$, $\delta(q, c) = q$, then $c = F(c)$ is a cycle and $k = 1]\}$. It is easy to see that $\mathcal{C}(L), I_2, I_1'$ are subsets of $\mathcal{I}$. Thus, if we prove that $\mathcal{I}$ is a finite set, the conclusion holds. On the other hand, each $w \in \mathcal{I}$ is the label of a path π in $\mathcal{A}$ such that, for every $q \in Q$, π contains at most two occurrences of q (otherwise, $u, z \in A^*$, $c_1, c_2 \in A^+$ and $q \in Q$ exist such that $w = uc_1c_2z$, $\delta(q, c_1) = q$, $\delta(q, c_2) = q$. Thus, either $c_1 \neq c_2$ or $c_1 = c_2$. In the first case, using Definition 7.2, $c = c_1c_2$ is not a cycle and $w \notin \mathcal{I}$. In the second case, we also have $w = u(c_1)^2z \notin \mathcal{I}$). Consequently, $|w| \leq 2|Q|$ and $\mathcal{I}$ is a finite set. ■

We state our main result of this section.

Theorem 7.1 *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. Let X^* be a fingerprint closed star language with respect to a trim deterministic automaton $\mathcal{A}$ recognizing X^*. Let $\mathcal{C} = \mathcal{C}(X^*)$ be the set of the fingerprints of X^* with respect to the same automaton $\mathcal{A}$ (Definition 7.3). Let us consider the set I_2 defined in Lemma 7.1 and let $SC_{PA} = (A, I, R)$ where $I = \sim I_2$ and $R = \{1|1\$1|\beta\ ;\ \beta \in F(c),\ F(c) \in \mathcal{C},\ c$ cycle$\}$. Then, I and R are finite sets (Lemma 7.1) and $I = \sim I_2 \subseteq \sim X^*$. Let us denote $C = C(SC_{PA})$ and let us prove that $C = \sim X^*$.

Firstly, we will show that $^{\sim}X^* \subseteq C$. Let $^{\sim}y \in {}^{\sim}X^*$. By induction on $|y|$, we prove that $^{\sim}y \in C$, i.e., $^{\sim}y$ is generated by the splicing system given above. If $^{\sim}y \in I$, then obviously we have $^{\sim}y \in C$.

Thus, suppose $^{\sim}y \notin I$. Since $^{\sim}y \in {}^{\sim}X^*$, each representative of $^{\sim}y$ is the label of a successful path. Furthermore, since $^{\sim}y \notin I$, we have $y = xc^rz$ with $x, z \in A^*$, $c \in A^+$, $\delta(q_0, x) = q$, $\delta(q, c) = q$, $\delta(q, z) \in F$ and one of the following cases occurs:

Conditions (*)

1. $c \neq F(c)$ is a cycle;
2. $c = F(c)$ is a cycle and $r > 1$;
3. c is not a cycle.

Choose x with $|x|$ maximal with respect to conditions $(*)$ (notice that y is a word with a finite length $|y|$; for each prefix x of y, we have $|x| \leq |y|$ and so this maximal length exists).

Let us prove that we cannot have the first case. Indeed, otherwise, $c \neq F(c)$ is a cycle, where we refer to the occurrence of the word c as the cycle c in the successful path y. Thus, by using Definition 7.3, c is not a simple cycle in ($\mathcal{A}$, as a factor of) y and $c = u(c')^tv$, where c' is a cycle in ($\mathcal{A}$, as a factor of) y and either $c' \neq F(c')$ or $c' = F(c')$ and $t > 1$. Furthermore, $u, v \neq 1$ (since $u_1, u_{k+1} \neq 1$ in Definition 7.2). So, $y = x(u(c')^tv)^rz$ with c' satisfying one of the conditions $(*)$ and $|xu| > |x|$, against the maximality of $|x|$.

Let us suppose that the second case occurs, i.e., $c = F(c)$ is a cycle and $r > 1$. Thus, $^{\sim}c \in I$. By using the splicing definition we have $^{\sim}c \in C$. Furthermore $y = xc^rz$ with $r - 1 > 0$. Since y is a successful path, then $y' = xc^{r-1}z$ is also a successful path, and, by using induction hypothesis, $^{\sim}y' \in C$.

Consider the linearization zxc^r of $^{\sim}y$. Set $zxc^r = u_2hu_1u_4ku_3$, with $h = zxc^{r-1}$, $u_1 = u_2 = k = u_3 = 1$, $u_4 = c = F(c)$. Clearly, $u_1|u_2\$u_3|u_4 = 1|1\$1|c \in R$. Moreover, $^{\sim}u_2hu_1 = {}^{\sim}zxc^{r-1} \in C$ and $^{\sim}u_4ku_3 = {}^{\sim}c \in C$: by using the splicing definition we have $^{\sim}zxc^r = {}^{\sim}y \in C$.

Finally, let us suppose that the third case holds, i.e., c is not a cycle. Once again, we have three different cases, described below:

1. The state q is also an internal state which is crossed by the transition $\delta(q,c) = q$ and there exists $c' \in A^+$, $t > 1$ such that $c = (c')^t$, $\delta(q,c') = q$.

2. The state q is also an internal state which is crossed by the transition $\delta(q,c) = q$ and there exists $c', c'' \in A^+$ such that $\delta(q,c') = q = \delta(q,c'')$, $y = xc^rz = x(c'c'')^rz$.

3. The internal states crossed by the transition $\delta(q,c) = q$ are different from q, $c = u(c')^tv$ where $u \in A^+$ is the label of transitions in which no state is crossed twice, $\delta(q,u) = q'$, $\delta(q',c') = q'$, $\delta(q',v) = q$ and c' is not a cycle.

Suppose that the first case holds. Then, $y = x(c')^{tr}z$. If c' is a cycle, then y also satisfies one of the first two conditions $(*)$ and $^\sim y \in C$. Otherwise, c' is not a cycle, $y = xc'(c')^{(t-1)r+r-1}z$ with $(t-1)r + r - 1 > 0$ and c' satisfies one of the conditions $(*)$ with $|xc'| > |x|$, which is against the maximality of $|x|$.

Suppose that the second case holds. Say $c' \in A^+$ the shortest word such that $\delta(q,c') = q$, $c = c'c''$, $c'' \in A^+$. We have that $c' = F(c')$ is a cycle, since otherwise $y = x(c'c'')^rz = x(uc_1vc'')^rz$, where $u \in A^+$ and either c_1 is not a cycle or c_1 is a cycle but $c_1 \neq F(c_1)$. So, c_1 satisfies one of the conditions $(*)$ with $|xu| > |x|$, which is against the maximality of $|x|$. Thus, $y = x(c'c'')^rz = xc'c''(c'c'')^{r-1}z$. Since y is a successful path and $\delta(q,c') = q = \delta(q,c'')$, then $y' = xc''c^{r-1}z$ is also a successful path, and, by using induction hypothesis, $^\sim y' \in C$.

Consider the linearization c^rzx of $^\sim y$. Set $c^rzx = u_2hu_1u_4ku_3$, with $h = c''c^{r-1}zx$, $u_1 = u_2 = k = u_3 = 1$, $u_4 = c'$. Clearly, $u_1|u_2\$u_3|u_4 = 1|1\$1|c' \in R$. Moreover, $^\sim u_2hu_1 = {}^\sim c''c^{r-1}zx \in C$ and $^\sim u_4ku_3 = {}^\sim c' \in C$: by using the splicing definition we have $^\sim c^rzx = {}^\sim y \in C$.

Finally, let us prove that the third case cannot occur. Indeed, otherwise, $y = x(u(c')^tv)^rz$ with c' satisfying one of the conditions $(*)$ and $|xu| > |x|$,

once again against the maximality of $|x|$.

Vice versa, $C \subseteq^{\sim} X^*$ follows by using Proposition 7.1, since $I \subseteq^{\sim} X^*$. ■

Inspired by Theorem 7.1, decidability questions can obviously be asked. Namely, can we decide whether a star language is fingerprint closed? One of the difficulties in answering this question is that in order to prove a star language L does not have this property, we must check all finite automata recognizing L. So, a further development of this work will be to prove (or disprove) that a star language which is fingerprint closed with respect to a finite automaton, is always fingerprint closed with respect to a standard automaton, e.g., the minimal automaton recognizing it.

Example 7.4 We point out that rational languages exist which are not star languages and which are the full linearization of regular circular splicing languages. Indeed, $^{\sim}(ab)^*$ has $(ab)^* \cup (ba)^*$ as the full linearization, and this is not a star language. On the other hand, $^{\sim}(ab)^*$ is a regular circular splicing language since it is generated by $SC_{PA} = (A, I, R)$, where $A = \{a, b\}, I = \{1, ^{\sim}ab\}, R = \{ab|1\$1|ab\}$. Indeed, for each $n > 1$, $w = {}^{\sim}(ab)^n$ can be obtained by using the rule in R from $^{\sim}ab$ and $^{\sim}(ab)^{n-1}$: an induction over n shows that $L \subseteq C(SC_{PA})$. Once again, using induction over $|w|$, we prove that $C(SC_{PA}) \subseteq L$: if w is obtained by using the rule in R from w_1, w_2, since $|w_1| < |w|$, $|w_2| < |w|$, by induction $w_1, w_2 \in {}^{\sim}(ab)^*$. Thus, $w_1 = {}^{\sim}hab, w_2 = {}^{\sim}kab$ imply $h, k \in (ab)^*$. Finally, $w \in {}^{\sim}(ab)^*$.

Example 7.5 Star languages exist which do not satisfy the hypothesis contained in Theorem 7.1. For instance, consider $L = A^* \setminus a^+ = A^*bA^* = (a^*ba^*)^*$ over a two-letter alphabet $A = \{a, b\}$. It is clear that L is a star language. Furthermore, consider the trim deterministic automaton $\mathcal{A}$ recognizing L and depicted in Figure 7.3. We see that a is a cycle in $\mathcal{A}$ and $a^+ \cap L = \emptyset$. Now, let $\mathcal{A}$ be a trim deterministic automaton recognizing X^*. Let k be a positive integer greater than the number of the states in $\mathcal{A}$ and consider $ba^k \in L$. It is easy to see that there exists j such that $j \leq k$ and a^j is a cycle in $\mathcal{A}$. Thus, L is not a fingerprint closed language with respect to $\mathcal{A}$. However, L is not C_{PA} generated. Indeed, by contradiction, suppose

that $C = {}^{\sim}L = C(SC_{PA})$ with $SC_{PA} = (A, I, R)$. Since ${}^{\sim}(a^*b) \subseteq C$, $n \in \boldsymbol{N}$ exists such that $w = {}^{\sim}(a^n b) \in C \setminus I$. Thus, ${}^{\sim}hu_1u_2 \in C$, ${}^{\sim}ku_3u_4 \in C$ also exist such that $w = {}^{\sim}u_2hu_1u_4ku_3$. Since we have only one occurrence of b in w, we get ${}^{\sim}hu_1u_2 \in {}^{\sim}a^*$ or ${}^{\sim}ku_3u_4 \in {}^{\sim}a^*$, which is a contradiction.

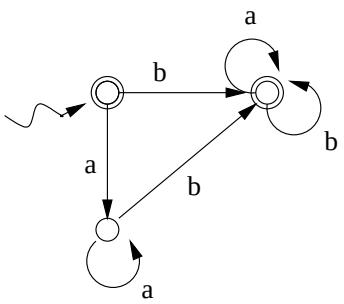

Figure 7.3: *Automaton for* $(a^*ba^*)^*$.

Example 7.6 On the contrary, ${}^{\sim}(a^*ba^*)^*$ is C_{PI} generated by $SC_{PI} = (A, I, R)$, with $I = \{{}^{\sim}b, {}^{\sim}ab, {}^{\sim}aab\} \cup 1$ and $R = \{(b, a; b), (a, b; a), (b, b; b), (a, a; a), (ab, b; aa), (b, ab; b)\}$. Indeed, denote $L = (a^*ba^*)^*$ and $r_1 = (b, a; b)$, $r_2 = (a, b; a), r_3 = (b, b; b)$, $r_4 = (a, a; a), r_5 = (ab, b; aa), r_6 = (b, ab; b)$. We immediately observe that we must have $r_2 = \overline{r}_1$, $r_3 = \overline{r}_3$, $r_4 = \overline{r}_4$, $r_6 = \overline{r}_5$, following the notation in Pixton's definition. Let us prove that $C(SC_{PI}) \subseteq {}^{\sim}L$ by using induction on the length of the words. Indeed, looking at the form of r_1, r_2, r_3, r_4, we see that using these rules and starting from two words ${}^{\sim}\epsilon\alpha$, ${}^{\sim}\epsilon'\alpha'$ in ${}^{\sim}L$ we have a word ${}^{\sim}\epsilon\beta\epsilon'\beta' = {}^{\sim}\epsilon\alpha\epsilon'\alpha'$ which is again in ${}^{\sim}L$, since L is a submonoid closed under conjugacy relation. Otherwise, if $w = {}^{\sim}\epsilon\beta\epsilon'\beta'$ is obtained by using r_5, r_6 and ${}^{\sim}\epsilon\alpha$, ${}^{\sim}\epsilon'\alpha'$ in ${}^{\sim}L$, we have that $\alpha = ab, \beta = aa$, $\alpha' = b = \beta'$, $\epsilon, \epsilon' \in A^*$. Thus, ${}^{\sim}\epsilon\beta\epsilon'\beta' = {}^{\sim}\epsilon aa\epsilon' b \in {}^{\sim}L = {}^{\sim}(A^* \setminus a^+)$.

Once again we prove that ${}^{\sim}L \subseteq C(SC_{PI})$ by using induction over $|w|$, $w \in {}^{\sim}L$, and we distinguish two cases: $|w|_b = 1$ and $|w|_b > 1$. Now, if $|w|_b = 1$, then $w = {}^{\sim}a^t ba^{t'}$, $t \geq 2$, $t' \geq 1$ (i.e., $t + t' \geq 3$, otherwise, $w \in I$). Thus, w can be obtained from $w_1 = {}^{\sim}aba^{t-2} \in {}^{\sim}L$, $w_2 = {}^{\sim}ba^{t'} \in {}^{\sim}L$ by using $r_5, \overline{r}_5$.

If $|w|_b = m > 1$, then clearly we can write $w = w_1 w_2$, where $|w_1|_b = m_1$, $|w_2|_b = m_2$, $m_1 + m_2 = m$ and $w_1, w_2 \in^{\sim} L$. Now, we can see that rules $r_1, \overline{r}_1$ or $r_3, \overline{r}_3$ or $r_4, \overline{r}_4$ allow us to produce w from w_1, w_2.

A remarkable class of fingerprint closed star languages is given in the next section.

7.3 Circular splicing and codes

As far as we know, the structure of regular languages which are closed under the conjugacy relation is unknown, even for the simple case of languages which are the Kleene closure of a regular language. Nevertheless, this structure has been completely described in [4, 67] when X^* is a free monoid. The necessary definitions for recalling this result can be found in [3]. We briefly report them below.

An algebraic description of some subclasses of the class of the regular languages $L \subseteq A^*$ has been given by means of the *syntactic monoid* $\mathcal{M}(L)$ of L, defined in Section 1.1. Historically, this notion arose in the context of variable-length codes. We recall that X^* is a free monoid if and only if X is a *code*, i.e. for all $x_1, \ldots, x_n, x'_1, \ldots, x'_m \in X$, we have

$$x_1 \cdots x_n = x'_1 \cdots x'_m \Leftrightarrow n = m \text{ and } \forall i \in \{1, \ldots, n\}\ x_i = x'_i$$

A remarkable class of codes is the class of *biprefix codes* C. $C \subseteq A^*$ is biprefix if no word in C is a proper prefix or a proper suffix of another element in C, i.e., $C \cap CA^+ = C \cap A^+C = \emptyset$. For instance, uniform codes $A^d, d \geq 1$, are biprefix codes.

Here we report two known results on codes and conjugacy. Theorem 7.2 will be used to prove Proposition 7.2. Theorem 7.3 completely describes finite group codes.

Theorem 7.2 *[4, 67] Let $X \subseteq A^+$ be a code. Then X^* is closed under conjugacy relation if and only if $\mathcal{M}(X^*)$ is a group.*

We note that, under the hypothesis that X is a regular language, $\mathcal{M}(X^*)$ is a group if and only if X is a *group code*, i.e., 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 (see [3, 4] for this result). Group codes are biprefix codes.

Theorem 7.3 *[4, 67] Let $X \subseteq A^+$ be a group code. X is finite if and only if $X = A^d$, $d \geq 1$.*

Example 7.7 *[3]* Group codes exist which are not finite. Let $A = \{a, b\}$ and $M_a = (b^*(ab^*a)^*)^*$ be the set of words with an even number of a's. M_a is generated by $X_a = b \cup ab^*a$, which is a (biprefix) code. Then $M_a = X_a^*$ is a free monoid. Moreover, X_a is a regular group code, where $X_a^* = \phi^{-1}(0)$, $\phi\colon A^* \to \boldsymbol{Z}\ /2\boldsymbol{Z}$ is the morphism given by $\phi(a) = 1, \phi(b) = 0$ and with $\{0\}$ a subgroup of $(\boldsymbol{Z}/2\boldsymbol{Z}, +)$. Observe that for the submonoid $X^* = \{w \in A^* || w|_a = |w|_b\}$ composed of the words in A^* having as many a's as b's, we have $X^* = \phi^{-1}(0)$ where $\phi\colon A^* \to \boldsymbol{Z}$ is such that $\phi(a) = 1, \phi(b) = -1$, and with $\{0\}$ a subgroup of $(\boldsymbol{Z}, +)$. X is called a *Dyck code* over A and it is a (non regular) group code.

Remark 7.1 We can observe that each regular language closed under conjugacy is contained in a star language generated by a regular group code which is minimal with respect to inclusion. Indeed, let L be a regular language which is closed under conjugacy. A^* is a free monoid which is regular, closed under conjugacy and which contains L. Thus, the intersection of all regular free monoids which contain L and are closed under conjugacy, is a free monoid containing L, closed under conjugacy (thus generated by a regular group code) and minimal with respect to inclusion.

Finally, Proposition 7.2 shows that the hypothesis contained in Theorem 7.1 is satisfied by regular group codes.

Proposition 7.2 *Let X be a regular group code and let $\mathcal{A}$ be a trim deterministic finite state automaton recognizing X^*. For any cycle c in $\mathcal{A}$, $c \in X^*$.*

Proof. Let X be a regular group code and let $\mathcal{A}$ be a trim deterministic finite state automaton recognizing X^*. Let c be a cycle in $\mathcal{A}$. Then, $x, y \in A^*$

exist such that $xcy \in X^*$ and $xy \in X^*$. Moreover, since X^* is closed under conjugacy, we have $yxc, yx \in X^*$ and, since X^* is biunitary, we have $c \in X^*$. We recall that X is a biprefix code if and only if X^* is biunitary, i.e., X^* is both left and right unitary, where a submonoid X^* is right (resp. left) unitary if, for all $u, v \in A^*$, $u, uv \in X^*$ (resp. $u, vu \in X^*$) implies $v \in X^*$ [3]. ■

Remark 7.2 Thanks to Proposition 7.2, for each regular group code X, X^* is fingerprint closed with respect to each trim deterministic finite state automaton $\mathcal{A}$ recognizing X^*. The above property allows us to state that the star language, not fingerprint closed, $(a^*ba^*)^*$, cannot be generated by a regular group code, otherwise $a \in L$ (see Example 7.5). This property will be very useful in the last chapter.

Corollary 7.1 *For each regular group code X, $^\sim X^*$ is a circular splicing language.*

Example 6.3 *(continued).* Consider $L_1 = \{w \in A^* \mid \exists h, k \in \boldsymbol{N} \quad |w|_a = 2k, |w|_b = 2h\}$. Obviously, $L_1 = M_a \cap M_b$, where M_a, M_b have been defined in Example 7.7. Thus L_1 is a free monoid which is closed under conjugacy, since it is the intersection of two free monoids both closed under conjugacy. We have already observed that L_1 is a regular language. Furthermore, since L_1 is a free monoid, it follows that $L_1 = X^*$, where X is a code [3]. Moreover, it is well known that when X is a code, X^* is regular if and only if X is regular [3]. Since X is a code which is regular and $X^* = L_1$ is closed under conjugacy relation, by using Theorem 7.2 we have that X is a group code. Consequently, $L_1 = X^*$ is a star language and $^\sim L_1 = {}^\sim X^*$ is C_{PA} generated, thanks to Corollary 7.1.

The splicing system $SC_{PA} = (A, I, R)$ generates $^\sim L_1$, when we choose $I = \{^\sim aa, ^\sim bb, ^\sim abab\} \cup 1$ and $R = \{aa|1\$1|1, bb|1\$1|1, ab|1\$1|1\}$.

Let us prove that $^\sim L_1 = C(SC_{PA})$. Since L_1 is a star language, we have that $C(SC_{PA}) \subseteq {}^\sim L_1$, by using Proposition 7.1. Vice versa, we show that every $w \in {}^\sim L_1$ is in $C(SC_{PA})$, by using induction on $|w| = n$. We obviously have $w \in {}^\sim L_1 \cap I \subseteq C(SC_{PA})$. Then, suppose that $w \in {}^\sim L_1 \setminus I$ and so

$|w|_a = 2k, |w|_b = 2h$, for $h, k \in \boldsymbol{N}$. If aa is a factor of w then $w = {}^{\sim}w_1 aa w_2$ with $|w_1|_a + |w_2|_a = |w|_a - 2 = 2k-2$, $|w_1|_b + |w_2|_b = |w|_b = 2h$ and we can suppose $w_1, w_2 \in A^+$ (we cannot have $w_1 w_2 \in A \cup 1$). Thus, ${}^{\sim}w_2 w_1 \in {}^{\sim}L_1$ and $|w_2 w_1| < |w|$: by using induction hypothesis, ${}^{\sim}w_2 w_1 \in C(SC_{PA})$. We also have ${}^{\sim}aa \in I \subseteq C(SC_{PA})$. So, the rule $aa|1\$1|1$, applied to ${}^{\sim}w_2 w_1$ and ${}^{\sim}aa$, generates w. The case in which bb is a factor of w is symmetric to the case above and can be handled with a similar argument.

If neither aa nor bb are factors of w, then $w = {}^{\sim}(ab)^{2k}$ with $k > 1$. On the other hand ${}^{\sim}abab, {}^{\sim}(ab)^{2k-2} \in {}^{\sim}L_1$. we have ${}^{\sim}abab \in I \subseteq C(SC_{PA})$ and, by using induction hypothesis, ${}^{\sim}(ab)^{2(k-1)}$ is in $C(SC_{PA})$: the rule $ab|1\$1|1$, applied to ${}^{\sim}(ab)^{2(k-1)}$ and ${}^{\sim}abab$, generates w.

In the proof of Theorem 7.1, we have given the construction of a splicing system SC_{PA} which generates a fingerprint closed star language. Thanks to Remark 7.2, L_1 is fingerprint closed with respect to each trim deterministic finite state automaton $\mathcal{A}$ recognizing X^*. Now, let us illustrate the above-mentioned construction for L_1. Consider the automaton $\mathcal{A}$ which recognizes L_1 depicted in Figure 7.4.

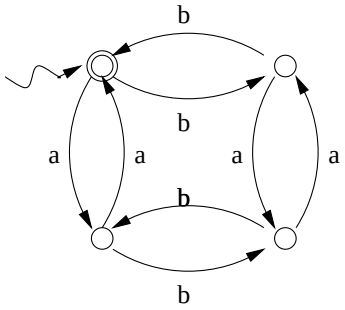

Figure 7.4: *Automaton for* L_1.

Thus, we have $\mathcal{C} = \bigcup_{c\ cycle} F(c) = \{abaaba,\ aa,\ abba,\ babbab,\ bb,\ baab,\ abbbaaab,\ abab,\ abbbab,\ abaaab,\ baaabbba,\ baba,\ baaaba,\ babbba\}$.

Finally, we obtain the circular splicing system $SC_{PA} = (A, I', R')$ generating ${}^{\sim}L_1$ with $R' = \{1|1\$1|\alpha \mid \alpha \in \mathcal{C}\}$ and $I' = 1 \cup {}^{\sim}\{aa, abba, abaaba,\ bb, baab, babbab,\ abab, abbbab, abaaab,\ baba, baaaba, babbba\ \}$. Observe that

$|I'| > |I|$ and $|R'| > |R|$.

As we have already said, we now prove that all star languages X^* with X a finite set have their circularization in $C(Fin, Fin)$. In the proof of this result, we also give a splicing system generating $^\sim X^*$ the construction of which is simpler than the one given in the proof of Theorem 7.1.

Proposition 7.3 *For all finite sets X, Y such that $X^* \cup Y$ is closed under the conjugacy relation and with $X \cap Fact(Y) = \emptyset$, we have that $^\sim(X^* \cup Y)$ (resp. $^\sim(X^+ \cup Y)$) is a circular splicing language.*

Proof. It is clear that the splicing system $SC_{PA} = (A, I, R)$ generates $^\sim(X^* \cup Y)$, when we choose $I = \{^\sim x \mid x \in X \cup Y\} \cup 1$ (resp. $I = \{^\sim x \mid x \in X \cup Y\}$) and $R = \{x_i|1\$1|x_j \mid x_i, x_j \in X\}$. Indeed, no rule in R can be applied to words w_1, w_2 with $w_1 \in {}^\sim Y$ or $w_2 \in {}^\sim Y$. Furthermore, $^\sim Y \subseteq I \subseteq C(SC_{PA})$. Finally, starting with $^\sim X$, rules in R generate $^\sim X^*$. ■

All the examples of star languages X^* which are fingerprint closed and which we have reported above, are such that X is a (regular group) code. As a result, the problems below arise.

Problem 7.1 *Do star languages X^* exist which do not satisfy the hypothesis contained in Theorem 7.1 (i.e., which are not fingerprint closed with respect to $\mathcal{A}$, for all trim deterministic finite automaton $\mathcal{A}$), with X being not a finite set and which are C_{PA} generated ?*

Problem 7.2 *Are there fingerprint closed star languages X^* which are not generated by a finite set or by a regular group code X ?*

Chapter 8

The case of one-letter alphabet and uniform languages

In this section, we will consider the special case of a one-letter alphabet $A = \{a\}$. Here, 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. Using Lemmata 8.1 and 8.3 as preliminary results, we give a characterization of $Reg^{\sim} \cap C(Fin, Fin)$ on one-letter alphabet, and so we provide an answer to Problem 5.1 in this particular case (Proposition 8.1). Observe that circular splicing systems with an initial finite set and an empty set R of rules generates only finite sets. This particular situation is not significant for a characterization of $Reg^{\sim} \cap C(Fin, Fin)$, so in the next part of this chapter we will often suppose $R \neq \emptyset$. As usual we will refer to Paun systems, but we also make some observations regarding circular Pixton splicing systems (Remark 8.2). In our knowledge, no result is given in the case of one-letter alphabet. We have only reported, in Section 1.3, a characterization with respect to SH systems. Here we give a necessary and sufficient condition for circular Paun finite splicing systems.

8.1 The characterization

The following lemma allows us to understand the behaviour of a rule for a language on one-letter alphabet. Indeed, we will see that only the lengths of the words are involved.

Lemma 8.1 *Let $SC_{PA} = (A, I, R)$ be a circular splicing system with $A = \{a\}$, $R = \{r_1, \ldots, r_t\}$ and $r_i = a^{m_{1,i}}|a^{m_{2,i}}\$a^{m_{3,i}}|a^{m_{4,i}}$, for $i \in \{1, \ldots, t\}$. For all $i \in \{1, \ldots, t\}$, for all $x, y \in a^*$, with $|x| \geq m_{1,i} + m_{2,i}$, $|y| \geq m_{3,i} + m_{4,i}$, the circular word $^\sim xy$ is generated by means of the rule r_i and starting from $^\sim x$, $^\sim y$.*

Proof. The conclusion follows by using Paun's definition of circular splicing in the special case $A = \{a\}$. ■

We will use some notations and elementary notions from group theory recalled below (see for instance [28]).

Consider the *cyclic group* of order n which we realize, as usual, as the quotient group $\boldsymbol{Z}_n$ of the integers modulo n (i.e., the group of the congruence classes modulo n). For subsets T_1, T_2, T of $\boldsymbol{N}$ and $n \in \boldsymbol{N}$, $T_1 + T_2 = T \pmod{n}$ means that for any $t \in T$ a unique pair (t_1, t_2) exists with $t_1 \in T_1$, $t_2 \in T_2$ and $t_1 + t_2 = t \pmod{n}$. When $T = \{0, \ldots, n-1\} = \boldsymbol{Z}_n$, the pair (T_1, T_2) is called a *factorization* of $\boldsymbol{Z}_n$ (see [28] for definitions and main results and [23] for a survey on this argument).

T is a *periodic* subset of $\boldsymbol{Z}_n$ if $t \in \boldsymbol{Z}_n$ exists with $t + T = T \pmod{n}$. Each subgroup G of $\boldsymbol{Z}_n$ is periodic (since $0 \in G$), but periodic subsets exist which are not subgroups. For instance, $\{0, 2, 4\}$ and $\{1, 3, 5\}$ are both periodic subsets of $\boldsymbol{Z}_6$ ($\{0, 2, 4\} + 2 = \{0, 2, 4\} \pmod 6$ and $\{1, 3, 5\} + 2 = \{1, 3, 5\} \pmod 6$), but the former is a subgroup of $\boldsymbol{Z}_6$ whereas the latter is not a subgroup ($5 + 1 \notin \{1, 3, 5\}$). It is well known that G is a subgroup of $\boldsymbol{Z}_n$ if and only if there exists a divisor k of n such that $a^G = \{a^{tk} \mid t \in \{0, \ldots, n/k - 1\}\}$ is the cyclic group of order n/k generated by a^k. For a subset G of $\boldsymbol{N}$, we also set $a^G = \{a^g \mid g \in G\}$. In the following result we have reported the well known structure of the subgroups of $\boldsymbol{Z}_n$, used in the next of this section.

Lemma 8.2 *Let G' be a subgroup of $\boldsymbol{Z}_n$ of period p. Then, set $n = pr$, we have $G' = \{0 + pk \mid 0 \leq k \leq r-1\}$. Furthermore, for each $t \in G'$, $G' = \{t + pk \mid 0 \leq k \leq r-1\} (\mathrm{mod}\ n)$.*

The Lemma below is fundamental for our characterization.

Lemma 8.3 *Let $SC_{PA} = (A, I, R)$ be a circular splicing system with $A = \{a\}$, $R = \{r_1, \ldots, r_t\}$ and $r_i = a^{m_{1,i}}|a^{m_{2,i}}\$a^{m_{3,i}}|a^{m_{4,i}}$, for $i \in \{1, \ldots, t\}$. Denote $m = max\ \{m_{1,i} + m_{2,i},\ m_{3,i} + m_{4,i} \mid i \in \{1, \ldots, t\}\}$, $L = C(SC_{PA})$ and $L_2 = \{w \in L \mid |w| \geq m\}$. Then, $L = L_1 \cup L_2$ where $L_1 = L \setminus L_2$ is a finite subset of a^*. Furthermore, either $L_2 = \emptyset$ or there exists $n \in \boldsymbol{N}$ such that $L_2 = (a^G)^+$, where G is a set of representatives of the elements in a (periodic) subgroup of $\boldsymbol{Z}_n$ and $a^G \cap Fact(L_1) = \emptyset$.*

Proof. Let us denote $m = max\ \{m_{1,i} + m_{2,i},\ m_{3,i} + m_{4,i} \mid i \in \{1, \ldots, t\}\}$, $L = C(SC_{PA})$ and $L_2 = \{w \in L \mid |w| \geq m\}$. Then, we obviously have $L = L_1 \cup L_2$ where $L_1 = L \setminus L_2$ is a finite subset of a^*.
If $L_2 = \emptyset$, the proof is concluded. Otherwise, let $n \in \boldsymbol{N}$ with $n = min\{|l| \mid l \in L_2\}$. For all $j \in \{0, \ldots, n-1\}$ such that $(a^n)^* a^j \cap L_2 \neq \emptyset$, let $h_j = min\{h \mid a^{nh+j} \in L_2\}$. We have already observed that $h_0 = 1$ and $h_j > 0$, for all j's. Let $G \subseteq \boldsymbol{N}$ with $G = \{h_j n + j \mid j \in \{0, \ldots, n-1\}, (a^n)^* a^j \cap L_2 \neq \emptyset\}$.
Let us show that $L_2 = (a^G)^+$. We show that $(a^G)^+ \subseteq L_2$, by using induction on the length of $a^g \in (a^G)^+$. Indeed, either $a^g \in a^G \subseteq L_2$ or $a^g = a^{g_1} a^{g_2}$ with $a^{g_1} \in (a^G)^+$, $a^{g_2} \in (a^G)^+$ and $|a^{g_1}| < |a^g|$, $|a^{g_2}| < |a^g|$. In the last case, by using induction hypothesis, $a^{g_1}, a^{g_2} \in L_2$ and, by using Lemma 8.1, $a^g = a^{g_1} a^{g_2} \in L_2$. So, $(a^G)^+ \subseteq L_2$. In order to prove that $L_2 \subseteq (a^G)^+$, let $a^l \in L_2$ and let h, j be such that $l = nh + j$, with $h \in \boldsymbol{N}$ and $j \in \{0, \ldots, n-1\}$. We already know that $h \geq h_j$. Since $a^{h_j n + j} \in (a^G)^+$ and $a^n \in (a^G)^+$, we have $a^{h_j n + j}(a^n)^* \subseteq (a^G)^+$ and so $a^l = a^{nh+j} \in a^{h_j n + j}(a^n)^* \subseteq (a^G)^+$. Clearly we have $a^G \cap Fact(L_1) = \emptyset$.
Finally, let us prove that G is a set of representatives of the elements of a subset G' of $\boldsymbol{Z}_n$ where G' is a (periodic) subgroup of the finite cyclic group of order n. Thus, G' is defined as the set of the (different) classes

modulo n of the elements in G. We already know that $n \in G$. Furthermore, for each $g_1, g_2 \in G$, we have $a^{g_1}a^{g_2} \in (a^G)^+ = L_2$. Thus, for $h \in \boldsymbol{N}$ and $k \in \{0, \ldots, n-1\}$ such that $g_1 + g_2 = hn + k$, we have $h \geq h_k$ with $h_k n + k \in G$. Thus G' is a submonoid of a finite group (namely $\boldsymbol{Z}_n$): a classical result from group theory states that G' is a subgroup (of $\boldsymbol{Z}_n$, indeed for each $g \in G'$, G' being a submonoid, we have $(n-1)g \in G'$ and $(n-1)g$ is the inverse of g). ■

Proposition 8.1 *A subset L of a^* is C_{PA} generated if and only if there exist 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 $a^G \cap Fact(L_1) = \emptyset$.*

Proof. Thanks to Proposition 7.3 and to Lemma 8.3, the conclusion easily follows. ■

A simpler characterization of the C_{PA} generated circular languages over $\{a\}$ is given below.

Corollary 8.1 *A subset L of a^* is C_{PA} generated if and only if there exist finite subsets L_1, a^G of a^* and a positive integer n such that $L = L_1 \cup (a^G)^+$ and, when $a^G \neq \emptyset$, $max\{|l| \mid l \in L_1\} < n = min\{|l| \mid l \in a^G\}$.*

Proof. Thanks to Propositions 7.3 and 8.1, the conclusion easily follows. ■

Example 8.1 Let $L = \{a^3, a^4\} \cup \{a^6, a^{14}, a^{16}\}^+$. Then L satisfies the hypothesis contained in Proposition 8.1. According to Proposition 7.3, L is generated by the splicing system $SC_{PA} = (\{a\}, I, R)$, where $I = \{a^3, a^4, a^6, a^{14}, a^{16}\}$ and $R = \{a^6|1\$1|a^6,\ a^6|1\$1|a^{14},\ a^6|1\$1|a^{16},\ a^{14}|1\$1|a^{14},\ a^{14}|1\$1|a^{16},\ a^{16}|1\$1|a^{16}\ \}$.

We end this section with some observations. We begin with a result concerning the descriptional complexity of a circular splicing system which generates a circular language $L \subseteq a^*$ (Proposition 8.2). We then make a comparison between the two cases $|A| = 1$ and $|A| > 1$, with respect to Problem 5.1 for Paun systems (Remark 8.1). Comments on Pixton systems with $|A| = 1$ are given in Remark 8.2.

Proposition 8.2 *Let $L \subseteq a^*$ be a C_{PA} generated language. Then, there exists a (minimal) splicing system (A, I, R) generating L with either $R = \emptyset$ or $R = \{r\}$ containing only one rule.*

Proof. Let $L \subseteq a^*$ be a C_{PA} generated language. Then, using Corollary 8.1 and with the same notations, $L = L_1 \cup (a^G)^+$ and a positive integer n exists such that, when $a^G \neq \emptyset$, $n = min\{|l| \mid l \in a^G\}$, $L_1 = \{l \in L \mid |l| < n\}$. If $a^G = \emptyset$ then the circular splicing system (A, I, R) where $R = \emptyset$ and $I = L_1$ obviously generates $L = L_1$.
Thus, suppose that $a^G \neq \emptyset$. Consider the circular splicing system (A, I, R) where $R = \{a^n \mid 1 \$ 1 \mid a^n\}$ and $I = L_1 \cup a^G$. Obviously, I is a finite set and $I \subseteq L$.
Let us prove that $L = C(SC_{PA})$. We show that $L \subseteq C(SC_{PA})$. Clearly, $I \subseteq C(SC_{PA})$ and, for each $a^g \in L \setminus I$, we prove that $a^g \in C(SC_{PA})$ by using induction on $|a^g|$. Since $a^g \in L \setminus I$, then $a^g \in (a^G)^+ \setminus a^G$ which in turn implies $a^g = a^{g_1}a^{g_2}$ with $a^{g_1} \in (a^G)^+$, $a^{g_2} \in (a^G)^+$, $|a^{g_1}| < |a^g|$, $|a^{g_2}| < |a^g|$. Furthermore, $a^{g_1} \in (a^G)^+$ implies $g_1 \geq n$ and, analogously, $a^{g_2} \in (a^G)^+$ implies $g_2 \geq n$. By using induction hypothesis, $a^{g_1}, a^{g_2} \in C(SC_{PA})$ and, by using Lemma 8.1, $a^g = a^{g_1}a^{g_2} \in C(SC_{PA})$.

We now prove that $C(SC_{PA}) \subseteq L$ with a slight variation of the argument contained in Proposition 7.1 and by using induction on the length of $a^l \in C(SC_{PA})$. Let $a^l \in C(SC_{PA})$ and let h, j be such that $l = nh + j$, with $h \in \boldsymbol{N}$ and $j \in \{0, \ldots, n-1\}$. Since $I \subseteq L$, we can suppose that a^l has been generated by using the rule in R and starting from two words $a^{nh_1+j_1}, a^{nh_2+j_2} \in C(SC_{PA})$ of shorter length than a^l (since they are different from a^l). Then, $h_1 \geq 1$, $h_2 \geq 1$ (otherwise the rule in R cannot be applied) and $a^l = a^{nh_1+j_1}a^{nh_2+j_2}$. By using induction hypothesis, $a^{nh_1+j_1}$, $a^{nh_2+j_2} \in L$. Then $a^{nh_1+j_1}, a^{nh_2+j_2} \in (a^G)^+$ (since $h_1, h_2 \geq 1$), and thus $a^l = a^{nh_1+j_1}a^{nh_2+j_2} \in (a^G)^+$, since $(a^G)^+$ is a submonoid. ■

Example 8.1 *(continued)* In virtue of Proposition 8.2, $L = \{a^3, a^4\} \cup \{a^6, a^{14}, a^{16}\}^+$ has a minimal circular splicing system which generates it, namely, $SC_{PA} = (\{a\}, I, R)$ where $I = \{a^3, a^4, a^6, a^{14}, a^{16}\}$ and $R = \{a^n|1\$ 1|a^n\} = \{a^6|1\$1|a^6\}$. Looking at the proof of Proposition 8.2, and us-

ing the same argument as in this proof, we can see that another minimal splicing system can be defined, namely, $SC_{PA} = (\{a\}, I, R)$, where $I = \{a^3, a^4, a^6, a^{12}, a^{14}, a^{16}, a^{18}, a^{20}, a^{22}\}$ and $R = \{a^{12}|1\$1|a^{12}\}$. As a matter of fact, if L satisfies the hypothesis of Proposition 8.1, for every $h \geq 1$, the circular splicing system $(\{a\}, I_h, R_h)$, where $R_h = \{a^{hn}|1\$1|a^{hn}\}$, generates L if we choose a sufficiently large I_h.

Example 8.2 As another example, Let us consider the rational language $L = \{a^3, a^7\} \cup \{a^4, a^5\}^+$. We can see that even if L is expressed in a form that does not correspond to the hypothesis of Proposition 8.1, L is C_{PA} generated by choosing $SC_{PA} = (\{a\}, I, R)$, where $I = \{a^3, a^4, a^5, a^7, a^8, a^9, a^{10}, a^{12}, a^{13}, a^{14}, a^{15}\}$ and $R = \{a^8|1\$1|a^8\}$. Furthermore, following the proof of Lemma 8.3, we see that L can be written as $\{a^3, a^4, a^5, a^7\} \cup \{a^8, a^9, a^{10}, a^{12}, a^{13}, a^{14}, a^{15}\}^+ = \{a^3, a^4, a^5, a^7\} \cup \{a^8, a^9, a^{10}, a^{12}, a^{13}, a^{14}, a^{15}, a^{19}\}^+$ and in this form, L satisfies the hypothesis of Proposition 8.1.

We close this section with some remarks and a generalization of the above characterization.

Remark 8.1 The computational power of Paun circular splicing systems dramatically decreases when we restrict ourselves to alphabets of cardinality one. Indeed, in Proposition 8.1, we have proven that in this case $Reg^{\sim} \cap C(Fin, Fin) = C(Fin, Fin)$ is the class of the languages L satisfying the hypotheses of Proposition 7.3, that is, with the form $L = L_1 \cup (a^G)^+$, G being a finite set and $a^G \cap Fact(L_1) = \emptyset$. We believe that these languages L are fingerprint closed (with respect to a trim deterministic automaton recognizing L). Furthermore, with the same notations as in Proposition 8.1 and in virtue of Theorem 7.3, $(a^G)^*$ is generated by a group code if and only if $a^G = a^n$, i.e., $(a^n)^* a^j \cap (a^G)^+ = \emptyset$, for all $j \in \{1, \ldots, n-1\}$.

Remark 8.2 Languages exist which are C_{PI} generated but not C_{PA} generated even in the case of a one-letter alphabet (thus showing already in this case that Problem 6.1 does not have a positive answer). Indeed, we will see that $cyclic(aa) = (a^2)^* a$ is C_{PI} generated (Proposition 9.6) but $cyclic(aa)$

is not C_{PA} generated (Example 9.2). As in the case of a Paun system each rule in a Pixton system can be applied to a pair of words x, y provided that $|x|, |y|$ are large enough.

8.2 The case of regular languages on one-letter alphabet is decidable

A decidability question naturally follows on from the last example: given a language L, with $L \subseteq a^*$, 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, thanks to the Rice theorem [40]. On the contrary, this question is decidable when we restrict ourselves to regular languages [2]. We will see that we can decide whether a regular language $L \subseteq a^*$ is C_{PA} generated or not by means of properties of the corresponding automaton recognizing L.

In other words, we prove that the following question is decidable.

Problem 8.1 *Given a regular language L, with $L \subseteq a^*$, can we decide whether L is a C_{PA} generated language?*

As a matter of fact, we will suppose that $L \subseteq a^*$ is given by means of the minimal finite state automaton $\mathcal{A}$, which will be the instance of our Problem 8.1. Thus, in Proposition 8.5 we give two conditions: 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.

We have reported in Proposition 8.3 the characterization given in the previous section sligthly modified with respect to the statement proved there. As a matter of fact this modification concerns the expression of G and can be immediately obtained by reading the proof of Proposition 8.1.

Proposition 8.3 *A subset $L = {}^{\sim}L$ of a^* is C_{PA} generated if and only if there exist a finite subset L_1 of a^*, a positive integer n and a (periodic)*

[2] result presented at WORDS01

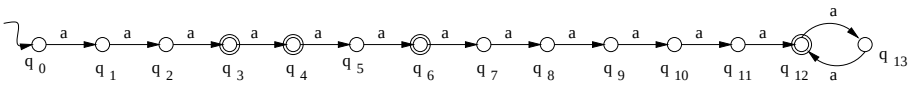

Figure 8.1: *Automaton for* $L = \{a^3, a^4\} \cup \{a^6, a^{14}, a^{16}\}^+$.

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* $\lambda_m = min\{\mu \mid a^{m+\mu n} \in (a^G)^+\}$.

Given a regular language $L \subseteq a^*$, well-known results describe the structure of every deterministic finite state automaton $\mathcal{A}$= $(\{a\}, \{q_0, q_1, \ldots, q_v\}, \delta, q_0, F)$ recognizing L. The transition function δ is defined as follows: for each $i \in \{0, \ldots v-1\}$ we have $\delta(q_i, a) = q_{i+1}$ and there exists $l \in \{0, \ldots, v-1\}$ such that $\delta(q_v, a) = q_l$. In other words, the transition graph of $\mathcal{A}$ has a (frying-)pan shape: the path crossing the states $q_0, \ldots, q_l$ is the handle of the pan, the closed path crossing the states $q_l, \ldots, q_v$ is the pan without the handle. As an example consider the regular C_{PA} generated language $L = \{a^3, a^4\} \cup \{a^6, a^{14}, a^{16}\}^+$. Two deterministic finite state automata recognizing L are depicted in Figure 8.1 and Figure 8.2.

In the next we use Proposition 8.4 (see [3, 40]). Given a deterministic finite state automaton $\mathcal{A} = (A, Q, \delta, q_0, F)$, for all $q \in Q$, we denote $L_q = \{w \in A^* \mid \delta(q, w) \in F\}$.

Proposition 8.4 *[3, 40] Let* $\mathcal{A} = (A, Q, \delta, q_0, F)$ *be a finite state automaton recognizing the regular language* $L = L(\mathcal{A})$. $\mathcal{A}$ *is minimal if and only if, for every* $q, q' \in Q$, *we have* $L_q \neq L_{q'}$.

We only glance at the conditions reported in Proposition 8.5. 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 $\mathcal{A}$. Furthermore, there exist $p, r \in \boldsymbol{N}$ such that $n = pr$ and p is the length of this closed path.

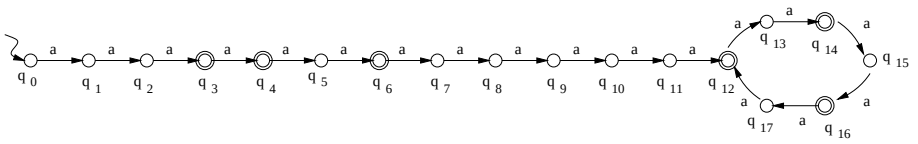

Figure 8.2: Another automaton for $L = \{a^3, a^4\} \cup \{a^6, a^{14}, a^{16}\}^+$

Proposition 8.5 *Let $L \subseteq a^*$ be a regular language. Let $\mathcal{A}$= $(\{a\}, \{q_0, q_1, \ldots, q_v\}, \delta, q_0, F)$ be the minimal finite state automaton recognizing L, where for each $i \in \{0, \ldots v-1\}$ we have $\delta(q_i, a) = q_{i+1}$ and there exists $l \in \{0, \ldots, v-1\}$ 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 $n, p, r \in \boldsymbol{N}$ such that $\mathcal{A}$ satisfies the following two conditions.*

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

Proof. We know that $L \subseteq a^*$ is a finite set if and only if, for each $\mathcal{A}$ recognizing L, we have $q_h \notin F$, for all $h \geq l$. Thus, let us characterize the C_{PA} generated languages which are infinite.

Let us suppose that the two conditions contained in the statement hold for the minimal finite state automaton $\mathcal{A}$ recognizing $L \subseteq a^*$. Let us prove that $L = L(\mathcal{A})$ satisfies the conditions reported in Proposition 8.1, i.e., there exist a finite subset L_1 of a^* and a set G of representatives of the elements in a subgroup G' of $\boldsymbol{Z}_n$ (of period p), such that $L = L_1 \cup (a^G)^+$ and $a^G \cap Fact(L_1) = \emptyset$.

Denote $L_1 = \{a^j \mid \delta(q_0, a^j) = q_{j'} \in F, j' < l\} = \{a^j \mid q_j \in F, j < l\}$. Obviously, L_1 is a finite set and $L_1 \subseteq L(\mathcal{A})$. Let $G' = \{pk \mid 0 \leq k \leq r-1\}$, where p, r are as in condition 2. Using Lemma 8.2, we have that G' is a subgroup of $\boldsymbol{Z}_n$ (of period p). Furthermore, $G = \{n + pk \mid 0 \leq k \leq r-1\}$ is a set of representatives of the elements in G'. Let us prove that $L(\mathcal{A}) \setminus L_1 = (a^G)^+$.

Let $a^j \in L(\mathcal{A}) \setminus L_1$. Then, $\delta(q_0, a^j) = q_n$. Consequently, there exists $k' \in \boldsymbol{N}$ such that $j = n + pk' = m + \mu n$ with $m \in G'$ and $\mu \geq 1$, the last equality being obtained when we divide k' by r. Thus, $a^j \in a^G a^{(\mu-1)n} \subseteq (a^G)^+$. Conversely, let $a^j \in (a^G)^+$. Then, $j = x_1 + x_2 + \ldots x_b$ with $x_i \in G$. So, $x_i \geq n$ and, since G' is a group, we have $j = m + \mu n$ with $m \in G'$ and $\mu \geq 1$. Thus, $j = n + pk + (\mu - 1)n$, with $0 \leq k \leq r - 1$, and $a^j \in L(\mathcal{A}) \setminus L_1$.

Conversely, let L be a subset of a^* which is C_{PA} generated and let us prove that the minimal finite state automaton recognizing L satisfies the two conditions in the statement. This part of the proof is organized as follows: we will construct a finite automaton $\mathcal{A}$ recognizing L. We show that, in two special cases, $\mathcal{A}$ can be modified in order to obtain two other automata $\mathcal{A}_1$, $\mathcal{A}_2$ which once again recognize L. Then, we show that each of the three automata $\mathcal{A}$, $\mathcal{A}_1$, $\mathcal{A}_2$ satisfies the two conditions in the statement. Finally, we prove that either $\mathcal{A}$ or $\mathcal{A}_1$ or $\mathcal{A}_2$ is the minimal finite state automaton recognizing L. Observe that, in order to define any automaton recognizing L, we must define the set Q of the states, the set F of the final states and the state q_l.

Thanks to Proposition 8.1, there exist a finite subset L_1 of a^*, a positive integer n and a subgroup G' of $\boldsymbol{Z}_n$ (of period p) such that $n = pr$ and $L = L_1 \cup (a^G)^+$, where G is a set of representatives of the elements in G' and $a^G \cap Fact(L_1) = \emptyset$.

Set $L_1 = a^D$, $G' = \{m_1, \ldots, m_r\}$ with $0 = m_1 < m_2 < \ldots < m_r$, $G = \{n, m_2 + \lambda_2 n, \ldots, m_r + \lambda_r n\}$ and let λ_t be such that $1 \leq \lambda_t = max\{\lambda_i \mid i \in \{1, \ldots, r\}\}$. Intuitively, the transition diagram of the three automata $\mathcal{A}$, $\mathcal{A}_1$, $\mathcal{A}_2$ is a pan having a closed path of length p and an handle of a length which depends on the chosen automaton. In other words, both $\mathcal{A}_1$ and $\mathcal{A}_2$ can be obtained from $\mathcal{A}$ by shortening the handle in $\mathcal{A}$. Indeed, the handle will have a length $\lambda_t n$ in the transition diagram of $\mathcal{A}$; the handle will have a length $\lambda_t n - s$, $1 \leq s \leq p - 1$, in the transition diagram of $\mathcal{A}_1$; the handle will have a length $m_{t'} - (\lambda_t - 1)n - (p - 1)$, for a suitable $m_{t'} > m_t$, in the transition diagram of $\mathcal{A}_2$.

Definition of $\mathcal{A}$, $\mathcal{A}_1$, $\mathcal{A}_2$.

Precisely, we define $\mathcal{A} = (\{a\}, Q, \delta, q_0, F)$ as follows. We define $Q =$

$\{q_0, q_1, \ldots, q_v\} = \{q_0, q_1, \ldots, q_{\lambda_t n}, q_{\lambda_t n+1}, \ldots q_{\lambda_t n+p-1}\}$ and we set $q_l = q_{\lambda_t n}$. We also define $F = \{q_d \mid d \in D\} \cup \{q_{n+\mu n} \mid 0 \leq \mu < \lambda_t\} \cup \{q_{m_i+\mu n} \mid 2 \leq i \leq r, \lambda_i < \lambda_t, \lambda_i \leq \mu < \lambda_t\}$.

Let us make precise hypotheses, which allow us to deine $\mathcal{A}_1$. Suppose that $p > 1$ and that there exists s, with $1 \leq s \leq p-1$, such that $D \cap \{\lambda_t n - s, \ldots, \lambda_t n - 1\} = \emptyset$. Observe that such s always exists when $\lambda_t > 1$ and in this case, we have $s = p - 1$ (indeed, for all $d \in D$, we have $d < n < 2n - (p-1)$). Let s be maximal with respect to this condition, we modify $\mathcal{A}$ for obtaining $\mathcal{A}_1 = (\{a\}, Q_1, \delta_1, q_0, F_1)$ as follows: we define $Q_1 = \{q_0, q_1, \ldots, q_{\lambda_t n}, q_{\lambda_t n+1}, \ldots q_{\lambda_t n+p-1-s}\}$, δ_1 is the restriction of δ over $Q_1 \times A$, $F_1 = F$ and we set $q_l = q_{\lambda_t n-s}$.

We now give the hypotheses under which we can define $\mathcal{A}_2$, related to the existence of a positive integer t'. Either $\lambda_t > 1$ and that there exists t', with $t < t' \leq r$, such that, for each t_1 with $t' \leq t_1 \leq r$, we have $\lambda_{t_1} \leq \lambda_t - 1$ or $\lambda_t = 1$ and there exists t', with $t < t' \leq r$, such that, for each t_1 with $t' \leq t_1 \leq r$, we have $a^{m_{t'}}, a^{m_{t'+1}}, \ldots, a^{m_r} \in L$. Take t' minimal with respect to these conditions. We modify $\mathcal{A}$ for obtaining $\mathcal{A}_2 = (\{a\}, Q_2, \delta_2, q_0, F_2)$ as follows: we define $Q_2 = \{q_0, q_1, \ldots, q_{m_{t'}+(\lambda_t-1)n}\}$, δ_2 is the restriction of δ over $Q_2 \times A$. We also define $F_2 = F \cap Q_2 = F \setminus (q_{\lambda_t n} \cup \{q_{m_i+(\lambda_t-1)n} \mid i \geq t'\}) = \{q_d \mid d \in D\} \cup \{q_{n+\mu n} \mid 0 \leq \mu < \lambda_t - 1\} \cup \{q_{m_i+\mu n} \mid 2 \leq i \leq t', \lambda_i < \lambda_t, \lambda_i \leq \mu < \lambda_t\} \cup \{q_{m_i+\mu n} \mid t' < i \leq r, \lambda_i < \lambda_t, \lambda_i \leq \mu < \lambda_t - 1\}$. We set $q_l = q_{m_{t'}+(\lambda_t-1)n-(p-1)}$.

Claim: $\mathcal{A}$, $\mathcal{A}_1$ and $\mathcal{A}_2$ satisfy the two conditions in the statement.

Indeed, $\{q_h \mid q_h \in Q,\ h \geq l\} \cap F = \{q_h \mid q_h \in Q_1,\ h \geq l\} \cap F_1 = q_{\lambda_t n}$ and $\{q_h \mid q_h \in Q_2,\ h \geq l\} \cap F_2 = q_{m_{t'}+(\lambda_t-1)n}$ (condition 1). Furthermore, we know that $n = pr$ and $m_{t'} = (t'-1)p$ (Lemma 8.2). Then, in $\mathcal{A}$ we have $v - l + 1 = \lambda_t n + p - 1 - \lambda_t n + 1 = p$ (second part of condition 2) and $\lambda_t n = \lambda_t pr = pr'$ (first part of condition 2). In $\mathcal{A}_1$, we have $v - l + 1 = \lambda_t n + p - 1 - s - (\lambda_t n - s) + 1 = p$ (second part of condition 2) and, as we have already observed, $\lambda_t n = pr'$ (first part of condition 2). Finally, in $\mathcal{A}_2$ we have $v - l + 1 = m_{t'} + (\lambda_t - 1)n - (m_{t'} + (\lambda_t - 1)n - (p-1)) + 1 = p$ (second part of condition 2) and, by Lemma 8.2, there exists $r' \in \boldsymbol{N}$ such

that $m_{t'} + (\lambda_t - 1)n = pr'$ (first part of condition 2).

Claim: $L = L(\mathcal{A}) = L(\mathcal{A}_1) = L(\mathcal{A}_2)$.

Indeed, let j be such that either $\delta(q_0, a^j) = q_{j'} \in F$ or $\delta_1(q_0, a^j) = q_{j'} \in F_1$ or $\delta_2(q_0, a^j) = q_{j'} \in F_2$. Then, either $a^j \in L_1$, or $a^j = a^{m_i+\mu n}$ with $\mu \geq \lambda_i$, and so $a^j = a^{m_i+\mu n} \in (a^G)^+$, or $a^j = a^{n+\mu n} \in (a^G)^+$. Thus, $L(\mathcal{A}) \subseteq L$, $L(\mathcal{A}_1) \subseteq L$ and $L(\mathcal{A}_2) \subseteq L$.

Conversely, let us firstly prove that $L \subseteq L(\mathcal{A})$ and $L \subseteq L(\mathcal{A}_1)$.

Consider $a^j \in L$. Then, either $a^j \in L_1$, i.e., $j \in D$, and so $\delta(q_0, a^j) = q_j \in F$, $\delta_1(q_0, a^j) = q_j \in F_1$, or $a^j \in (a^G)^+$, i.e., $j = x_1 + x_2 + \ldots x_b$ with $x_i \in G$. In this last case, since G' is a group, we have $j(\mathrm{mod}\ n) \in G'$, i.e., $j = m_i + \mu n$. We also have $\lambda_i \leq \mu$ since $a^j \in (a^G)^+$ and $\lambda_i = min\{\mu \mid a^{m_i+\mu n} \in L \setminus L_1\}$ (see Proposition 8.1). Thus, $j = m_i + \mu n$, with μ such that either $\lambda_i \leq \mu < \lambda_t$, or $\lambda_i \leq \lambda_t \leq \mu$. In the first case we have $\delta(q_0, a^j) = q_j \in F$, $\delta_1(q_0, a^j) = q_j \in F_1$. In the second case, thanks to Lemma 8.2, $j = m_i + \mu n = pi + \lambda_t n + (\mu - \lambda_t)n$, with $i < r$ and so $\delta(q_0, a^j) = q_{\lambda_t n} \in F$, $\delta_1(q_0, a^j) = q_{\lambda_t n} \in F_1$. We have proven that $L = L(\mathcal{A}) = L(\mathcal{A}_1)$.

For getting the end of the proof, i.e., $L \subseteq L(\mathcal{A}_2)$, consider $a^j \in L$. Then, once again, either $a^j \in L_1$, i.e., $j \in D$, and so $\delta_2(q_0, a^j) = q_j \in F_2$, or $a^j \in (a^G)^+$, i.e., $j = x_1 + x_2 + \ldots x_b$ with $x_i \in G$. In this last case, since G' is a group, we have $j(\mathrm{mod}\ n) \in G'$, i.e., $j = m_i + \mu n$. We also have $\lambda_i \leq \mu$ since $a^j \in (a^G)^+$ and $\lambda_i = \min\{\mu \mid a^{m_i+\mu n} \in L \setminus L_1\}$ (see Proposition 8.1). Thus, $j = m_i + \mu n$. If either $\lambda_i \leq \mu < \lambda_t$ for $2 \leq i \leq t'$ or $\lambda_i \leq \mu < \lambda_t - 1$ for $t' < i \leq r$, then $\delta_2(q_0, a^j) = q_j \in F_2$. Otherwise, consider the positive integer t' related to the definition of $L(\mathcal{A}_2)$: we have either $i < t'$ or $i \geq t'$. If $i < t'$, we have $j = m_i + \mu n = ip + (\lambda_t - 1)n + pr + (\mu - \lambda_t)n = t'p + (\lambda_t - 1)n + (r - t' + i)p + (\mu - \lambda_t)n = m_{t'} + (\lambda_t - 1)n + (r - t' + i)p + (\mu - \lambda_t)n$ and so, being $r - t' + i \geq 0$, $\delta_2(q_0, a^j) = q_{m_{t'}+(\lambda_t-1)n} \in F_2$. If $i \geq t'$, we have $j = m_i + \mu n = ip + (\lambda_t - 1)n + (\mu - \lambda_t + 1)n = m_{t'} + (\lambda_t - 1)n + (i - t')p + (\mu - \lambda_t + 1)n$ and so, being $i - t' \geq 0$, $\delta_2(q_0, a^j) = q_{m_{t'}+(\lambda_t-1)n} \in F_2$.

A property $\mathcal{P}$ of $\mathcal{A}$, $\mathcal{A}_1$, $\mathcal{A}_2$.

Let us prove that the following relations hold in $\mathcal{A}$ and $\mathcal{A}_1$.

$$\forall g \geq p.\ L_{q_{\lambda_t n-p}} = L_{q_{\lambda_t n}}$$

Looking at the definition of $\mathcal{A}$ (resp. $\mathcal{A}_1$), we see that

$$\begin{aligned} g \geq p,\ a^g \in L_{q_{\lambda_t n}} &\Leftrightarrow \delta(q_{\lambda_t n}, a^g) = q_{\lambda_t n} \\ &\Leftrightarrow \exists \mu \geq 1\ .\ g = \mu p \\ &\Leftrightarrow \delta(q_{\lambda_t n - p}, a^g) = q_{\lambda_t n} \\ &\Leftrightarrow g \geq p,\ a^g \in L_{q_{\lambda_t n-p}} \end{aligned}$$

An analogous argument works for proving that the following relations hold in $\mathcal{A}_2$.

$$\forall g \geq p.\ L_{q_{m_{t'}+(\lambda_t-1)n-p}} = L_{q_{m_{t'}+(\lambda_t-1)n}}$$

Looking at the definition of $\mathcal{A}_2$, we see that

$$\begin{aligned} g \geq p,\ a^g \in L_{q_{m_{t'}+(\lambda_t-1)n}} &\Leftrightarrow \delta(q_{m_{t'}+(\lambda_t-1)n}, a^g) = q_{m_{t'}+(\lambda_t-1)n} \\ &\Leftrightarrow \exists \mu \geq 1\ .\ g = \mu p \\ &\Leftrightarrow \delta(q_{m_{t'}+(\lambda_t-1)n-p}, a^g) = q_{m_{t'}+(\lambda_t-1)n-p} \\ &\Leftrightarrow g \geq p,\ a^g \in L_{q_{m_{t'}+(\lambda_t-1)n-p}} \end{aligned}$$

Claim: Let $q_i, q_j \in Q \cup Q_1 \cup Q_2$ with $i < j$. If $j - i \neq 0 (\mathrm{mod}\ p)$, then $L_{q_i} \neq L_{q_j}$ in $\mathcal{A}$, $\mathcal{A}_1$, $\mathcal{A}_2$.

Consider first the case of automaton $\mathcal{A}$. There exists $h \in \boldsymbol{N}$ such that either $i + h = \lambda_t n$ or $i + h = \lambda_t n + p$ in $\mathcal{A}$. Thus, we have $\delta(q_0, a^{i+h}) = \delta(q_i, a^h) \in F$. By contradiction, let us suppose that $L_{q_i} = L_{q_j}$. Then, we also have $\delta(q_0, a^{j+h}) = \delta(q_j, a^h) \in F$. On the other hand, $j + h \geq i + h \geq l$. So, we have $\delta(q_0, a^{j+h}) = q_{\lambda_t n}$. As a consequence, we have $j + h \equiv i + h (\mathrm{mod}\ p)$, which is a contradiction since $j - i \neq 0 (\mathrm{mod}\ p)$.

Now, consider the case of automaton $\mathcal{A}_1$. There exists $h \in \boldsymbol{N}$ such that either $i + h = \lambda_t n$ or $i + h = \lambda_t n + p$ in $\mathcal{A}_1$. Thus, we have $\delta_1(q_0, a^{i+h}) =$

$\delta_1(q_i, a^h) \in F_1$. By contradiction, let us suppose that $L_{q_i} = L_{q_j}$. Then, we also have $\delta_1(q_0, a^{j+h}) = \delta_1(q_j, a^h) \in F_1$. On the other hand, $j+h \geq i+h \geq l$. So, we have $\delta_1(q_0, a^{j+h}) = q_{\lambda_t n}$. As a consequence, we have $j + h \equiv i + h \pmod{p}$, which is a contradiction since $j - i \neq 0 \pmod{p}$.

Finally, consider the case of automaton $\mathcal{A}_2$. There exists $h \in \boldsymbol{N}$ such that $i + h = m_{t'} + (\lambda_t - 1)n$ or $i + h = m_{t'} + (\lambda_t - 1)n + p$ in $\mathcal{A}_2$.

Thus, we have $\delta_2(q_0, a^{i+h}) = \delta_2(q_i, a^h) \in F_2$. By contradiction, let us suppose that $L_{q_i} = L_{q_j}$. Then, we also have $\delta_2(q_0, a^{j+h}) = \delta_2(q_j, a^h) \in F_2$. On the other hand, $j+h \geq i+h \geq l$. So, we have $\delta_2(q_0, a^{j+h}) = q_{m_{t'}+(\lambda_t-1)n}$. As a consequence, we have $j + h \equiv i + h \pmod{p}$, which is a contradiction since $j - i \neq 0 \pmod{p}$.

Claim: Either $L_{q_i} \neq L_{q_j}$ in $\mathcal{A}$ for all $q_i, q_j \in Q$ with $i < j$ and $j - i = 0 \pmod{p}$ (and $\mathcal{A}$ is minimal) or we can construct $\mathcal{A}_1$ or $\mathcal{A}_2$. If we can construct $\mathcal{A}_1$, then either $L_{q_i} \neq L_{q_j}$ in $\mathcal{A}_1$ for all $q_i, q_j \in Q_1$ with $i < j$ and $j - i = 0 \pmod{p}$ (and $\mathcal{A}_1$ is minimal) or we can construct $\mathcal{A}_2$. If we can construct $\mathcal{A}_2$, $L_{q_i} \neq L_{q_j}$ in $\mathcal{A}_2$ for all $q_i, q_j \in Q$ with $i < j$ and $j - i = 0 \pmod{p}$ and $\mathcal{A}_2$ is minimal.

Let us consider the automaton $\mathcal{A}$. In virtue of the last claim, if $L_{q_i} \neq L_{q_j}$ in $\mathcal{A}$ for all $q_i, q_j \in Q$ with $i < j$ and $j - i = 0 \pmod{p}$, then $\mathcal{A}$ is minimal. So, let i, j be such that $q_i, q_j \in Q$, $L_{q_i} = L_{q_j}$ and $j = i + \mu p$, with $\mu \geq 1$. We have either $i \leq \lambda_t n - p$ (case (a)) or $i > \lambda_t n - p$ (case (b)). In case (a) for all g with $g < p$, we have $i \leq \lambda_t n - p + g$ and $\delta(q_i, a^{\lambda_t n - p + g - i}) = \delta(q_0, a^{\lambda_t n - p + g}) = \delta(q_{\lambda_t n - p}, a^g) \in F$ if and only if $\delta(q_j, a^{\lambda_t n - p + g - i}) = \delta(q_{i+\mu p}, a^{\lambda_t n - p + g - i}) = \delta(q_0, a^{\lambda_t n - p + g + \mu p}) = \delta(q_{\lambda_t n}, a^{g+(\mu-1)p}) = \delta(q_{\lambda_t n}, a^g) \in F$. Thus, in case (a), by using the property $\mathcal{P}$ of $\mathcal{A}$, we conclude that $L_{q_{\lambda_t n - p}} = L_{q_{\lambda_t n}}$. Thus, $1 \in L_{q_{\lambda_t n}}$ implies $1 \in L_{q_{\lambda_t n - p}}$, i.e., $a^{\lambda_t n - p} = a^{m_r + (\lambda_t - 1)n} \in L$ which allows us to construct $\mathcal{A}_2$. In case (b), i.e. $i > \lambda_t n - p$, by contradiction suppose that we cannot construct $\mathcal{A}_1$ nor $\mathcal{A}_2$. In particular, since we cannot construct $\mathcal{A}_1$, either $p = 1$ or $q_{\lambda_t n - 1} \in F$. We cannot have $p = 1$ since $j > i > \lambda_t n - p$ and $Q = \{q_0, q_1, \ldots, q_v\} = \{q_0, q_1, \ldots, q_{\lambda_t n}\}$ imply $j = i = \lambda_t n$, a contradiction. So, $p > 1$ and $q_{\lambda_t n - 1} \in F$. We also have $i \leq \lambda_t n$ since $j = i + \mu p$, with $\mu \geq 1$. So, let g be such that $\delta(q_i, a^g) = q_{\lambda_t n - 1} \in F$. Since $L_{q_i} = L_{q_j}$, we

have $\delta(q_j, a^g) \in F$; thus, $j = i + \mu p > \lambda_t n - p + \mu p$, with $\mu \geq 1$, implies $\delta(q_j, a^g) = q_{\lambda_t n}$. On the other hand, we also have $\delta(q_i, a^{g+1}) = q_{\lambda_t n}$ and, since $L_{q_i} = L_{q_j}$, we have $\delta(q_j, a^{g+1}) \in F$, a contradiction.

We now suppose that we can construct the automaton $\mathcal{A}_1$ (and so $p > 1$). In virtue of the last claim, if $L_{q_i} \neq L_{q_j}$ in $\mathcal{A}_1$ for all $q_i, q_j \in Q_1$ with $i < j$ and $j - i = 0 (\mathrm{mod}\ p)$, then $\mathcal{A}_1$ is minimal. So, let i, j be such that $q_i, q_j \in Q_1$, $L_{q_i} = L_{q_j}$ and $j = i + \mu p$, with $\mu \geq 1$. We have either $i \leq \lambda_t n - p$ (case (a)) or $i > \lambda_t n - p$ (case (b)). In case (a) for all g with $g < p$, we have $i \leq \lambda_t n - p + g$ and $\delta_1(q_i, a^{\lambda_t n - p + g - i}) = \delta_1(q_0, a^{\lambda_t n - p + g}) = \delta_1(q_{\lambda_t n - p}, a^g) \in F_1$ if and only if $\delta_1(q_j, a^{\lambda_t n - p + g - i}) = \delta_1(q_{i+\mu p}, a^{\lambda_t n - p + g - i}) = \delta_1(q_0, a^{\lambda_t n - p + g + \mu p}) = \delta_1(q_{\lambda_t n}, a^{g + (\mu - 1)p}) = \delta_1(q_{\lambda_t n}, a^g) \in F_1$. Thus, in case (a), by using the property $\mathcal{P}$ of $\mathcal{A}_1$, we conclude that $L_{q_{\lambda_t n - p}} = L_{q_{\lambda_t n}}$. Thus, $1 \in L_{q_{\lambda_t n}}$ implies $1 \in L_{q_{\lambda_t n - p}}$, i.e., $a^{\lambda_t n - p} = a^{m_r + (\lambda_t - 1)n} \in L$ which allows us to construct $\mathcal{A}_2$. In case (b), i.e. $i > \lambda_t n - p$, by contradiction suppose that we cannot construct $\mathcal{A}_2$. We have $i \leq \lambda_t n - (s + 1)$ since $j = i + \mu p$, with $\mu \geq 1$. In addition, we know that $1 \leq s \leq p - 1$ and, since $\lambda_t n - p < i \leq \lambda_t n - (s + 1)$ we have $s < p - 1$. Looking at the definition of $\mathcal{A}_1$, we know that $q_{\lambda_t n - (s+1)} \in F_1$. So, let g be such that $\delta_1(q_i, a^g) = q_{\lambda_t n - (s+1)} \in F_1$. Since $L_{q_i} = L_{q_j}$, we have $\delta_1(q_j, a^g) \in F_1$; thus, $j = i + \mu p > \lambda_t n - p + \mu p$, with $\mu \geq 1$, implies $\delta_1(q_j, a^g) = q_{\lambda_t n}$. On the other hand, we also have $\delta_1(q_i, a^{g+s+1}) = q_{\lambda_t n}$ and, since $L_{q_i} = L_{q_j}$, we have $\delta_1(q_j, a^{g+s+1}) \in F_1$, a contradiction since $s + 1 < p$.

Finally, suppose that we construct $\mathcal{A}_2$ and let us prove that $\mathcal{A}_2$ is minimal. In virtue of the last claim, if $L_{q_i} \neq L_{q_j}$ in $\mathcal{A}_2$ for all $q_i, q_j \in Q_2$ with $i < j$ and $j - i = 0 (\mathrm{mod}\ p)$, then $\mathcal{A}_2$ is minimal. So, by contradiction, let i, j be such that $q_i, q_j \in Q_2$, $L_{q_i} = L_{q_j}$ and $j = i + \mu p$, with $\mu \geq 1$. We have either $i \leq m_{t'} + (\lambda_t - 1)n - p$ (case (a)) or $i > m_{t'} + (\lambda_t - 1)n - p$ (case (b)). In case (a) for all g with $g < p$, we have $i \leq m_{t'} + (\lambda_t - 1)n - p + g$ and $\delta_2(q_i, a^{m_{t'} + (\lambda_t - 1)n - p + g - i}) = \delta_2(q_0, a^{m_{t'} + (\lambda_t - 1)n - p + g}) = \delta_2(q_{m_{t'} + (\lambda_t - 1)n - p}, a^g) \in F_2$ if and only if $\delta_2(q_j, a^{m_{t'} + (\lambda_t - 1)n - p + g - i}) = \delta_2(q_{i+\mu p}, a^{m_{t'} + (\lambda_t - 1)n - p + g - i}) = \delta_2(q_0, a^{m_{t'} + (\lambda_t - 1)n - p + g + \mu p}) = \delta_2(q_{m_{t'} + (\lambda_t - 1)n}, a^{g + (\mu - 1)p}) = \delta_2(q_{m_{t'} + (\lambda_t - 1)n}, a^g) \in F_2$. Thus, in case (a), by using the property $\mathcal{P}$ of $\mathcal{A}_2$, we conclude that $L_{q_{m_{t'} + (\lambda_t - 1)n - p}} = L_{q_{m_{t'} + (\lambda_t - 1)n}}$. Thus, $1 \in L_{q_{m_{t'} + (\lambda_t - 1)n}}$

implies $1 \in L_{q_{m_{t'}+(\lambda_t-1)n-p}}$, i.e., $a^{m_{t'}+(\lambda_t-1)n-p} = a^{m_{t'-1}+(\lambda_t-1)n} \in L$, which contradicts the hypothesis that t' is minimal. In case (b), $i > m_{t'} + (\lambda_t - 1)n - p$. We also have $i < m_{t'} + (\lambda_t - 1)n - (p-1)$ since $j = i + \mu p$, with $\mu \geq 1$. Thus, $m_{t'} + (\lambda_t - 1)n - p < i < m_{t'} + (\lambda_t - 1)n - (p-1)$ which is a contradiction no such nonnegative integer i can exist. ■

Automata in Figures 8.3 and 8.4 are examples of minimal automata obtained thanks to the constructions of the second and third automaton respectively in the proof of Proposition 8.5.

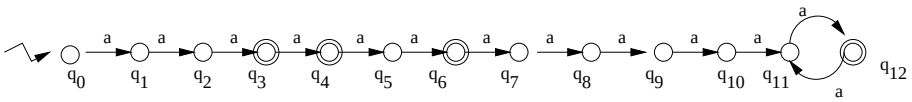

Figure 8.3: Automaton for $L = \{a^3, a^4\} \cup \{a^6, a^{14}, a^{16}\}^+$

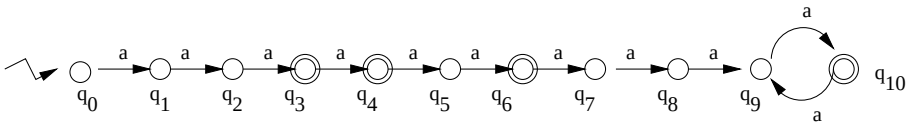

Figure 8.4: Automaton for $L = \{a^3, a^4\} \cup \{a^6, a^{14}, a^{10}\}^+$

It is now clearly evident that we can decide whether a regular language $L \subseteq a^*$ is a C_{PA} generated language, i.e., whether the minimal finite state automaton $\mathcal{A}$ recognizing L satisfies the two conditions reported in Proposition 8.5. So, we have the following result.

Proposition 8.6 *Given a regular language L, with $L \subseteq a^*$, we can decide whether L is a C_{PA} generated language.*

8.3 Uniform languages

In the previous section we have seen that we can decide whether or not a regular language $L \subseteq a^*$ is C_{PA} generated, by means of properties of the

corresponding automaton recognizing L. We see how the characterization given in Proposition 8.1 can be easily generalized to languages L with the form $L = \bigcup_{j \in J} A^j$, A being an alphabet with $|A| \geq 1$ (Definition 8.1).

Definition 8.1 *A language L is an* uniform language *if there exists a subset J of $\boldsymbol{N}$ such that $L = A^J = \bigcup_{j \in J} A^j = \bigcup_{j \in J} \{w \in A^* \mid |w| = j\}$.*

We recall that, given $j \in \boldsymbol{N}$, the set $A^j = \{w \in A^* || w| = j\}$ is a code, called *uniform code*. Observe that $Lin(^\sim(A^j)) = A^j$, i.e., we can identify A^j with its circularization. Since $Lin(^\sim A^J) = A^J$, we can identify each language A^J with its circularization.

Using Lemmata 8.4 and 8.5 as preliminary results, we give a characterization of all subsets J of $\boldsymbol{N}$ such that $A^J \in Reg^\sim \cap C_{PA}(Fin, Fin)$. As in the case of one-letter alphabet, in the next part of this section we will often suppose $R \neq \emptyset$.

Lemma 8.4 *Let $SC_{PA} = (A, I, R)$ be a circular splicing system such that there exists $J \subseteq \boldsymbol{N}$ with $C(SC_{PA}) = A^J$. Denote $R = \{r_1, \ldots, r_t\}$ and $r_i = w_{1,i}|w_{2,i}\$w_{3,i}|w_{4,i}$, where $w_{j,i} \in A^*$, for $j \in \{1,2,3,4\}$, $i \in \{1, \ldots, t\}$. For all $i \in \{1, \ldots, t\}$, for all $x, y \in A^J$ with $|x| \geq |w_{1,i}| + |w_{2,i}|$, $|y| \geq |w_{3,i}| + |w_{4,i}|$, there exist x', y', $|x'| = |x|$ and $|y'| = |y|$ such that $^\sim x'y'$ is generated by means of the rule r_i. Consequently, $^\sim xy \in A^J = C(SC_{PA})$.*

Proof. Let us consider $SC_{PA} = (A, I, R)$ with $C(SC_{PA}) = A^J$, $R = \{r_1, \ldots, r_t\}$ and $r_i = w_{1,i}|w_{2,i}\$w_{3,i}|w_{4,i}$. Let $x, y \in A^J$ such that $|x| \geq |w_{1,i}| + |w_{2,i}|$ and $|y| \geq |w_{3,i}| + |w_{4,i}|$. Consider the prefix h (resp. k) of x (resp. y) such that $|h| = |x| - |w_{1,i}w_{2,i}|$ (resp. $|k| = |y| - |w_{3,i}w_{4,i}|$). Set $x'' = hw_{1,i}w_{2,i}$, $y'' = kw_{3,i}w_{4,i}$. Rule r_i applied to x'', y'' generate $^\sim x'y'$, where $x' \sim x''$ and $y'' \sim y'$. Since $|x'| = |x''| = |x|$, $|y'| = |y''| = |y|$ and $x, y \in A^J$, we have that $x', x'', y', y'' \in A^J$. Moreover $C(SC_{PA}) = A^J$ implies $^\sim x'y' \in C(SC_{PA}) = A^J$. Finally, $|xy| = |x'y'|$ and hence $xy \in A^J = C(SC_{PA})$. ■

All the proofs of the results below are obtained by simple extension of the analogous result for the case of one-letter alphabet. They are reported for the sake of completeness.

Lemma 8.5 *Let $SC_{PA} = (A, I, R)$ be a circular splicing system such that there exists $J \subseteq \boldsymbol{N}$ with $C(SC_{PA}) = A^J$. Denote $R = \{r_1, \ldots, r_t\}$ and $r_i = w_{1,i}|w_{2,i}\$w_{3,i}|w_{4,i}$, where $w_{j,i} \in A^*$, for $j \in \{1,2,3,4\}$, $i \in \{1, \ldots, t\}$. Let $m = max\ \{|w_{1,i}| + |w_{2,i}|,\ |w_{3,i}| + |w_{4,i}| \ \mid i \in \{1, \ldots, t\}\}$ and $L_2 = \{w \in A^J \mid |w| \geq m\}$. Then, $A^J = L_1 \cup L_2$ where $L_1 = A^J \setminus L_2$ is a finite subset of A^J. Furthermore, either $L_2 = \emptyset$ or there exists $n \in \boldsymbol{N}$ such that $L_2 = (A^G)^+$, where G is a set of representatives of the elements in a (periodic) subgroup of $\boldsymbol{Z}_n$ and $A^G \cap Fact(L_1) = \emptyset$.*

Proof. Let us denote $m = max\ \{|w_{1,i}|+|w_{2,i}|,\ |w_{3,i}|+|w_{4,i}| \mid i \in \{1, \ldots, t\}\}$, $A^J = C(SC_{PA})$ and $L_2 = \{w \in A^J \mid |w| \geq m\}$.

Then, we obviously have $A^J = L_1 \cup L_2$ where $L_1 = A^J \setminus L_2$ is a finite subset of A^J. If $L_2 = \emptyset$, the proof is concluded. Otherwise, let $n \in \boldsymbol{N}$, with $n = min\{j \mid j \in J, j > m\}$. For all $l \in \{0, \ldots, n-1\}$ such that $(A^n)^* A^l \cap A^J \neq \emptyset$, let $h_l = min\{h \mid A^{nh+l} \subseteq L_2\}$. We have already observed that $h_0 = 1$ and $h_l > 0$, for all l's. Let $G \subseteq \boldsymbol{N}$ with $G = \{h_l n + l \mid l \in \{0, \ldots, n-1\},\ (A^n)^* A^l \cap L_2 \neq \emptyset\}$.

Let us prove that $L_2 = (A^G)^+$. We show that $(A^G)^+ \subseteq L_2$, by using induction on the length of $w \in A^g \subseteq (A^G)^+$. Indeed, either $w \in A^G \subseteq L_2$ or $w = w_1 w_2$, with $w_1 \in A^{g_1}, w_2 \in A^{g_2}$, $A^{g_1} \subseteq (A^G)^+$, $A^{g_2} \subseteq (A^G)^+$ and $g_1 < g, g_2 < g$. In the last case, by using induction hypothesis, $A^{g_1}, A^{g_2} \subseteq L_2$ and, by using Lemma 8.4, $A^g = A^{g_1} A^{g_2} \subseteq L_2$. So, $(A^G)^+ \subseteq L_2$. In order to prove that $L_2 \subseteq (A^G)^+$, let $w \in A^l \subseteq L_2$ and let h, t be such that $l = nh + t$, with $h \in \boldsymbol{N}$ and $t \in \{0, \ldots, n-1\}$. We already know that $h \geq h_t$. Since $A^{h_t n + t} \subseteq (A^G)^+$ and $A^n \subseteq (A^G)^+$, we have $A^{h_t n+t}(A^n)^* \subseteq (A^G)^+$ and so $w \in A^l = A^{nh+t} \subseteq A^{h_t n+t}\ (A^n)^* \subseteq (A^G)^+$. Clearly we have $A^G \cap Fact(L_1) = \emptyset$.

Finally, let us prove that G is a set of representatives of the elements of a subset G' of $\boldsymbol{Z}_n$ where G' is a (periodic) subgroup of the finite cyclic group of order n. Thus, G' is defined as the set of the (different) classes modulo n of the elements in G. We already know that $n \in G$. Furthermore, for each $g_1, g_2 \in G$, we have $A^{g_1} A^{g_2} \subseteq (A^G)^+ = L_2$. Thus, for $h \in \boldsymbol{N}$ and $k \in \{0, \ldots, n-1\}$ such that $g_1 + g_2 = hn + k$, we have $h \geq h_k$ with $h_k n + k \in G$. Thus G' is a submonoid of a finite group (namely $\boldsymbol{Z}_n$):

a classical result from group theory states that G' is a subgroup (of $\boldsymbol{Z}_n$, indeed for each $g \in G'$, G' being a submonoid, we have $(n-1)g \in G'$ and $(n-1)g$ is the inverse of g). ■

A characterization of the uniform languages with a C_{PA} generated circularization is given in Proposition 8.7. This characterization is obtained by using Proposition 7.3 and Lemma 8.5.

Proposition 8.7 *Given $J \subseteq \boldsymbol{N}$, a language $L = A^J$ is C_{PA} generated if and only if there exist 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 $A^G \cap Fact(L_1) = \emptyset$.*

Corollary 8.2 provides a simpler characterization of the uniform languages with a C_{PA} generated circularization.

Corollary 8.2 *Given $J \subseteq \boldsymbol{N}$, a language $L = A^J$ is C_{PA} generated if and only if there exist finite subsets L_1, A^G of A^J and a positive integer n such that $L = L_1 \cup (A^G)^+$ and, when $A^G \neq \emptyset$, $max\{|l| \mid l \in L_1\} < n = min\{|l| \mid l \in A^G\}$.*

Proof. Thanks to Propositions 7.3 and 8.1, the conclusion easily follows. ■

We end this section with some observations concerning the descriptional complexity of a circular splicing system which generates a circular language $L = A^J$ (Proposition 8.8). Observe that we are not able to exhibit a minimal circular splicing system, as we have done for languages on one-letter alphabet (Proposition 8.2).

Proposition 8.8 *Let $L = A^J$ be a C_{PA} generated language, $J \subseteq \boldsymbol{N}$. Then, there exists $n \in \boldsymbol{N}$ and a splicing system (A, I, R) generating L with either $R = \emptyset$ or R containing at most $\frac{|A^n| \cdot (|A^n|+1)}{2}$ rules.*

Proof. Let $L = A^J$ be a C_{PA} generated language. Then, using Corollary 8.1 and with the same notations, $L = L_1 \cup (A^G)^+$ and a positive integer n exists such that, when $A^G \neq \emptyset$, $n = min\{|l| \mid l \in A^G\}$, $L_1 = \{l \in L \mid |l| < n\}$. If

$A^G = \emptyset$ then the circular splicing system (A, I, R) where $R = \emptyset$ and $I = L_1$ obviously generates $L = L_1$.
Thus, suppose that $A^G \neq \emptyset$ and set $A^n = \{\alpha_i \mid i \in \{1, \ldots, |A|^n\}\}$. Consider the circular splicing system (A, I, R) where $R = \{\alpha_i \mid 1 \$ 1 \mid \alpha_j;\ j \geq i,\ i \in \{1, \ldots, |A|^n\}\}$ and $I = L_1 \cup A^G$. Obviously, I is a finite set and $I \subseteq A^J$. Let us prove that $A^J = C(SC_{PA})$. We show that $A^J \subseteq C(SC_{PA})$. Obviously, $I \subseteq C(SC_{PA})$ and, for each $w \in A^g \in A^J \setminus I$, we prove that $w \in C(SC_{PA})$ by using induction on $|w|$. Since $w \in A^J \setminus I$, then $w \in (A^G)^+ \setminus A^G$ which in turn implies $w = w_1 w_2$ with $w_1 \in (A^G)^+$, $w_2 \in (A^G)^+$. Furthermore, $w_1 \in (A^G)^+$ implies $|w_1| \geq n$ and, analogously, $w_2 \in (A^G)^+$ implies $|w_2| \geq n$, i.e., $n \leq |w_1| < |w|$, $n \leq |w_2| < |w|$.

By using induction hypothesis, $w_1, w_2 \in C(SC_{PA})$. Furthermore, there exist $i, j \in \{1, \ldots, |A|^n\}$ such that ${}^\sim w_1 = {}^\sim \alpha_i x_1$, ${}^\sim w_2 = {}^\sim \alpha_j x_2$. Thus, either the rule $\alpha_i \mid 1 \$ 1 \mid \alpha_j$ or the rule $\alpha_j \mid 1 \$ 1 \mid \alpha_i$ is in R and may be applied to w_1, w_2. Thus, by using the second part of the statement contained in Lemma 8.4, we have $w = w_1 w_2 \in C(SC_{PA})$.

We now prove that $C(SC_{PA}) \subseteq A^J$ with a slight variation of the argument contained in Proposition 7.1 and by using induction on the length of $w \in C(SC_{PA})$. Let $w \in A^l \subseteq C(SC_{PA})$ and let h, j be such that $l = nh + j$, with $h \in \boldsymbol{N}$ and $j \in \{0, \ldots, n-1\}$. Since $I \subseteq A^J$, we can suppose that w has been generated by using a rule in R and starting from two words x, y, such that $x \in A^{nh_1 + j_1}$, $y \in A^{nh_2 + j_2}$, $x, y \in C(SC_{PA})$ of shorter length than w (since they are different from w). Then, $h_1 \geq 1$, $h_2 \geq 1$ (otherwise rules in R cannot be applied) and $w = xy$. By using induction hypothesis, $x, y \in A^J$. Then $x, y \in (A^G)^+$ (since $h_1, h_2 \geq 1$) and thus $w = xy \in (A^G)^+$, since $(A^G)^+$ is a submonoid. Finally, $|R| = \sum_{i=1}^{|A^n|} i = \frac{|A^n| \cdot (|A^n|+1)}{2}$. ■

Example 8.3 Let $L = \{A^3, A^4\} \cup \{A^6, A^{14}, A^{16}\}^+$. Then L is an uniform language which is C_{PA} generated since L satisfies the hypothesis contained in Proposition 8.7 with $n = 6$. Denote $A^6 = \{\alpha_i \mid i \in \{1, \ldots, |A|^6\}\}$. According to Proposition 8.8, $L = C(SC_{PA})$, where $SC_{PA} = (A, I, R)$, $I = \{A^3, A^4, A^6, A^{14}, A^{16}\}$ and $R = \{\alpha_i \mid 1 \$ 1 \mid \alpha_j;\ j \geq i, i \in \{1, \ldots, |A|^6\}\}$.

Chapter 9

Circular regular Paun generated languages: cyclic and weak cyclic languages

We have already seen that regular circular C_{PA} generated languages exist which are not the Kleene closure (closed under conjugacy relation) of a regular language, namely $^{\sim}(ab)^*$ (see Example 7.4). The aim of this chapter [3] is to give other examples of such languages. Precisely, we construct a class of languages called *cyclic languages* (Definition 9.1) and *weak cyclic languages* (Section 9.2).

After some preliminary definitions, in Propositions 9.2, 9.3 we prove that these classes of languages are generated by finite (Paun) circular splicing, even if they are not star languages. The same family of cyclic languages allows us to give a negative answer to the natural question whether a regular circular C_{PA} generated language is always fingerprint closed (Proposition 9.4).

[3] Results contained in [12]

9.1 Cyclic languages: definitions and results

In this chapter we will often consider prefixes (resp. proper prefixes) x of w which are not the empty word. Thus, we set $Pref(w) = \{x \in A^+ \mid x \text{ is a proper prefix of } w\}$. Proper prefixes of the words w' such that $w' \sim w$ are also needed, and so we set $Pref_c(w) = \{x \in A^+ \mid \exists w' \sim w.$ x is a proper prefix of $w'\}$. Finally, we recall that a word $w \in A^*$ is *unbordered* if $w \neq xyx$, $x \in A^+$.

Definition 9.1 (Cyclic languages) *For each $w \in A^+$, we define $cyclic(w) = {}^{\sim}(w^*Pref_c(w)) = \cup_{p \in Pref_c(w)} {}^{\sim}(w^*p)$.*

Example 9.1 Let $w = abc$. Then, $Pref_c(abc) = \{a, ab, b, bc, c, ca\}$ and

$$cyclic(w) = {}^{\sim}(abc)^*a \cup {}^{\sim}(abc)^*ab \cup {}^{\sim}(abc)^*b \cup$$
$$\cup {}^{\sim}(abc)^*bc \cup {}^{\sim}(abc)^*c \cup {}^{\sim}(abc)^*ca$$

Remark 9.1 Notice that words w, w' exist such that $w \sim w'$ and $cyclic(w) \neq cyclic(w')$. For example, $cyclic(abc) \neq cyclic(cab)$ since $abca \in cyclic(abc) \setminus cyclic(cab)$ (if $abca \in cyclic(cab)$ then we would also have $abca \sim caba$ which is impossible). Analogously, $cyclic(aba) \neq cyclic(baa)$ since $abab \in cyclic(aba) \setminus cyclic(baa)$.

The technical lemma below will be useful for proving our main result.

Lemma 9.1 *For every $x, w \in A^+$, $x \in Pref_c(w)$ if and only if either x is a proper factor of w or $x = sp$, with s being a suffix of w, p being a prefix of w and $|sp| < |w|$. Furthermore, for every $x \in Pref_c(w)$, $x_1, x_2 \in A^+$, if $x = x_1x_2$ then $x_1, x_2 \in Pref_c(w)$.*

Proof.

Let $x, w \in A^+$. Suppose that either x is a proper factor of w or $x = sp$, with s being a suffix of w, p being a prefix of w and $|sp| < |w|$. Then, obviously we have $x \in Pref_c(w)$.

Conversely, suppose that $x \in Pref_c(w)$. Thus, there exist $u \in A^+$, $y_1, y_2 \in A^*$ such that $w = y_1y_2$, $xu = w' = y_2y_1 \sim w$. Two cases can occur: either $|y_2| \geq |x|$ or $|y_2| < |x|$.

In the first case, there exists $t \in A^*$ such that $y_2 = xt$. Consequently, we have $w = y_1xt$ with $y_1t \neq 1$ (otherwise $x = w = w'$) and x is a proper factor of w. In the second case, there exists $t \in A^+$ such that $x = y_2t$. Consequently, we have $y_1 = tu$ and $w = tuy_2$. So $x = sp$, with $s = y_2$ being a suffix of w, $p = t$ being a prefix of w and $|sp| < |w|$ (since $u \neq 1$).

In order to prove the second part of the statement, suppose that $x = x_1x_2$ with $x \in Pref_c(w)$, $x_1, x_2 \in A^+$. By using the first part of the statement, we have that either x is a proper factor of w or $x = sp$, with s being a suffix of w, p being a prefix of w and $|sp| < |w|$. In the first case x_1, x_2 are also proper factors of w and so $x_1, x_2 \in Pref_c(w)$. In the second case, we have $x_1x_2 = sp$. Consequently, either x_1 is a prefix of s or x_2 is a suffix of p. If x_1 is a prefix of $s = x_1s'$ then x_1 is a proper factor of w and we have $x_2 = s'p$, with s' being a suffix of w, p being a prefix of w and $|s'p| < |sp| < |w|$. So, we have $x_1, x_2 \in Pref_c(w)$. Otherwise, x_2 is a suffix of $p = p'x_2$ and $x_1 = sp'$, with s being a suffix of w, p' being a prefix of w and $|sp'| < |sp| < |w|$ and once again, x_2 being a proper factor of w, we have $x_1, x_2 \in Pref_c(w)$. ■

We will now prove the main result of this section, namely that $cyclic(w)$ is C_{PA} generated provided that w is an unbordered word with $|w| > 2$. The proof of this result is broken up into several intermediate results. Lemmata 9.2 and 9.3 allow us to find a Paun circular splicing system such that all the circular words in $cyclic(w)$ are generated by using its rules, as proved in Lemma 9.4. Next, in Lemma 9.8 (by using Lemmata 9.5, 9.6 and 9.7 as preliminary results), we prove that if a circular word $^\sim x$ is generated by the above-mentioned Paun circular splicing system then $^\sim x \in cyclic(w)$. In this section, let us suppose that we use Paun's definition for circular splicing.

Let us briefly sketch what we will prove in Lemmata 9.2 and 9.3. We must generate all the words with the form w^tp_g where t is a positive integer and $p_g \in Pref_c(w)$. In the next part of this section we will suppose that $|p_g| = g$. In Lemma 9.2 we consider the case $g = 1$, whereas the case $g > 1$ is examined in Lemma 9.3.

Intuitively, a circular word $^\sim x$ is generated by a rule r in a circular splicing system if one of the representatives of $^\sim x$ is the concatenation of two other words each being in turn a representative of a generated word and

having a suffix and a prefix related to the rule r. If $w = ps$ is a factorization of w, we will show in Lemma 9.2 that we can define *circular* rules such that

1. $w^t p_1 \sim (w^g p_1 p)(s w^{g'})$ and $^{\sim}w^t p_1$ is generated starting from $^{\sim}w^g p_1 p$, $^{\sim}s w^{g'} \in cyclic(w)$, if p_1 is a suffix of w;

2. $w^t p_1 \sim (w^g p)(s w^{g'} p_1)$ and $^{\sim}w^t p_1$ is generated starting from $^{\sim}w^g p$, $^{\sim}p_1 s w^{g'} \in cyclic(w)$, if p_1 is a factor of w, with the additional hypothesis that $p_1 s$ is once again a suffix of w;

3. $w^t p_1 \sim (w^g p)(s p_1 w^{g'})$ and $^{\sim}w^t p_1$ is generated starting from $^{\sim}w^g p$, $^{\sim}s p_1 w^{g'} \in cyclic(w)$, if p_1 is a prefix of w.

Lemma 9.2 *Let $w \in A^+$ be a word with $|w| > 2$. Let $p_1 \in Pref_c(w)$ (i.e., let $p_g \in Pref_c(w)$ with $g = 1$) and let $w = ps$ be a factorization of w with $p \neq w$, $s \neq w$.*

1. *If p_1 is a suffix of w and $p_1 \neq s$, the circular word $^{\sim}\, w^t p_1 \in cyclic(w)$ is generated by means of the rule*

$$R_{p_1} = w p_1 p \mid w \$ w \mid s w$$

and starting from $^{\sim}w^{t_1 - 1} p_1 p w \in cyclic(w)$ and $^{\sim}w^{t_2 - 1} s w \in cyclic(w)$, with $t_1 + t_2 = t - 1$, $t_1 \geq 2$, $t_2 \geq 2$.

2. *If p_1 is a factor of w and p is such that p_1 is a proper suffix of p (so $p_1 s$ is once again a proper suffix of w), the circular word $^{\sim}w^t p_1 \in cyclic(w)$ is generated by means of the rule*

$$R_{p_1} = w p \mid w \$ w p_1 \mid s w$$

and starting from $^{\sim}w^{t_1 - 1} p w \in cyclic(w)$ and $^{\sim}w^{t_2 - 1} p_1 s w \in cyclic(w)$, with $t_1 + t_2 = t - 1$, $t_1 \geq 2$, $t_2 \geq 2$.

3. *If p_1 is a prefix of w and $p_1 \neq p$, the circular word $^{\sim}w^t p_1 \in cyclic(w)$ is generated by means of the rule*

$$R_{p_1} = w p \mid w \$ w \mid s p_1 w$$

and starting from $^{\sim}w^{t_1 - 1} p w \in cyclic(w)$ and $^{\sim}w^{t_2 - 1} s p_1 w \in cyclic(w)$, with $t_1 + t_2 = t - 1$, $t_1 \geq 2$, $t_2 \geq 2$.

Proof. Let t_1, t_2 be positive integers with $t_1 \geq 2$, $t_2 \geq 2$ and let $w = ps$ be a factorization of w with $p \neq w$, $s \neq w$. Obviously, if ${}^{\sim}w^t p_1 \in cyclic(w)$ then p_1 is a suffix or a prefix or a factor of w (Lemma 9.1). It is also obvious that if p_1 is a suffix of w and $p_1 \neq s$, then ${}^{\sim}w^{t_1-1}p_1 pw \in cyclic(w)$ and ${}^{\sim}w^{t_2-1}sw \in cyclic(w)$, once again by using Lemma 9.1. By using the same lemma, we can also conclude that if p_1 is a factor of w and $p_1 s$ is a proper suffix of w, then ${}^{\sim}w^{t_1-1}pw \in cyclic(w)$ and ${}^{\sim}w^{t_2-1}p_1 sw \in cyclic(w)$. Finally, once again in virtue of Lemma 9.1, if p_1 is a prefix of w and $p_1 \neq p$, then ${}^{\sim}w^{t_1-1}pw \in cyclic(w)$ and ${}^{\sim}w^{t_2-1}sp_1 w \in cyclic(w)$. Then, by using Paun's definition of circular splicing, it is easy to see that the conclusion holds. ■

Lemma 9.3 below deals with rules for generating circular words ${}^{\sim}w^t p_g$ where t is a positive integer and $p_g \in Pref_c(w)$, $g > 1$. In the proof of this lemma we will also use a part of the statement contained in Lemma 9.1. If $p_g = ps$ is a factorization of p_g, we will show in Lemma 9.3 that we can define *circular* rules such that $w^t p_g \sim (w^g p)(sw^{g'})$ and ${}^{\sim}w^t p_g$ is generated starting from ${}^{\sim}w^g p$ and ${}^{\sim}sw^{g'}$.

Lemma 9.3 *Let $w \in A^*$ be a word with $|w| > 2$. Let $p_g \in Pref_c(w)$ with $g > 1$ and let $p_g = ps$ be a factorization of p_g with $p \neq p_g$, $p \neq 1$, $s \neq p_g$, $s \neq 1$. The circular word ${}^{\sim}w^t p_g \in cyclic(w)$ is generated by means of the rule*

$$R_{p_g} = wp \mid w \$ w \mid sw$$

and starting from ${}^{\sim}w^{t_1-1}pw \in cyclic(w)$ and ${}^{\sim}w^{t_2-1}sw \in cyclic(w)$, with $t_1 + t_2 = t$, $t_1 \geq 2$, $t_2 \geq 2$.

Proof. Let t_1, t_2 be positive integers with $t_1 \geq 2$, $t_2 \geq 2$ and let $p_g = ps$ be a factorization of p_g with $p \neq p_g$, $p \neq 1$, $s \neq p_g$, $s \neq 1$. By using Lemma 9.1, we have ${}^{\sim}w^{t_1-1}pw \in cyclic(w)$ and ${}^{\sim}w^{t_2-1}sw \in cyclic(w)$. Then, by using Paun's definition of circular splicing, it is easy to see that the conclusion holds. ■

We show that $cyclic(w)$ is spliced by the SC_{PA} defined using Lemmata 9.2 and 9.3, when w is a word whose length is greater than 2.

Lemma 9.4 *Let $w \in A^*$ be a word with $|w| > 2$, let $SC_{PA} = (A, I, R)$ be the circular splicing system defined by*

$$I = \{^\sim(w^t Pref_c(w)) \mid 0 \leq t \leq 4\}, \qquad R = \cup_{p_g \in Pref_c(w)} R_{p_g}$$

where the R_{p_g}'s are defined according to Lemmata 9.2 and 9.3. Then, $cyclic(w) \subseteq C(SC_{PA})$.

Proof. We prove that $^\sim w^t Pref_c(w) \subseteq C(SC_{PA})$ by induction on t and that for a fixed t, $^\sim w^t p_g \in C(SC_{PA})$, $p_g \in Pref_c(w)$, by induction over g. If $t \in \{0,1,2,3,4\}$ we have $^\sim w^t Pref_c(w) \subseteq I$ and so $^\sim w^t Pref_c(w) \subseteq C(SC_{PA})$. Let us suppose that $^\sim w^j Pref_c(w) \subseteq C(SC_{PA})$, for $j < t$ and let us prove that $^\sim w^t Pref_c(w) \subseteq C(SC_{PA})$. Indeed, for all $p_g \in Pref_c(w)$ with $g = 1$ (resp. $g > 1$), in virtue of Lemma 9.2 (resp. Lemma 9.3) the circular word $^\sim w^t p_g \in cyclic(w)$ is generated by means of the rule R_{p_g}, starting from $^\sim w^{t_1 - 1} p' w \in cyclic(w)$ and $^\sim w^{t_2 - 1} s' w \in cyclic(w)$, with $t_1 + t_2 = t - 1$ (resp. $t_1 + t_2 = t$, $|p'| < |p_g|$, $|s'| < |p_g|$), $t_1 \geq 2$, $t_2 \geq 2$. By induction hypothesis over t (resp. over g), $^\sim w^{t_1 - 1} p' w \in C(SC_{PA})$ and $^\sim w^{t_2 - 1} s' w \in C(SC_{PA})$. Consequently, $^\sim w^t p_g \in C(SC_{PA})$, by definition of circular splicing. ■

Example 9.1 *(continued)* By using Lemma 9.4, each word in $cyclic(abc)$ is generated by the circular splicing system $SC_{PA} = (A, I, R)$ defined by $I = \{^\sim((abc)^t Pref_c(abc)) \mid 0 \leq t \leq 4, \}$, $R = R_a \cup R_b \cup R_c \cup R_{ab} \cup R_{bc} \cup R_{ca}$, where $R_a = wab|w\$w|caw$, $R_b = wab|w\$wb|cw$, $R_c = wca|w\$w|bcw$, $R_{ab} = wa|w\$w|bw$, $R_{bc} = wb|w\$w|cw$, $R_{ca} = wc|w\$w|aw$. Thanks to Lemma 9.8, we will see that $cyclic(abc) = C(SC_{PA})$.

Using several intermediate results, we now prove that every word generated by the circular splicing system defined in Lemma 9.4, is an element of $cyclic(w)$. From now on, we set $Suff(w) = \{x \in A^+ \mid x \text{ is a proper suffix of } w\}$.

Proposition 9.1 *[51] Two words $x, y \in A^+$ are conjugate if and only if there exists $z \in A^*$ such that $xz = zy$. More precisely, the above equality holds if and only if there exist $u, v \in A^*$ such that $x = uv$, $y = vu$, $z \in u(vu)^*$.*

As a matter of fact, in [16], an unbordered word is defined as a word satisfying the property contained in the lemma below the proof of which is reported here for the sake of completeness.

Lemma 9.5 *For every unbordered word w with $|w| \geq 2$, we have $Pref(w) \cap Suff(w) = \emptyset$.*

Proof. By contradiction, suppose that $z \in Pref(w) \cap Suff(w)$. Then, there exist $x, y \in A^+$ such that $xz = w = zy$. Thus, by using Proposition 9.1, words $u, v \in A^*$ exist and a nonnegative integer t also exists such that $x = uv$, $y = vu$, $z = u(vu)^t$. Furthermore, since $z \neq 1$, we have either $u \neq 1$ or $v \neq 1$, $t \geq 1$. Consequently we have $w = u(vu)^{t+1}$ with $u \neq 1$ or $v \neq 1$, $t \geq 1$, which is in contradiction with the hypothesis of w being unbordered. ■

Lemma 9.6 *Let $w \in A^*$ be an unbordered word with $|w| \geq 2$. For all $p \in A^+$ with $|p| < |w|$, we have $w \notin Suff(w)p$.*

Proof. By contradiction, suppose that there exist $p \in A^+$ with $|p| < |w|$ and $s \in Suff(w)$ such that $w = sp$. Thus, we also have $s \in Pref(w) \cap Suff(w)$, in contradiction with Lemma 9.5. ■

Lemma 9.7 *Let $w \in A^*$ be an unbordered word with $|w| \geq 2$. For all $p, p' \in A^+$ with $|p| < |w|$, $|p'| < |w|$, for all $h \in A^*$, $q \in \boldsymbol{N}$, $q \geq 2$, if $hwp'w \sim w^q p$ then $h = w^{q-2}$ and $p = p'$. In particular the result holds when $p, p' \in Pref_c(w)$.*

Proof. Suppose that, for $p, p' \in A^+$ with $|p| < |w|$, $|p'| < |w|$, $h \in A^*$, $q \in \boldsymbol{N}$, $q \geq 2$, we have $hwp'w \sim w^q p$. Observe that if $h = 1$, then $q = 2$ ($q < 2$ is impossible and $q > 2$ implies $|p'| > |w|$ which is once again impossible). Two cases can occur:

1. $hwp'w = w'_2 w^g p w^{g'} w'_1$ with $w'_1 w'_2 = w$, $w'_1, w'_2 \in A^*$ and $g + g' = q - 1$,
2. $hwp'w = p_2 w^q p_1$ with $p_1 p_2 = p$, $p_1, p_2 \in A^*$.

Case 1. In the first case, w_1' is a prefix of w. Then, in virtue of Lemma 9.5, we have either $w_1' = 1$ or $w_1' = w$ and so we must have

$$hwp'w = w^{g+1}pw^{g'} \quad (*)$$

or

$$hwp' = w^{g}pw^{g'} \quad (**)$$

We notice that we cannot have equality $(*)$ with $g' \neq 1$ (otherwise we would also have either $sp' = w$ or $sp = w$, with $s \in Suff(w)$, in contradiction with Lemma 9.6) nor can we have equality $(**)$ with $g' \neq 0$ (otherwise, once again, we would also have $sp' = w$ with $s \in Suff(w)$, in contradiction with Lemma 9.6).

Thus, suppose that either $(*)$ holds with $g' = 1$ or $(**)$ holds with $g' = 0$. In both cases we have $hwp' = w^{q-1}p$. If $p = p'$ then $h = w^{q-2}$ and the proof is ended. Otherwise, either p' is a proper suffix of p or p is a proper suffix of p'. As a result, we also have either $hw = w^{q-1}p''$ with $p'' \in Pref(p)$ or $hwp'' = w^{q-1}$ with $p'' \in Pref(p')$. In both cases we have $w = sp''$ with $0 < |p''| < |w|$, $s \in Suff(w)$, in contradiction with Lemma 9.6.

Case 2. Assume that $hwp'w = p_2w^qp_1$ with $p_1p_2 = p$, $p_1, p_2 \in A^*$. Then, we have either $p_1 \neq 1$ or $p_1 = 1$. Correspondingly, we also have either $w = sp_1$ or $w = sp'$ with $s \in Suff(w)$. In both cases we are in contradiction with Lemma 9.6. ■

We prove that the SC_{PA} splicing system defined by using Lemmata 9.2 and 9.3 generates only words in $cyclic(w)$, when w is an unbordered word whose length is greater than 2.

Lemma 9.8 *Let $w \in A^*$ be an unbordered word with $|w| > 2$, let $SC_{PA} = (A, I, R)$ be the circular splicing system defined by*

$$I = \{ {}^{\sim}(w^t Pref_c(w)) \mid 0 \leq t \leq 4 \}, \quad R = \cup_{p_g \in Pref_c(w)} R_{p_g}$$

where the R_{p_g}'s are defined as in Lemmata 9.2 and 9.3. Then, $C(SC_{PA}) \subseteq cyclic(w)$.

Proof. Let us prove that $C(SC_{PA}) \subseteq cyclic(w)$ for the circular splicing system $SC_{PA} = (A, I, R)$ defined in the statement above. Let $^{\sim}z \in C(SC_{PA})$, we will prove that $^{\sim}z \in cyclic(w)$, by using induction over $|z|$. Since $I \subseteq cyclic(w)$, let us suppose that $^{\sim}z$ is generated by using one of the rules in R. Then, $^{\sim}z = {}^{\sim}u_2hu_1u_4ku_3$ with $u_1|u_2\$u_3|u_4 \in R$ and $^{\sim}hu_1u_2 \in C(SC_{PA})$, $^{\sim}ku_3u_4 \in C(SC_{PA})$. Since for every rule in R we have $u_i \neq 1$, for $i \in \{1,2,3,4\}$, then $|hu_1u_2| < |z|$, $|ku_3u_4| < |z|$ and so, by using induction hypothesis, $^{\sim}hu_1u_2 \in cyclic(w)$, $^{\sim}ku_3u_4 \in cyclic(w)$. Looking at the form of rules in R, there exist nonnegative integers q, j and $p, p', t, t' \in Pref_c(w)$ such that $hu_1u_2 = hwp'w \sim w^qp$, $ku_3u_4 = kwt'w \sim w^jt$. Consequently, $q \geq 2$, $j \geq 2$, and in virtue of Lemma 9.7, $h = w^{q-2}$, $p = p'$, $k = w^{j-2}$, $t = t'$. Finally, $z \sim u_2hu_1u_4ku_3$, and by looking at the form of the rules in R, we see that there exists $v \in Pref_c(w)$ such that for $^{\sim}u_2hu_1u_4ku_3$ one of the following four cases occur: $u_2hu_1u_4ku_3 \sim w^qvww^j$ or $u_2hu_1u_4ku_3 \sim w^qww^jv$ or $u_2hu_1u_4ku_3 \sim w^qwvw^j$ or $u_2hu_1u_4ku_3 \sim w^qvw^j$. Thus $^{\sim}z = {}^{\sim}u_2hu_1u_4ku_3 \in cyclic(w)$ and the conclusion holds. ■

Finally, in virtue of Lemmata 9.4 and 9.8, we have our result.

Proposition 9.2 *For all unbordered words $w \in A^*$, with $|w| > 2$, $cyclic(w)$ is C_{PA} generated.*

Proposition 9.3 (resp. Proposition 9.4) shows that the class of cyclic languages is different from the class of star languages (resp. fingerprint closed languages).

Proposition 9.3 *There exist unbordered words $w \in A^*$, with $|w| > 2$, such that $cyclic(w) \neq X^*$, for every subset X of A^*.*

Proof. For $w = abc$, we have $^{\sim}a \in cyclic(abc)$, $^{\sim}ab \in cyclic(abc)$ and $^{\sim}aab \notin cyclic(abc)$. Thus, $cyclic(abc) \neq X^*$, for every subset X of A^*. ■

Remark 9.2 Notice that we can easily prove that $cyclic(abc) \neq {}^{\sim}(X^+ \cup Y)$, for every pair of finite subsets X, Y of A^*. Indeed, by contradiction, suppose that such a pair X, Y exists with $cyclic(abc) = {}^{\sim}(X^+ \cup Y)$. Thus, a positive

integer n also exists such that $^{\sim}(abc)^n a \in cyclic(abc)\backslash {}^{\sim}Y$, $^{\sim}(abc)^n ab \in cyclic(abc)\backslash {}^{\sim}Y$. Now, we have a contradiction since $^{\sim}(abc)^n a(abc)^n ab \in {}^{\sim}(X^+ \cup Y) \backslash cyclic(abc)$.

The following observation will be used in the proof of Proposition 9.4.

Lemma 9.9 *For all words $w, w' \in A^+$ such that $|w'| = g|w|$, g being a positive integer, we have $^{\sim}w' \notin cyclic(w)$.*

Proof. By contradiction, suppose that for $w, w' \in A^+$ we have $|w'| = g|w|$ and $^{\sim}w' \in cyclic(w)$ with g being a positive integer. Using Definition 9.1, there exists a nonnegative integer q, and $p \in Pref_c(w)$ also exists such that $^{\sim}w' = {}^{\sim}w^q p$. Then, by using the above equalities we get $|w'| = g|w| = q|w| + t$, where $t = |p| < |w|$ which is a contradiction. ■

Proposition 9.4 *There exist unbordered words $w \in A^*$, with $|w| > 2$, such that $cyclic(w)$ is not a fingerprint closed language with respect to $\mathcal{A}$, for every trim deterministic automaton $\mathcal{A}$ recognizing $cyclic(w)$.*

Proof. For the sake of simplicity, take $w = abc$ (the argument below works for all words w).

Let $\mathcal{A} = (A, Q, \delta, q_0, F)$ be a trim deterministic automaton recognizing $cyclic(abc)$. Let v be the nonempty label of a closed path in $\mathcal{A}$, i.e., $\delta(q, v) = q$ for $q \in Q$, which is also a cycle. We can always suppose that v is a simple cycle in $\mathcal{A}$ (otherwise, using Definition 7.2, $v = u_1 c_1^{p_1} u_2 \dots u_k c_k^{p_k} u_{k+1}$ 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, $p_i \geq 1$. Furthermore, $\delta(q, u_1 c_1^{n_1} u_2 \dots u_k c_k^{n_k} u_{k+1}) = q$, for all $n_i \in \boldsymbol{N}$. In addition, we can suppose that $v' = u_1 u_2 \dots u_k u_{k+1}$ is a simple cycle and $\delta(q, v') = q$). We now prove that no positive integer n exists such that $v^n \in cyclic(abc)$, so $F(v) \notin cyclic(abc)$ and $cyclic(abc)$ is not a fingerprint closed language with respect to $\mathcal{A}$ (see Definition 7.4). Since $\mathcal{A}$ is trim, there exist $u, z \in A^*$ such that $uv^*z \in cyclic(abc)$. In particular, $uz \in cyclic(abc)$ which implies $uz \neq 1$ and, for $g, t \in \boldsymbol{N}$ with $0 < t < |abc|$, $|uz| = g|abc| + t$. On the other hand, $g', t' \in \boldsymbol{N}$ also exist such that $|v| = g'|abc| + t'$, $0 \leq t' < |abc|$. If $t' = 0$ we have ended the proof, thanks

to Lemma 9.9. Otherwise, $t, t' \in \{1,2\}$ and, given that $uvz \in cyclic(abc)$, using Lemma 9.9 once again, we have $t + t' \pmod 3 \in \{1,2\}$, i.e., $t = t' = 1$ or $t = t' = 2$. In both cases we have $uvvz \in cyclic(abc)$ with $|uvvz| = 0 \pmod 3$, which contradicts Lemma 9.9. ■

We have proven that $cyclic(w)$ is C_{PA} generated under the hypotheses that w is an unbordered word and $|w| > 2$. A natural question is whether the same result holds when we remove the above-mentioned hypotheses. We are not able to answer this question with respect to the first hypothesis. On the contrary, Example 9.2 shows that the hypothesis $|w| > 2$ in Proposition 9.2 is necessary.

Example 9.2 Let $w \in (\{a,b\})^2$. Thus, let us show that $cyclic(w)$ is not C_{PA} generated. Indeed, by contradiction, suppose that the contrary holds. We observe that every word w' in $cyclic(w)$ has an odd length, i.e., there exists a nonnegative integer n such that $|w'| = 2n + 1$. Since $cyclic(w)$ is not a finite language, for a sufficiently large n there exist two words $x, y \in cyclic(w)$, $|x| = 2k + 1, |y| = 2h + 1$, with $h, k \in \boldsymbol{N}$, such that w' is obtained by using a rule and by starting from x, y. On the other hand, looking at the definition of circular splicing, we see that the length of w' is the sum of the lengths of x and y. However, this sum is an even number and this is a contradiction.

9.2 Weak cyclic languages

This section contains Propositions 9.5 that suggests how we can modify $cyclic(w)$ defining a similar class of regular languages (*weak cyclic languages*), closed under conjugacy relation and C_{PA} generated. We will only define the weak cyclic language $Wcyclic(abc)$. Observe that $Wcyclic(abc)$ differs from $cyclic(abc)$ only in the last subset $^{\sim}(abc)^*ac$.

Proposition 9.5 *Consider the following regular language, closed under conjugacy relation:*

$$Wcyclic(abc) = {}^{\sim}(abc)^*a \cup {}^{\sim}(abc)^*b \cup {}^{\sim}(abc)^*c \cup$$
$${}^{\sim}(abc)^*ab \cup {}^{\sim}(abc)^*bc \cup {}^{\sim}(abc)^*ac.$$

$Wcyclic(abc)$ is C_{PA} generated by the splicing system $SC_{PA} = (A, I, R)$ where we set $w = abc$, $I = \{{}^{\sim}(w)^i a, {}^{\sim}(w)^i b, {}^{\sim}(w)^i c, {}^{\sim}(w)^i ab, {}^{\sim}(w)^i bc, {}^{\sim}(w)^i ac \mid 0 \leq i \leq 4\}$, $R = \{R_a, R_b, R_c, R_{ab}, R_{bc}, R_{ac}\}$ and $R_a = wab|w\$wa|cw$, $R_b = wab|w\$wb|cw$, $R_c = wa|cw\$w|bcw$, $R_{ab} = wa|w\$w|bw$, $R_{bc} = wb|w\$w|cw$, $R_{ac} = wa|w\$w|cw$.

Proof. Let us denote $Wcyclic(abc) = {}^{\sim}(abc)^*Pref_{wc}$, where $Pref_{wc} = \{a, b, c, ab, bc, ac\}$. We prove that ${}^{\sim}w^t Pref_{wc}(w) \subseteq C(SC_{PA})$ by using induction on t and, for a fixed t, that ${}^{\sim}w^t p_g \in C(SC_{PA})$, $p_g \in Pref_{wc}(w)$, by using induction over g. If $t \in \{0, 1, 2, 3, 4\}$ we have ${}^{\sim}w^t Pref_{wc}(w) \subseteq I$ and so ${}^{\sim}w^t Pref_{wc}(w) \subseteq C(SC_{PA})$. Let us suppose that ${}^{\sim}w^j Pref_{wc}(w) \subseteq C(SC_{PA})$, for $j < t$ and let us prove that ${}^{\sim}w^t Pref_{wc}(w) \subseteq C(SC_{PA})$. Indeed, for all $p_g \in Pref_{wc}(w)$ with $|p_g| = 1$ the circular word ${}^{\sim}w^t p_g \in Wcyclic(w)$ is generated by means of the rule R_{p_g}, starting from ${}^{\sim}w^{t_1-1}pw \in Wcyclic(w)$ and ${}^{\sim}w^{t_2-1}sw \in Wcyclic(w)$, with $t_1 + t_2 = t - 1$, $t_1 \geq 2$, $t_2 \geq 2$ (in virtue of Lemma 9.2 for R_b, by using Paun's definition of circular splicing for R_a and R_c). By using induction hypothesis over t, ${}^{\sim}w^{t_1-1}pw \in C(SC_{PA})$ and ${}^{\sim}w^{t_2-1}sw \in C(SC_{PA})$, consequently ${}^{\sim}w^t p_g \in C(SC_{PA})$, by using the definition of circular splicing.

Furthermore, for all $p_g \in Pref_{wc}(w)$ with $|p_g| > 1$ the circular word ${}^{\sim}w^t p_g \in Wcyclic(w)$ is generated by means of the rule R_{p_g}, starting from ${}^{\sim}w^{t_1-1}pw \in Wcyclic(w)$ and ${}^{\sim}w^{t_2-1}sw \in Wcyclic(w)$, with $t_1 + t_2 = t$, $ps = p_g$, $|p| < |p_g|$, $|s| < |p_g|$, $t_1 \geq 2$, $t_2 \geq 2$ (in virtue of Lemma 9.3 for R_{ab} and R_{bc}, and by using Paun's definition of circular splicing for R_{ac}). By using induction hypothesis over g, ${}^{\sim}w^{t_1-1}pw \in C(SC_{PA})$ and ${}^{\sim}w^{t_2-1}sw \in C(SC_{PA})$, consequently ${}^{\sim}w^t p_g \in C(SC_{PA})$, by using the definition of circular splicing.

Let us prove that $C(SC_{PA}) \subseteq Wcyclic(w)$ for the circular splicing system $SC_{PA} = (A, I, R)$ defined in the statement above. Let ${}^{\sim}z \in C(SC_{PA})$. We will prove that ${}^{\sim}z \in Wcyclic(w)$, by using induction over $|z|$. Since

$I \subseteq Wcyclic(w)$, let us suppose that $^{\sim}z$ is generated by using one of the rules in R. Then, $^{\sim}z = {}^{\sim}u_2hu_1u_4ku_3$ with $u_1|u_2\$u_3|u_4 \in R$ and $^{\sim}hu_1u_2 \in C(SC_{PA})$, $^{\sim}ku_3u_4 \in C(SC_{PA})$. Since for every rule in R we have $u_i \neq 1$, for $i \in \{1,2,3,4\}$, then $|hu_1u_2| < |z|$, $|ku_3u_4| < |z|$ and so, by using induction hypothesis, $^{\sim}hu_1u_2 \in Wcyclic(w)$, $^{\sim}ku_3u_4 \in Wcyclic(w)$. Looking at the form of rules in R, there exist nonnegative integers q, j and $p, p', t, t' \in Pref_{wc}(w)$ such that $hu_1u_2 = hwp'w \sim w^qp$, $ku_3u_4 = kwt'w \sim w^jt$. Consequently, $q \geq 2$, $j \geq 2$, and in virtue of Lemma 9.7, $h = w^{q-2}$, $p = p'$, $k = w^{j-2}$, $t = t'$. Finally, $z \sim u_2hu_1u_4ku_3$ and, by looking at the form of the rules in R, we see that there exists $v \in Pref_{wc}(w)$ such that for $^{\sim}u_2hu_1u_4ku_3$ one of the following three cases occur: $u_2hu_1u_4ku_3 \sim w^qvw^j$ or $u_2hu_1u_4ku_3 \sim w^qww^jv$ or $u_2hu_1u_4ku_3 \sim vw^qww^j$. Thus $^{\sim}z = {}^{\sim}u_2hu_1u_4ku_3 \in Wcyclic(w)$ and the conclusion holds. ■

9.3 From Paun splicing to Pixton splicing

Proposition 9.6 shows that $cyclic(w)$ is C_{PI} generated for each $w \in (\{a,b\})^2$. This language, together with others presented in Chapter 6, allows us to investigate Pixton systems, as will be done in the last chapter. We use Lemma 9.10 as a preliminary observation.

Lemma 9.10 *We have* $Lin(cyclic(ab)) = \{(ab)^sa(ab)^t, b(ab)^sa(ab)^ta, (ab)^s b(ab)^t, b(ab)^sb(ab)^ta \mid s,t \geq 0\}$. *Furthermore,* $cyclic(ab) = cyclic(ba)$.

Proof. We recall that $cyclic(ab) = {}^{\sim}(ab)^*a \cup {}^{\sim}(ab)^*b$. Thus, it is easy to show that $Lin(cyclic(ab)) = \{(ab)^sa(ab)^t, b(ab)^sa(ab)^ta, (ab)^s\ b(ab)^t, b(ab)^sb(ab)^ta \mid s,t \geq 0\}$.

Furthermore, we notice that each word in $Lin(cyclic(ab))$ belongs to $Lin(cyclic(ba))$. Indeed,

- $(ab)^sa(ab)^t \sim a(ba)^{s+t}$, $^{\sim}a(ba)^{s+t} \in cyclic(ba)$;

- $$(ab)^s b(ab)^t = \begin{cases} a(ba)^{s-1}b(ba)^t b, \\ {\sim}a(ba)^{s-1}b(ba)^t b \in cyclic(ba) & \text{for } s > 0, \\ (ba)^t b,\ {\sim}(ba)^t b \in cyclic(ba) & \text{for } s = 0; \end{cases}$$

- $b(ab)^s a(ab)^t a = (ba)^{s+1}a(ba)^t$, ${\sim}(ba)^{s+1}a(ba)^t \in cyclic(ba)$;

- $b(ab)^s b(ab)^t a = (ba)^s b(ba)^{t+1}$, ${\sim}(ba)^s b(ba)^{t+1} \in cyclic(ba)$.

Consequently, $cyclic(ab) \subseteq cyclic(ba)$. Symmetrically, $cyclic(ba) \subseteq cyclic(ab)$ (it suffices to substitute the a's with the b's and vice versa in the above equalities). ■

Proposition 9.6 *For each $w \in (\{a,b\})^2$, $cyclic(w)$ is C_{PI} generated.*

Proof. We prove the statement for $w = ab$ (Case 1) and for $w = aa$ (Case 2). The other cases (when $w = ba$ or $w = bb$) can be obtained by substituting the a's with the b's and vice versa in Cases 1 and 2.

Case 1. Suppose $w = ab$. We will prove that $cyclic(ab) = C(SC_{PI})$ by considering $SC_{PI} = (A, I, R)$ with $I = \{{\sim}a, {\sim}b, {\sim}aba, {\sim}abb\}$ and $R = R_a \cup R_b$, with $R_a = \{r_a, \overline{r}_a\}$, $r_a = (aab, ab; abab)$, $\overline{r}_a = (ab, aab; 1)$; $R_b = \{r_b, \overline{r}_b\}$, $r_b = (abb, ab; abab)$, $\overline{r}_b = (ab, abb; 1)$. Firstly, let us show that $C(SC_{PI}) \subseteq cyclic(ab)$, by using induction on the number of applications of the splicing operation. Clearly $I \subseteq cyclic(ab)$. Thus, consider ${\sim}w_1, {\sim}w_2 \in C(SC_{PI})$ such that $({\sim}w_1, {\sim}w_2) \vdash_{r_x, \overline{r}_x} {\sim}w'$, for $r_x, \overline{r}_x \in R_x$, $x \in \{a, b\}$. By induction hypothesis, ${\sim}w_1, {\sim}w_2 \in cyclic(ab)$. If $x = a$ (i.e., we consider $r_a, \overline{r}_a$), then ${\sim}w_1 = {\sim}\alpha\epsilon = {\sim}aab\epsilon \in cyclic(ab)$ and ${\sim}w_2 = {\sim}\alpha'\epsilon' = {\sim}ab\epsilon' \in cyclic(ab)$. This means that $aab\epsilon, ab\epsilon' \in \{(ab)^s a(ab)^t, b(ab)^s a(ab)^t a, (ab)^s b(ab)^t, b(ab)^s b(ab)^t a \mid s, t \geq 0\}$ (Lemma 9.10). This implies $\epsilon \in (ab)^*$ and $ab\epsilon' \in \{(ab)^s a(ab)^t, (ab)^s b(ab)^t \mid s, t \geq 0\}$. Thus, ${\sim}w' = {\sim}\epsilon abab\epsilon' \in cyclic(ab)$. Otherwise, if $x = b$ (i.e., we consider $r_b, \overline{r}_b$), then ${\sim}w_1 = {\sim}\alpha\epsilon = {\sim}abb\epsilon \in cyclic(ab)$, ${\sim}w_2 = {\sim}\alpha'\epsilon' = {\sim}ab\epsilon' \in cyclic(ab)$. We already know that $ab\epsilon' \in \{(ab)^s a(ab)^t, (ab)^s b(ab)^t \mid s, t \geq 0\}$ and, as above, ${\sim}abb\epsilon \in cyclic(ab)$ implies $\epsilon \in (ab)^*$ (Lemma 9.10). Thus, ${\sim}w' = {\sim}\epsilon abab\epsilon' \in cyclic(ab)$.

Vice versa, let us prove that $cyclic(ab) \subseteq C(SC_{PI})$, by induction on the length of $^{\sim}w' \in cyclic(ab)$. If $w' \in {}^{\sim}(ab)^i a$, with $i \in \{0,1\}$, then $^{\sim}w' \in I$ (similarly, if $w' \in {}^{\sim}(ab)^i b$, with $i \in \{0,1\}$, the same holds). Suppose that the statement holds for every $w' \in {}^{\sim}(ab)^i a$, with $1 < i < n$. Thus, $^{\sim}(ab)^{n-1}a \in C(SC_{PI})$ and we also have $^{\sim}aba \in I \subseteq C(SC_{PI})$. Consequently, starting from $^{\sim}(ab)^{n-1}a$ and $^{\sim}aba$, and by applying $r_a, \overline{r}_a$, we can generate $^{\sim}(ab)^{n-2}ababa = {}^{\sim}(ab)^n a$. Analogously, starting from $^{\sim}(ab)^{n-1}b \in C(SC_{PI})$ and $^{\sim}abb \in I \subseteq C(SC_{PI})$, and by applying rules $r_b, \overline{r}_b$, we can generate $^{\sim}(ab)^{n-2}\ ababb = {}^{\sim}(ab)^n b$.

Case 2. Suppose $w = aa$. We now show that $cyclic(aa) = C(SC_{PI})$, for $SC_{PI} = (A, I, R)$ with $I = \{^{\sim}a, ^{\sim}aaa\}$ and $R = \{r, \overline{r}\}$, $r = (aaa, aa; aaaa)$, $\overline{r} = (aa, aaa; 1)$. Notice that we have $cyclic(aa) = (a^2)^* a$.

Let us firstly show that $C(SC_{PI}) \subseteq cyclic(aa)$, by using induction on the length of $^{\sim}w' \in C(SC_{PI})$. Clearly $I \subseteq cyclic(aa)$. Thus, consider $^{\sim}w_1, ^{\sim}w_2 \in C(SC_{PI})$ such that $(^{\sim}w_1, ^{\sim}w_2) \vdash_{r,\overline{r}} {}^{\sim}w'$, for $r, \overline{r} \in R$. By using induction hypothesis, $^{\sim}w_1, ^{\sim}w_2 \in cyclic(aa)$. Then $^{\sim}w_1 = {}^{\sim}\alpha\epsilon = {}^{\sim}aaa\epsilon \in cyclic(aa)$ and $^{\sim}w_2 = {}^{\sim}\alpha'\epsilon' = {}^{\sim}aa\epsilon' \in cyclic(aa)$. Thus, we have $\epsilon \in (aa)^*$, $\epsilon' \in (aa)^* a$ and so $^{\sim}w' = {}^{\sim}\epsilon aaaaa\epsilon' \in cyclic(aa)$.

Vice versa, let us prove that $cyclic(aa) \subseteq C(SC_{PI})$ by using induction on the length of $^{\sim}w' \in cyclic(aa)$. If $^{\sim}w' = {}^{\sim}(aa)^i a$, with $i \in \{0,1\}$, then $w \in I$. Suppose that the statement holds for every $w' \in {}^{\sim}(aa)^i a$, with $i \geq 0$, $n > 1$, $i < n$. Thus, $^{\sim}(aa)^{n-1}a \in C(SC_{PI})$ and we also have $^{\sim}aaa \in I \subseteq C(SC_{PI})$. Consequently, starting from $^{\sim}(aa)^{n-1}a$ and $^{\sim}aaa$, and by applying $r, \overline{r}$, we can generate $^{\sim}(aa)^n a$. ■

The results proved in these chapters lead us to the conclusion that it is not easy to characterize the regular C_{PA} generated languages. These results also suggest further investigations of the analogous problem of finding those regular languages which are generated by the more powerful circular Pixton splicing systems, a problem that is treated in the next (and last) chapter.

Chapter 10

Finite Pixton splicing systems

It is known that the class of circular regular C_{PI} generated languages is larger than the class of circular regular C_{PA} or C_H generated languages. This statement is shown by Proposition 6.1 and by several examples, a collection of them is in Chapter 6 and has been reported below.

Example 10.1 We have already proven that $(aa)^*b$, $L_d = \{w \in A^* \mid \exists h, k \in \boldsymbol{N}.\ |w|_a = 2k+1, |w|_b = 2h+1\}$ and $A^* \setminus a^+ = A^*bA^* = (a^*ba^*)^*$ are regular languages which are closed under conjugacy and whose circularization cannot be C_H or C_{PA} generated (Examples 6.1, 6.3, 7.5). We have also shown that $cyclic(aa), cyclic(bb), cyclic(ab) = cyclic(ba)$ are regular circular languages not C_H or C_{PA} generated (Example 9.2). On the contrary, each of these circularizations is C_{PI} generated (Examples 6.2, 6.4, 7.6, Proposition 9.6).

In this chapter [4] we define a subclass of regular languages closed under conjugacy relation and we will give a new class of C_{PI} generated languages. The main motivation for doing it is that this class contains all the above-mentioned examples and a class of languages proved to be C_{PA} generated in Chapter 7.3 (Proposition 10.4, Proposition 10.5, Section 10.3). We enlighten

[4] work in progress, partially presented at WORDS01 [9]

that all languages in our knowledge Pixton generated are contained in this class.

Given a trim deterministic automaton $\mathcal{A}$ recognizing L, we recall that the definition of cycle (Definition 7.2) is a formalism for referring to some special labels c of the closed paths in $\mathcal{A}$. We also use the definition of a (finite) subset of these special labels, called fingerprints (Definition 7.3).

Let us briefly sketch our idea. Starting with a regular language closed under conjugacy relation, in Definition 10.3 we define a circular splicing system SC_{PI} generating $^{\sim}L$, i.e., $^{\sim}L \subseteq C(SC_{PI})$, by means of fingerprints of cycles (Proposition 10.1). Indeed, intuitively, if a rational not finite language is generated by splicing, we nust exibit rules which are able to produce words with a non bounded number of occurrences of such labels as factors. The real crux of the matter is to prove the other inclusion. Indeed, from a more general point of view, we observe that in order to prove that a (circular) language is generated by a (circular) splicing system, two steps must be performed: we must firstly exibit a (circular) splicing system generating the (circular) language (i.e., in our case, $^{\sim}L \subseteq C(SC_{PI})$). Secondly, we must prove that the (circular) splicing system is "consistent" with respect to the given language, i.e., that it generates only words in the language ($C(SC_{PI}) \subseteq ^{\sim}L$). In Definition 10.4, we will define a class of languages for which the system given in Definition 10.3 is "consistent".

Remark 10.1 Observe that Pixton splicing is more difficult to manage than Paun splicing. Indeed, in Chapter 7, we have studied the application of Paun splicing (in which the terms of splicing are "put" together) on words that belong to a fingerprint closed star language (submonoid which is closed under conjugacy relation and containing all fingerprints of the cycles of the trim deterministic finite state automaton recognizing the language). These hypotheses allow us to generate only words of the language, by insertion of the cycles, which belong to the language, through suitable splicing rules. Here we want to work on a more general context, by using the more powerful Pixton's definition, but where pieces of the terms of splicing may be lost or substituted, when splicing is performed.

From now on L will be a regular language L which is closed under the conjugacy relation. We also need two new definitions concerning states having a special property (Definition 10.1) and particular sets of labels associated with them (Definition 10.2).

Definition 10.1 (Multicyclic state) *Let $\mathcal{A} = (Q, A, \delta, q_0, F)$ be a trim deterministic finite state automaton recognizing a regular language $L = L(\mathcal{A})$. Let $q \in Q$ and $c_1, \ldots, c_k$ be cycles such that $\delta(q, c_i) = q$, for $i \in \{1, \ldots, k\}$. We say that q is a* multicyclic *state in $\mathcal{A}$ if $k > 1$ and $i, j \in \{1, \ldots, k\}$, $a, b \in A$, with $a \neq b$, exist such that $F(c_i) \in aA^*, F(c_j) \in bA^*$. We say that $C_q = \{F(c_1), \ldots, F(c_k)\}$ is the set of the* cycles associated with *q when C_q is a set of fingerprints of cycles c such that $\delta(q, F(c)) = q$ and C_q is a maximal object in the class of such sets for the order of set inclusion.*

Example 7.1 *(continued).* Consider again the finite state automaton $\mathcal{A} = (\{q_0, q\}, \{a, b, d\}, \delta, q_0, \{q_0\})$ depicted in Figure 7.1. Then, q is the unique multicyclic state in $\mathcal{A}$. The set of the cycles associated with q is $C_q = \{a, d, bb\}$.

Definition 10.2 (Product of cycles) *Let $\mathcal{A} = (Q, A, \delta, q_0, F)$ be a trim deterministic finite state automaton recognizing a regular language $L = L(\mathcal{A})$. Let q be a multicyclic state in $\mathcal{A}$ and let $C_q = \{F(c_1), \ldots, F(c_k)\}$ be the set of the cycles associated with q. The set $P_q = \{F(c_{i_1})F(c_{i_2}) \ldots F(c_{i_j}) \mid i_t \in \{1, \ldots, k\},\ i_t \neq i_{t'}\}$ is the set of the* products *associated with q.*

Example 7.1 *(continued).* Consider the finite state automaton $\mathcal{A} = (\{q_0, q\}, \{a, b, d\}, \delta, q_0, \{q_0\})$ depicted in Figure 7.1. Then, for the unique multicyclic state q in $\mathcal{A}$, having $C_q = \{a, d\}$ as the set of the cycles associated with it, $P_q = \{a, d, bb, ad, da, abb, bba, dbb, bbd,\ adbb, abbd, dabb, dbba, bbad, bbda\}$ is the set of the products associated with q.

10.1 A finite circular Pixton splicing system

The following lemma defines $SC_{PI} = (A, I, R)$ and states the finiteness of I and R. The proof of this result may be obtained thanks to Lemma 7.1.

This Lemma may be considered as an *Expansion Rules Step*, since rules expanding cycles are defined.

Lemma 10.1 *Let $\mathcal{A} = (Q, A, \delta, q_0, F)$ be a trim deterministic finite state automaton recognizing a regular language $L = L(\mathcal{A})$, which is closed under conjugacy relation. Let $\mathcal{C}(L) = \bigcup_{c\ cycle} F(c)$ be the set of the fingerprints $F(c)$ for L associated with c, for every cycle c in $\mathcal{A}$. Thus, $\mathcal{C}$ =C(L) is finite. Consider the set $I_1 = \{w \in A^* \mid \delta(q_0, w) \in F$ and $[\forall x, z \in A^*, c \in A^+,$ $q \in Q$ if $w = xc^kz$ with $\delta(q_0, x) = q$, $\delta(q, c) = q$, then either $c = F(c)$ is a cycle and $k = 1$ or q is a multiciclic state, $c \in P_q$ and $k = 1]\}$. Then, I_1 is finite. Furthermore, for every cycle c' in $\mathcal{A}$, we have $A^*F(c') \cap I_1 \neq \emptyset$.*
For each c', let $\gamma_{F(c')}$ be a fixed word in A^* such that $\gamma_{F(c')}F(c') \in I_1$ and consider the set R defined as follows. $R = \bigcup_{c\ cycle} R_c \cup \bigcup_{q\ multicyclic\ state} R_q$, where we set

$$R_c = \{(F(c), \gamma_{F(c)}F(c); F(c)), (\gamma_{F(c)}F(c), F(c); F(c))\}$$

if c is simple, otherwise (c not simple), we set

$$R_c = \{(F(c), \gamma_{F(c')}F(c'); F(c)), (\gamma_{F(c')}F(c'), F(c); F(c')) \mid$$
$$c' \in \mathcal{C}, \gamma_{F(c')}F(c') \in I_1\}$$

Finally,

$$R_q = \{(F(c)p, \gamma_{F(c)}F(c); F(c)p), (\gamma_{F(c)}F(c), F(c)p\ ; F(c)) \mid$$
$$F(c) \in C_q,\ p \in P_q,\ F(c)p \in P_q\}$$

Thus, R is also finite.

Proof. Thanks to Lemma 7.1, we have that $\mathcal{C}$ is finite. In order to prove that I_1 is finite, observe that $I_1 = I_1' \cup I_1''$, where I_1' is finite and defined in Lemma 7.1 and $I_1'' = \{w \in A^* \mid \delta(q_0, w) \in F$ and $[\forall x, z \in A^*, c \in A^+,$ $q \in Q$ if $w = xc^kz$ with $\delta(q_0, x) = q$, $\delta(q, c) = q$, then q is a multiciclic state, $c \in C_q$ and $k = 1]\}$. Let us show the finiteness of I_1''. Indeed, any word $w \in I_1''$ is the label of a successful path π such that π contains at most

$|C_q|+1$ occurrences of q, for each state $q \in Q$. Otherwise, it is not difficult to prove that $q = \delta(q, c_1) = \delta(q, c_2) = \cdots = \delta(q, c_{|C_q|}) = \delta(q, c_{|C_q|+1})$, and so there exist $c_i = c_j$, $i \neq j$, where $c = c_1 c_2 \cdots c_{|C_q|} c_{|C_q|+1} \in P_q$, a contradiction. Since $\mathcal{A}$ is deterministic, for each multiciclic state $q \in Q$, we have $|C_q| \leq |A|$, and so $|w| \leq |Q|(|A|+1)$. Thus, also I_1'' is finite.

Now, let us show that $|R|$ is finite.

Firstly, let us observe that, for any cycle $c' = F(c')$, since the automaton $\mathcal{A}$ is trim, there exist $x, z \in A^*$ such that $xF(c')z$ is the label of a successful path. Since L is closed under the conjugacy relation, $zxF(c')$ is also the label of a successful path. Then, looking at the definition of I_1 it is quite easy to see that, by eventually erasing some factors of zx, we can obtain a word $\gamma_{F(c')}$ such that $\gamma_{F(c')}F(c') \in I_1$.

Let us prove that $|R_c| = |\bigcup_{c \ simple \ cycle} R_c| + |\bigcup_{c \ not \ simple \ cycle} R_c||$ is finite.

If c is simple, then there is only one rule in R_c, and so $|\bigcup_{c \ simple \ cycle} R_c| = |\mathcal{C}|$, i.e., $\bigcup_{c \ simple \ cycle} R_c$ is finite. Otherwise, for any $F(c)$, such that c is not simple, there are $|\mathcal{C}|$ rules, i.e., for any $F(c)$ there is one rule for each $F(c')$. Thus, $|\bigcup_{c \ not \ simple \ cycle} R_c| = |\mathcal{C}| \cdot |\mathcal{C}|$, i.e., which is a finite cardinality. Thus, $|\bigcup_{c \ cycle} R_c|$ is finite.

Furthermore, since we have at most $|Q|$ multicyclic states, and, for each multicyclic state q, we have $|C_q|$ fingerprints, where $|C_q| \leq |\mathcal{C}|$. Thus, $|R_q| \leq |C_q| \cdot |P_q| \leq |\mathcal{C}| \cdot |P_q|$. P_q is also finite. Indeed it is easy to see that we can obtain all the elements in P_q by taking all subsets of C_q and for each such subset T, all possible permutations of its elements, when we see T as a sequence. Then, we have that $|P_q|$ is finite. Consequently, $\bigcup_{q \ multicyclic \ state} R_q$ is finite and so, R is also finite. ■

We now define the above-mentioned Pixton system, constructed starting with L and we prove that each word in $^\sim L$ can be generated by it (Proposition 10.1). In the proof of Proposition 10.5 we will give an example of a Pixton system constructed by means of Definition 10.3.

Definition 10.3 *Let $\mathcal{A} = (Q, A, \delta, q_0, F)$ be a trim deterministic finite state automaton recognizing a regular language $L = L(\mathcal{A})$. Let us consider $SC_{PI} =$*

(A, I, R), where $I = {}^{\sim}I_1$ and $R = \bigcup_{c \text{ cycle}} R_c \cup \bigcup_{q \text{ multicyclic state}} R_q$, I_1 and R being defined in Lemma 10.1.

Proposition 10.1 *Let $\mathcal{A} = (Q, A, \delta, q_0, F)$ be a trim deterministic finite state automaton recognizing a regular language $L = L(\mathcal{A})$ which is closed under the conjugacy relation. Let us consider a Pixton system $SC_{PI} = (A, I, R)$ given in Definition 10.3. Thus, ${}^{\sim}L \subseteq C = C(SC_{PI})$.*

Proof. Let L be a regular language closed under the conjugacy relation and let $\mathcal{A}$ be a trim deterministic automaton recognizing L. Let us consider the system $SC_{PI} = (A, I, R)$ given in Definition 10.3. Thus $I = {}^{\sim}I_1 \subseteq {}^{\sim}L$. Let us denote $C = C(SC_{PI})$ and let us prove that ${}^{\sim}L \subseteq C$.

Let ${}^{\sim}y \in {}^{\sim}L$. By induction on $|y|$, we prove that ${}^{\sim}y \in C$, i.e., ${}^{\sim}y$ is generated by the splicing system given above. If ${}^{\sim}y \in I$, then obviously we have ${}^{\sim}y \in C$.

Thus, suppose ${}^{\sim}y \notin I$. Since ${}^{\sim}y \in {}^{\sim}L$, each representative of ${}^{\sim}y$ is the label of a successful path. Furthermore, since ${}^{\sim}y \notin I$, we have $y = xc^rz$ with $x, z \in A^*$, $c \in A^+$, $\delta(q_0, x) = q$, $\delta(q, c) = q$, $\delta(q, z) \in F$ and one of the following cases occurs:

Conditions (*)

1. $c \neq F(c)$ is a cycle;
2. $c = F(c)$ is a cycle and $r > 1$;
3. c is not a cycle.

Choose x with $|x|$ maximal with respect to conditions $(*)$ (notice that y is a word with a finite length $|y|$; for each prefix x of y, we have $|x| \leq |y|$ and so this maximal length exists).

Let us prove that we cannot have the first case. Indeed, otherwise, $c \neq F(c)$ is a cycle, where we refer to the occurrence of the word c as the cycle c in the successful path y. Thus, by using Definition 7.3, c is not a simple cycle in ($\mathcal{A}$ (otherwise $c = F(c)$ when c is a simple cycle), as a factor of) y and $c = u(c')^t v$, where c' is a cycle in ($\mathcal{A}$, as a factor of) y and either $c' \neq F(c')$ or $c' = F(c')$ and $t > 1$. Furthermore, $u, v \neq 1$ (since

$u_1, u_{k+1} \neq 1$ in Definition 7.2). So, $y = x(u(c')^t v)^r z$ with c' satisfying one of the conditions $(*)$ and $|xu| > |x|$, against the maximality of $|x|$.

Let us suppose that the second case occurs, i.e., $c = F(c)$ is a cycle (simple or not) and $r > 1$. Since y is a successful path, then $y' = xc^{r-1}z$ is also a successful path, and, by using induction hypothesis, $^\sim y' \in C$. Consider the linearization zxc^r of $^\sim y$. Set $zxc^r = \epsilon\beta\epsilon'\beta'$, with $\epsilon = zxc^{r-2}$, $\epsilon' = 1$, $\alpha = c = F(c)$, $\alpha' = \gamma_{F(c)}F(c)$, $\beta = \beta' = F(c) = c$. Clearly, $R_c = \{(F(c), \gamma_{F(c)}F(c); F(c)), (\gamma_{F(c)}F(c), F(c); F(c))\} \subseteq R$. Moreover, $^\sim\epsilon\alpha = {}^\sim\epsilon\beta = {}^\sim zxc^{r-1} \in C$ and $^\sim\epsilon'\alpha' = {}^\sim\gamma_{F(c)}F(c) \in I_1 \subseteq C$: by using the splicing definition we have $^\sim zxc^r = {}^\sim y \in C$.

Finally, let us suppose that the third case holds, i.e., c is not a cycle. Observe that, in this case, the maximality of x implies $r = 1$. By contradiction, if $r > 1$, since c is not a cycle, also c^{r-1} is not a cycle and so $y = xc^{r-1}cz$ with $|xc^{r-1}| > |x|$, a contradiction.

Thus, $y = xcz \notin I$. We can distinguish two cases: either the state q such that $\delta(q, c) = q$ is also an internal state which is crossed by the transition $\delta(q, c) = q$, or the internal states crossed by the transition $\delta(q, c) = q$ are different from q.

In the latter case, we have $c = u(c')^t v$ where $u \in A^+$ is the label of a transition in which no state is crossed twice, $\delta(q, u) = q'$, $\delta(q', c') = q'$, $\delta(q', v) = q$ and c' is not a cycle. Let us prove that this case cannot occur. Indeed, otherwise, $y = x(u(c')^t v)^r z$ with c' satisfying one of the conditions $(*)$ and $|xu| > |x|$, once again against the maximality of $|x|$.

As a consequence, the first case hold, i.e., q is also an internal state which is crossed by the transition $\delta(q, c) = q$. Thus either q is not a multiciclic state (*Case a)*) or q is a multiciclic state (*Case b)*).

Case a). q is not a multiciclic state. We have two cases:

a1) The state q is not a multiciclic state, but q is also an internal state which is crossed by the transition $\delta(q, c) = q$ and there exists $c' \in A^+$, $t > 1$ such that $c = (c')^t$, $\delta(q, c') = q$.

a2) The state q is not a multiciclic state, but q is also an internal state which is crossed by the transition $\delta(q, c) = q$ and there exists $c', c'' \in$

$$A^+,\ c' \neq c'' \text{ such that } \delta(q,c') = q = \delta(q,c''),\ y = xcz = x(c'c'')z.$$

We now prove that case $a1$) cannot hold. Indeed, $y = x(c')^t z$. If c' is a cycle, then y also satisfies one of the first two conditions $(*)$ and we have already proven that $^\sim y \in C$. Otherwise, c' is not a cycle, $y = xc'(c')^{(t-1)}z$ with $(t-1) > 0$ and c' satisfies one of the conditions $(*)$ with $|xc'| > |x|$, which is against the maximality of $|x|$.

Thus, let us consider case a2). Since x is maximal, we can suppose that $c = c'c''$ where $c' \in A^+$ is the shortest word such that $\delta(q,c') = q$, $c = c'c''$, $c'' \in A^+$. We have that $c' = F(c')$ is a cycle since otherwise $y = xc'c''z = x(uc_1vc'')z$, where $u \in A^+$ and either c_1 is not a cycle or c_1 is a cycle but $c_1 \neq F(c_1)$. So, c_1 satisfies one of the conditions $(*)$ with $|xu| > |x|$, which is against the maximality of $|x|$. Moreover, $c'' = F(c'')$ is a cycle, otherwise $x(c'c'')z = x(c'uc_2v)z$ where either c_2 is not a cycle, or c_2 is a cycle but $c_2 \neq F(c_2)$ and so c_2 satisfies one of the conditions $(*)$ with $|xc'u| > |x|$, which is against the maximality of $|x|$.

Now, if q is not a multiciclic state, then it is easy to see that c', c'' are not simple cycles (intuitively, c', c'' are labels of different subtransitions of the same closed path). Thus, since x is maximal, $y = x(c'c'')z$. Since y is a successful path and $\delta(q,c') = q = \delta(q,c'')$, then $y' = xc'z$ is also a successful path, and, by using induction hypothesis, $^\sim y' \in C$. Consider the linearization $zxc'c''$ of $^\sim y$. Set $zxc'c'' = \epsilon\beta\epsilon'\beta'$, with $\epsilon = zx$, $\beta = c' = F(c') = \alpha$, $\epsilon' = 1$, $\beta' = c'' = F(c'')$ and $\alpha' = \gamma_{F(c'')}F(c'')$. Clearly, $R_c = \{(F(c'), \gamma_{F(c'')}F(c''); F(c')), (\gamma_{F(c'')}F(c''), F(c')\ ; F(c''))\} \subseteq R$. Moreover, $^\sim\epsilon\alpha = {}^\sim\epsilon\beta = {}^\sim zxc' \in C$ and $^\sim\epsilon'\alpha' = {}^\sim\gamma_{F(c'')}F(c'') \in I_1 \subseteq C$: by using the splicing definition we have $^\sim zxc'c'' = {}^\sim zxc = {}^\sim y \in C$.

Case b). q is a multiciclic state. Then $c \notin P_q$, otherwise $y \in I_1$ and, since x is maximal with respect to c, there exists $c_i \in C_q$ that is repeated in c such that $c = c_ic_{i+1}\cdots c_jc_iz'$, where $c_ic_{i+1}\cdots c_j, c_{i+1}\cdots c_jc_i \in P_q$. Since $y = xc_ic_{i+1}\cdots c_jc_iz'z$ is the label of a successful path, also $y' = xc_ic_{i+1}\cdots c_jz'z$ is the label of a successful path and $|y'| < |y|$. Thus $^\sim y' \in C$. Now, set $z'zxc_ic_{i+1}\cdots c_jc_i = \epsilon\beta\epsilon'\beta'$, with $\epsilon = z'zx$, $\beta = c_ic_{i+1}\cdots c_j$, $\epsilon' = 1$, $\beta' = F(c_i) = c_i$. Clearly, $R_q = \{(F(c_i)c_{i+1}\cdots c_j, \gamma_{F(c_i)}F(c_i);$ $F(c_i)c_{i+1}\cdots c_j), (\gamma_{F(c_i)}F(c_i), F(c_i)c_{i+1}\cdots c_j\ ; F(c_i))\} \subseteq R$, where $\alpha = F(c_i)$

$c_{i+1}\cdots c_j$, $\alpha' = \gamma_{F(c_i)}F(c_i)$. Since ${}^{\sim}\epsilon\alpha = {}^{\sim}z'zxc_ic_{i+1}\cdots c_j = {}^{\sim}y' \in C$ and ${}^{\sim}\epsilon'\alpha' = {}^{\sim}\gamma_{F(c_i)}F(c_i) \in I_1 \subseteq C$, by using the splicing definition, we have ${}^{\sim}z'zxc_ic_{i+1}\cdots c_jc_i = {}^{\sim}y \in C$. ■

Remark 10.2 It is evident that, for each regular language L closed under conjugacy relation, there is a finite number of Pixton splicing systems that can be defined according to Definition 10.3 and differing from each other only in $\gamma_{F(c)}$.

Remark 10.3 Clearly, a more precise definition of R_c, when c is not a simple cycle, may decrease the cardinality of R. This may be done by giving a more precise definition of the relation between $F(c)$ and $F(c')$. Moreover, observe that, following the proof of the Proposition 10.1, when c is a simple cycle, rules in R_c allows us to enlarge c. If c is not simple, when $c = c'$, rules in R_c allows us to enlarge c and when $c \neq c'$ we can generated words obtained by different trasversals of the same closed path.

Given a regular language L which is closed under the conjugacy relation, we have seen that we can define a Pixton splicing system SC_{PI} such that ${}^{\sim}L \subseteq C(SC_{PI})$ even if we do not make any supplementary hypothesis on L (Proposition 10.1). This result shows that in order to characterize Pixton languages, the real crux of the matter is to find a splicing system SC_{PI} such that the inverse inclusion $C(SC_{PI}) \subseteq {}^{\sim}L$ also holds. As already observed in Chapter 7.1, we are faced with the following problem. We explicitly notice that the property for c of being a cycle of $\mathcal{A}$ is strictly depending on the automaton $\mathcal{A}$. Even if we restrict ourselves to a fixed automaton $\mathcal{A}$, we can have that c is a cycle in $\mathcal{A}$ and c is also the label of another (not necessary closed) path in $\mathcal{A}$. This is the reason for which below we will define a class $\mathcal{F}$ of languages by means of a "consistence condition" (see Definition 10.4) that guarantees the consistence of the splicing system, i.e., $C(SC_{PI}) \subseteq {}^{\sim}L$ (Proposition 10.2). As a matter of fact, we are able to show that star languages generated by group codes belong to $\mathcal{F}$. However, $\mathcal{F}$ also contains other languages, as shown in Proposition 10.5.

In Definition 10.4, we introduce a class $\mathcal{F}$ of regular languages which are closed under the conjugacy relation and satisfying the "consistence con-

dition" with respect to the Pixton system SC_{PI} given in Definition 10.3. Proposition 10.2 shows that each language in $\mathcal{F}$ has a circularization which is generated by this Pixton system.

Definition 10.4 (Consistence condition) *Let $L = L(\mathcal{A})$ be a regular language closed under conjugacy relation and recognized by the trim deterministic finite state automaton $\mathcal{A}$.*
$L \in \mathcal{F}$ if and only if there exists a Pixton system $SC_{PI} = (A, I, R)$ as given in Definition 10.3 such that

- *for every $R_c = \{(F(c), \gamma_{F(c)}F(c); F(c)), (\gamma_{F(c)}F(c), F(c); F(c))\} \subseteq R$, for every $^{\sim}\epsilon, ^{\sim}\epsilon' \in A^*$, if $^{\sim}F(c)\epsilon$, $^{\sim}\gamma_{F(c)}F(c)\epsilon' \in {}^{\sim}L$ then $^{\sim}\epsilon F(c)\epsilon' F(c) \in {}^{\sim}L$;*
- *for every $R_c = \{(F(c), \gamma_{F(c')}F(c'); F(c)), (\gamma_{F(c')}F(c'), F(c); F(c')) | c' \in \mathcal{C}, \gamma_{F(c')}F(c') \in I_1\} \subseteq R$, for every $^{\sim}\epsilon, ^{\sim}\epsilon' \in A^*$, if $^{\sim}F(c)\epsilon$, $^{\sim}\gamma_{F(c')}F(c')\epsilon' \in {}^{\sim}L$ then $^{\sim}\epsilon F(c)\epsilon' F(c') \in {}^{\sim}L$;*
- *for every $R_q = \{(F(c)p, \gamma_{F(c)}F(c); F(c)p), (\gamma_{F(c)}F(c), F(c)p\ ; F(c)) \mid F(c) \in C_q,\ p \in P_q,\ F(c)p \in P_q\} \subseteq R$, for every $^{\sim}\epsilon, ^{\sim}\epsilon' \in A^*$, if $^{\sim}F(c)p\epsilon$, $^{\sim}\gamma_{F(c)}F(c)\epsilon' \in {}^{\sim}L$ then $^{\sim}\epsilon F(c)p\epsilon' F(c) \in {}^{\sim}L$.*

Loosely speaking we force the consistence of the system (i.e., $C(SC_{PI}) = C \subseteq {}^{\sim}L$) through the consistence condition. In this way, if two words contain sites of a rule then the spliced word belongs to $^{\sim}L$.

Proposition 10.2 *Let L be a regular language such that $L \in \mathcal{F}$, let $SC_{PI} = (A, I, R)$ be a Pixton system as given in Definition 10.3. Then, $C(SC_{PI}) = C \subseteq {}^{\sim}L$.*

Proof. We prove that $C(SC_{PI}) = C \subseteq {}^{\sim}L$ by induction on the number of applications of the rules for generating $^{\sim}w'' \in C$. Since $I \subseteq {}^{\sim}L$, we can suppose that $^{\sim}w'' \in C \setminus I$ and that the conclusion holds for circular words obtained with a number of applications of the rules which is smaller than the one for generating $^{\sim}w''$. Since $^{\sim}w'' \in C \setminus I$, then there exist $^{\sim}w, ^{\sim}w' \in C$ and $r, \bar{r} \in R$ such that $^{\sim}w, ^{\sim}w' \vdash_{r,\bar{r}} {}^{\sim}w''$. Clearly, $^{\sim}w, ^{\sim}w' \in {}^{\sim}L$ since they

are generated in a number of steps which is smaller than the one for for generating $\sim w''$. Now, let us consider $\{r, \bar{r}\} \in \bigcup_{c\ cycle} R_c$. If c is simple, then $\{r, \bar{r}\} = R_c = \{(F(c), \gamma_{F(c)}F(c); F(c)), (\gamma_{F(c)}F(c), F(c); F(c))\} \subseteq R$, and so $\sim w = \sim \epsilon\alpha = \epsilon F(c)$, $\sim w' = \sim \epsilon'\alpha' = \sim \epsilon'\gamma_{F(c)}F(c)$ and, since $L \in \mathcal{F}$, $\sim w'' = \sim \epsilon\beta\epsilon'\beta' = \sim \epsilon F(c)\epsilon' F(c) \in \sim L$. If c is not simple, then $\{r, \bar{r}\} = R_c = \{(F(c), \gamma_{F(c')}F(c'); F(c)), (\gamma_{F(c')}F(c'), F(c); F(c'))\}$ and $\sim w = \sim \epsilon\alpha = \sim \epsilon F(c) \in \sim L$, $\sim w' = \sim \epsilon'\alpha' = \sim \epsilon'\gamma_{F(c')}F(c') \in \sim L$ and, since $L \in \mathcal{F}$, $\sim \epsilon F(c)$ $\epsilon' F(c') \in \sim L$. If $\{r, \bar{r}\} \in \bigcup_{q\ multicyclic} R_q$, i.e., $\{r, \bar{r}\} = R_q = \{(F(c)p, \gamma_{F(c)}$ $F(c); F(c)p)$, $(\gamma_{F(c)}F(c), F(c)p\ ;\ F(c))\}$, then $\sim w = \sim \epsilon\alpha = \sim \epsilon F(c)p \in \sim L$, $\sim w' = \sim \epsilon'\alpha' = \sim \epsilon'\gamma_{F(c)}F(c) \in \sim L$ and, since $L \in \mathcal{F}$, $\sim \epsilon F(c)p\epsilon' F(c) \in \sim L$. ■

Proposition 10.3 *Let L be a regular language such that $\sim L \subseteq C(SC_{PI})$, where $SC_{PI} = (A, I, R)$ is a Pixton system given in Definition 10.3 and such that $L \in \mathcal{F}$. Then, $C(SC_{PI}) = \sim L$.*

Proof. The conclusion easily follows by using Propositions 10.1 and 10.2. ■

By following the idea already used in the linear case (Section 3.1), the consistence condition may be considered as follows: a regular language L is closed with respect to the expansion rules (as defined in Lemma 10.1) if the consistence condition is satisfied, i.e., for every $\epsilon, \epsilon' \in A^*$, if $\sim F(c)\epsilon$, $\sim \gamma_{F(c)}F(c)\epsilon' \in \sim L$ then $\sim \epsilon F(c)$ $\epsilon' F(c) \in \sim L$, where $\gamma_{F(c)}$ is a word of finte length in L (similarly for the second and the third conditions). It could be interesting to investigate what the consistence condition means directly on L, since it expresses a combinatorial property of L.

10.2 Pixton splicing and codes

We show that the class of the free monoids X^* generated by a regular group code X is a subclass of $\mathcal{F}$ (Proposition 10.4), by using Proposition 7.2.

Proposition 10.4 *Let X^* be a free monoid generated by a regular group code X. Then $X^* \in \mathcal{F}$.*

Proof. We recall that if X is a regular group code, then X is a biprefix code. Furthermore X is biprefix if and only if X^* is biunitary (see proof of Proposition 7.2).

In order to prove that $X^* \in \mathcal{F}$, let $r, \bar{r} \in R_c = \{(F(c), \gamma_{F(c')}F(c'), F(c)), (\gamma_{F(c')}F(c'), F(c),\ F(c'))\} \subseteq R$, and let $^\sim\epsilon, ^\sim\epsilon' \in A^*$ such that $^\sim\epsilon F(c)$, $^\sim\epsilon'\gamma_{F(c')}F(c') \in {}^\sim X^*$. Thus, $\gamma_{F(c')}F(c')\epsilon' \in X^*$ and we know that $\gamma_{F(c')}F(c') \in I_1 \subseteq X^*$ (Definition 10.3). Since X^* is right unitary, we get $\epsilon' \in X^*$ and so $^\sim\epsilon' \in {}^\sim X^*$. Furthermore, $^\sim F(c)\epsilon \in {}^\sim X^*$ and $\epsilon F(c) \in X^*$, since X^* is closed under conjugacy relation. Moreover, by Proposition 7.2, $F(c') \in X^*$ and, since X^* is a submonoid, we get $\epsilon' F(c') \in X^*$ and, finally, $\epsilon F(c)\epsilon' F(c') \in X^*$ and so $^\sim\epsilon F(c)\epsilon' F(c') \in {}^\sim X^*$. The same argument may be applied for R_c when c is a simple cycle.

Similarly, let $R_q = \{(F(c)p, \gamma_{F(c)}F(c); F(c)p), (\gamma_{F(c)}F(c), F(c)p\,; F(c))\} \subseteq R$ and $\epsilon, \epsilon' \in A^*$ such that $^\sim\epsilon F(c)p \in {}^\sim X^*$, $^\sim\gamma_{F(c)}F(c)\epsilon' \in {}^\sim X^*$. Thus $\gamma_{F(c)}F(c)\epsilon' \in X^*$. Moreover, by construction, $\gamma_{F(c)}F(c) \in I_1 \subseteq X^*$. Since X^* is right unitary, then $\epsilon' \in X^*$ and so $^\sim\epsilon' \in {}^\sim X^*$. Nevertheless, $\epsilon F(c)p \in X^*$, $F(c) \in X^*$ and so $\epsilon F(c)pF(c) \in X^*$ and also $^\sim F(c)\epsilon F(c)p \in {}^\sim X^*$. Finally, $^\sim F(c)\epsilon F(c)p\epsilon' \in {}^\sim X^*$. ■

We recall that, by Proposition 6.1, the class of languages X^* which are free monoids with X being a regular group code is C_{PI} generated. Proposition 10.4 states that they belong to $\mathcal{F}$. Nevertheless, Proposition 10.5 shows that $\mathcal{F}$ also contains other languages.

Proposition 10.5 *$L = (a^*ba^*)^*$ is a star language (not fingerprint closed) which is not generated by a regular group code and which belongs to $\mathcal{F}$.*

Proof. Consider the finite state automaton $\mathcal{A}$ recognizing $L = L(\mathcal{A}) = (a^*ba^*)^* = A^* \setminus a^+$ and depicted in Figure 7.3. We yet know that $L = L(\mathcal{A}) = (a^*ba^*)^* = A^* \setminus a^+$ is a star language not fingerprint closed with respect to any automaton (see Example 7.5). Let us construct a Pixton system SC_{PI} following Definition 10.3.

A word $w \in L$ can be written as $w = a^{n_1}ba^{n_2}\ldots ba^{n_k}$ with $k \geq 2$ and $n_i \geq 0$. Looking at the definition of I, we see that $w \in I_1$ if and only if

either $w = a^{n_1}ba^{n_2}$ with $n_1 \in \{0,1,2\}$, $n_2 \in \{0,1\}$ or $w = a^{n_1}ba^{n_2}ba^{n_3}$ with $n_1 \in \{0,1,2\}, n_2, n_3 \in \{0,1\}$ and $n_2 n_3 = 0$. Thus, we get $I = 1 \cup {\sim}\{b, ba, bb, bab, bba, ab, aab, aba, aaba, abb, aabb, abab, aabab, abba, aabba\}$.

Let us construct R. There are only two (simple) cycles in $\mathcal{A}$, i.e., $a = F(a), b = F(b)$. We also choose $b = \gamma_{F(a)} = \gamma_{F(b)}$. Thus $R_a = \{(a, ba; a), (ba, a; a)\}$ and $R_b = \{(b, bb; b), (bb, b; b)\}$. Say q the final state in $\mathcal{A}$ different from q_0. We have that q is the unique multicyclic state in $\mathcal{A}$ and $C_q = \{a, b\}$, $P_q = \{a, b, ab, ba\}$. Thus we have $R_q = \{(ab, ba; ab), (ba, ab; a), (ba, bb; ba), (bb, ba; b)\}$. Consider $SC_{PI} = (A, I, R)$, with I, R as above. Let us prove that $L \in \mathcal{F}$. Firstly we show that for every $\epsilon, \epsilon' \in A^*$, if ${\sim}a\epsilon$, ${\sim}ba\epsilon' \in {\sim}L$, then ${\sim}\epsilon a\epsilon' a \in {\sim}L$. Indeed, since $L = A^* \setminus a^+$ and $|a\epsilon|_b \geq 1$, we have that the condition is satisfied. Analogously, if ${\sim}b\epsilon$, ${\sim}bb\epsilon' \in {\sim}L$, then ${\sim}\epsilon b\epsilon' b \in {\sim}L$, once again since $L = A^* \setminus a^+$ and $|b\epsilon|_b \geq 1$. In order to conclude that $L \in \mathcal{F}$, suppose that ${\sim}ab\epsilon$, ${\sim}ba\epsilon' \in {\sim}L$ or ${\sim}ba\epsilon$, ${\sim}bb\epsilon' \in {\sim}L$. Thus ${\sim}\epsilon ba\epsilon' a$, ${\sim}\epsilon ba\epsilon' b \in {\sim}L$ since $L = A^* \setminus a^+$. ■

In Remark 10.4, we give some observations concerning relationship between a well-known subclass of regular languages (*star free languages*) and languages whose circularization is C_{PA} or C_{PI} generated. Further investigations are in progress. We recall that a (regular) language$L \subseteq A^*$ is star free if L can be obtained from subsets of A by a finite number of set products, union and complements. For example, $A^* = \neg\emptyset$, $(ab)^* = A^* \setminus (bA^* \cup A^*a \cup A^*aaA^* \cup A^*bbA^*)$, $A^* \setminus a^+ = A^*ba^* = (a^*ba^*)^*$ are star free languages.

Remark 10.4 The class of languages whose circularization is C_{PA} generated cannot be compared with star free languages. Indeed, $(aa)^*$ is not star free, but ${\sim}(aa)^*$ is C_{PA} generated, while $(a^*ba^*)^* = A^* \setminus a^+ = A^*bA^*$ is star free, but, as already said, ${\sim}(a^*ba^*)^*$ is not C_{PA} generated. On the contrary, we do not know whether the class of languages whose circularization is C_{PI} generated can be compared with star free languages. Indeed, we will see that the the star free (star) language $(a^*ba^*)^*$ belongs to the family $\mathcal{F}$ as well as $(aa)^*$ (which is not star free). Thus, we have that there exist star languages which are not star free and whose circularizations are C_{PI} generated.

10.3 Examples

Let us denote $\mathcal{E} = \{\sim(aa)^*b, cyclic(aa), cyclic(bb), cyclic(ab), \sim L_d\}$, where L_d is L_2 in Example 6.3. We have already observed in Example 10.1 that each language in $\mathcal{E}$ cannot be C_{PA} generated.

We now prove that for each $C \in \mathcal{E}$ we have $Lin(C) \in \mathcal{F}$ and so C is C_{PI} generated (Proposition 10.3). In order to do so, thanks to Definition 10.4, we must exibit the $SC_{PI} = (A, I, R)$ according to Definition 10.3, such that the conditions on R are satisfied.

Proposition 10.6 $L = Lin(\sim(aa)^*b) \in \mathcal{F}$.

Proof. We preliminary observe that $L = \{w \in A^* \mid |w|_b = 1,\ |w|_a = 2g,\ g \geq 0\}$. Consider the trim deterministic finite state automaton depicted in Figure 10.1 and recognizing $L = L(\mathcal{A}) = Lin(\sim(aa)^*b) = (aa)^*\{b, aba\}(aa)^*$.

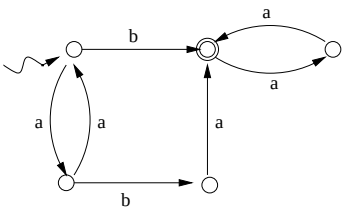

Figure 10.1: *Automaton for* $(aa)^*\{b, aba\}(aa)^*$.

Observe that we have only one (simple) cycle $aa = F(aa)$ and that no multicyclic state q exists in Q. Moreover, a word $w \in L$ can be written as $w = (aa)^h b(aa)^k$ or $w = (aa)^h aba(aa)^k$, with $h, k \geq 0$. Looking at the definition of I (Definition 10.3), we see that we must force $h, k \in \{0, 1\}$ and so $I = \sim\{b, aab, aabaa\}$.
Then we choose $\gamma_{F(aa)} = b$ and so $R = R_{aa} = \{(aa, baa; aa), (baa, aa; aa)\}$.
Now, if $\sim aa\epsilon$,$\sim baa\epsilon' \in \sim(aa)^*b$, we have that $\sim\epsilon \in \sim(aa)^*b$ and $\sim\epsilon' \in \sim(aa)^*$.
Thus, $\sim\epsilon aa\epsilon' aa \in \sim(aa)^*b$, since $|\epsilon aa\epsilon' aa|_a = 2g$ and $|\epsilon aa\epsilon' aa|_b = 1$. ■

Proposition 10.7 $L = Lin(\sim(aa)^*a) = Lin(cyclic(aa)) = (aa)^*a \in \mathcal{F}$.

Proof. We preliminary observe that $L = \{w \in a^* \mid |w|_a = 2g+1,\ g \geq 0\}$. Consider the trim deterministic finite state automaton depicted in Figure 10.2 and recognizing $L = L(\mathcal{A}) = Lin(cyclic(aa)) = (aa)^*a \in \mathcal{F}$.

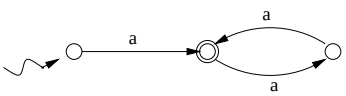

Figure 10.2: *Automaton for* $(aa)^*a$.

Observe that we have only one (simple) cycle aa and that no multicyclic state q exists in Q. Looking at the definition of I (Definition 10.3), we see that in a word $w = a^{2g}a$ we must force $g \in \{0,1\}$ and so $I = {}^{\sim}\{a, aaa\}$.
Let us choose $\gamma_{F(aa)} = a$ and so $R_{aa} = \{(aa, aaa; aa), (aaa, aa; aa)\}$.
Now, if ${}^{\sim}aa\epsilon$,${}^{\sim}aaa\epsilon' \in {}^{\sim}(aa)^*a$, we have that ${}^{\sim}\epsilon \in {}^{\sim}(aa)^*a$ and $\epsilon' \in {}^{\sim}(aa)^*$.
Thus, ${}^{\sim}\epsilon aa\epsilon' aa \in {}^{\sim}(aa)^*a$, since $|\epsilon aa\epsilon' aa|_a = 2g'+1, g' \geq 0$. ■

Proposition 10.8 $L = Lin({}^{\sim}(bb)^*b) = Lin(cyclic(bb)) = (bb)^*b \in \mathcal{F}$.

Proof. The conclusion easily follows by using the same argument as in Proposition 10.7. ■

Proposition 10.9 $L = Lin({}^{\sim}cyclic(ab)) = \{(ab)^s a(ab)^t, b(ab)^s a(ab)^t a, (ab)^s b(ab)^t, b(ab)^s b(ab)^t a \mid s,t \geq 0\} \in \mathcal{F}$.

Proof. By Lemma 9.10, we recall that $Lin(cyclic(ba)) = Lin(cyclic(ab))$. Consider the trim deterministic finite state automaton depicted in Figure 10.3 and recognizing $L = L(\mathcal{A}) = Lin({}^{\sim}cyclic(ab)) = \{(ab)^s a(ab)^t, b(ab)^s a(ab)^t a, (ab)^s b(ab)^t, b(ab)^s b(ab)^t a \mid s,t \geq 0\}$.
Observe that we have two (simple) cycles $ab = F(ab)$ and $ba = F(ba)$ and that no multicyclic state q exists in Q. Looking at the definition of I (Definition 10.3), we see that we must force $s,t \in \{0,1\}$ in $w \in \{(ab)^s a(ab)^t, b(ab)^s a(ab)^t a, (ab)^s b(ab)^t, b(ab)^s b(ab)^t a \mid s,t \geq 0\}$ and so

$$I = {}^{\sim}\{b, aab, aba, abaab, baa, baaba, babaa, babaaba,$$
$$b, bab, abb, abbab, bba, bbaba, babba, babbaba\}$$

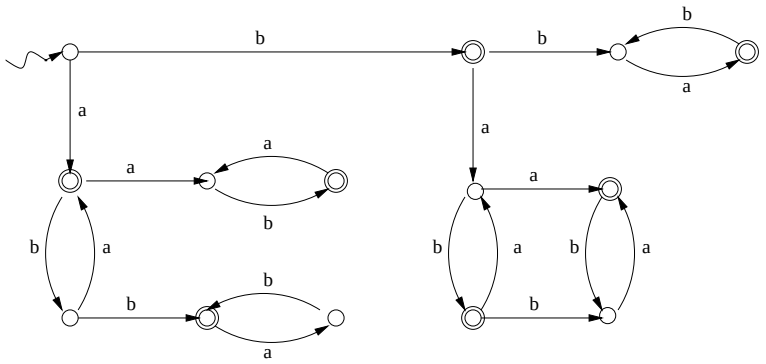

Figure 10.3: *Automaton for* $Lin(cyclic(ab))$.

Then we choose $\gamma_{F(ab)} = a$, $\gamma_{F(ba)} = b$ and so

$$R_{ab} = \{(ab, aab; ab), (aab, ab; ab)\}, R_{ba} = \{(ba, bba; ba), (bba, ba; ba)\}$$

Let us consider the condition in Definition 10.4 with respect to R_{ab}. Suppose that $^{\sim}ab\epsilon \in cyclic(ab)$, i.e., $ab\epsilon \in Lin(cyclic(ab))$. Then $\epsilon \in (ab)^*a(ab)^*$ or $\epsilon \in (ab)^*b(ab)^*$. Similarly, if $^{\sim}aab\epsilon' \in cyclic(ab)$, i.e. $aab\epsilon' \in Lin(cyclic(ab))$, then $\epsilon' \in (ab)^*$. Thus, $\epsilon ab\epsilon' ab \in (ab)^*a(ab)^+ \cup (ab)^*b(ab)^+$ and so $^{\sim}\epsilon ab\epsilon' ab \in cyclic(ab)$.

Let us consider the condition in Definition 10.4 with respect to R_{ba}. Suppose that $^{\sim}ba\epsilon \in cyclic(ab)$, i.e., $ba\epsilon \in Lin(cyclic(ab))$. Then, $\epsilon \in b(ab)^*a(ab)^*a \subseteq (ba)^*a(ba)^*$ or $\epsilon \in b(ab)^* \cup b(ab)^*b(ab)^*a \subseteq (ba)^*b(ba)^*$. Similarly, if $^{\sim}bba\epsilon' \in cyclic(ab)$, i.e., $bba\epsilon' \in Lin(cyclic(ab))$, then $\epsilon' \in b(ab)^*a \subseteq (ba)^*$. Thus, $\epsilon ba\epsilon' ba \in (ba)^*a(ba)^+ \cup (ba)^*b(ba)^+ \subseteq Lin(cyclic(ba))$. Since $Lin(cyclic(ba)) = Lin(cyclic(ab))$, we have $^{\sim}\epsilon ba\epsilon' ba \in cyclic(ab)$. ■

Proposition 10.10 $L_d = \{w \in A^* \mid \exists h, k \in \boldsymbol{N}.|w|_a = 2h + 1, |w|_b = 2k + 1\} \in \mathcal{F}$.

Proof. Consider the automaton $\mathcal{A}$ depicted in Figure 10.4 and recognizing $L_d = L(\mathcal{A})$.

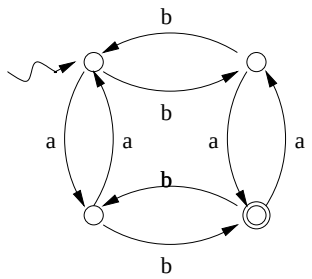

Figure 10.4: Automaton for $Lin(L_d)$)

Observe that each state in $\mathcal{A}$ is a multiciclic state and that we have the following set of cycles $\{a(b(aa)^ib)^ja, b(a(bb)^ia)^j)b, a(bb)^ib(aa)^jab, b(aa)^ia(bb)^j ba \mid i,j \leq 1\}$.

We assume that $SC_{PI} = (A, I, R)$ is the Pixton system constructed according to Definition 10.3.

We preliminary prove that for each fingerprint $F(c)$ we have $|F(c)|_a = 2h$ and $|F(c)|_b = 2k$. Indeed, if c is simple, then $c \in \{aa, bb, abab, baba\}$. Otherwise, using Definitions 7.2 - 7.3, there exist $u_1, \ldots, u_{k+1} \in A^*$, cycles $c_1, \ldots, c_k \in A^+$, positive integers $p_1, \ldots, p_k$ such that $c = u_1 c_1^{p_1} u_2 \cdots u_k c_k^{p_k} u_{k+1}$ and $F(c) = u_1 F(c_1) u_2 \cdots u_k F(c_k) u_{k+1}$ (where each u_i is the label of a transition in which no state is crossed twice and $u_1, u_{k+1} \neq 1$). Since $u_1 u_2 \cdots u_{k+1}$ is a simple cycle $c' = F(c')$, the conclusion easily follows by induction.

Let us prove now that $L_d \in \mathcal{F}$. Observe that, given a cycle c, the rules in R_c have the same structure, both in the case c simple cycle or not simple cycle. Thus, for each cycle c consider $R_c = \{(F(c), \gamma_{F(c)}F(c); F(c)), (\gamma_{F(c)}F(c), F(c); F(c))\}$. Notice that, since $\gamma_{F(c)}F(c) \in L_d$, $|\gamma_{F(c)}|_a$ and $|\gamma_{F(c)}|_b$ are both odd numbers. The same condition holds for $|\epsilon|_a$ and $|\epsilon|_b$ when ${}^\sim\epsilon F(c) \in {}^\sim L_d$. On the contrary, if $\epsilon'\gamma_{F(c)}F(c) \in {}^\sim L_d$ we have that $|\epsilon'|_a$ and $|\epsilon'|_b$ are both even numbers. Consequently, if ${}^\sim\epsilon F(c) \in {}^\sim L_d$ and $\epsilon'\gamma_{F(c)}F(c) \in {}^\sim L_d$ we have that $|\epsilon F(c)\epsilon' F(c)|_a$ and $|\epsilon F(c)\epsilon' F(c)|_b$ are both odd numbers and so ${}^\sim\epsilon F(c)\epsilon' F(c) \in {}^\sim L_d$. The same argument works for R_c, when c is not a simple cycle, as said before.

Let q be a multicyclic state and let us consider $R_q = \{(F(c)p, \gamma_{F(c)}F(c);$

$F(c)p), (\gamma_{F(c)}F(c), F(c)p\,; F(c)) \mid F(c) \in C_q,\ p \in P_q,\ F(c)p \in P_q\}$. Looking at the definition of P_q, we see that for each $p \in P_q$, we have $|p|_a = 2g$, $|p|_b = 2m$. Thus, if ${}^{\sim}\epsilon F(c)p \in {}^{\sim}L_d$, then $|\epsilon|_a$ and $|\epsilon|_b$ are both odd numbers. We have already observed that $|\gamma_{F(c)}|_a$ and $|\gamma_{F(c)}|_b$ are both odd number and that if $\epsilon'\gamma_{F(c)}F(c) \in {}^{\sim}L_d$, $|\epsilon'|_a$ and $|\epsilon'|_b$ are both even numbers. Consequently, if ${}^{\sim}F(c)p\epsilon \in {}^{\sim}L_d$ and $\gamma_{F(c)}F(c)\epsilon' \in {}^{\sim}L_d$ we have that $|\epsilon F(c)p\epsilon' F(c)|_a$ and $|\epsilon F(c)p\epsilon' F(c)|_b$ are both odd numbers and so ${}^{\sim}\epsilon F(c)p\epsilon' F(c) \in {}^{\sim}L_d$. ■

Conclusions

In this thesis we have tried to provide a quite exhaustive overview and some new investigations on Molecular Computing and Splicing Systems. We have approached linear and circular (iterated) splicing systems in depth, and then restricted to the finite case. This has allowed us to compare splicing systems and automata and find a new model able to generate infinite (but regular) languages starting with finite languages. We have provided comparisons of different definitions of splicing and several examples of regular languages, generated or not. The reseach has succeded in defining new families of regular languages by means of automata or combinatorial properties and proving them to be splicing languages. Connections between splicing and codes have also been given. In the case of one-letter alphabet it has been proved to be decidable whether or not a regular language is generated by circular splicing. A circular regular language not generated by splicing has been presented, showing that, at least in the simplest formulation, circular splicing cannot be used to characterize circular regularity, which is still an open problem. The present study highlights that, up to now, no circular regular splicing language exist which is not contained in one of the classes we have defined.

In order to characterize regular languages generated by linear splicing, we have used syntactic monoid, so stating a sufficient condition for being linear splicing generated. We argue that this approach, we are already using, can result effective for bounding the length of splicing rules and to complete the characterization, which is a very difficult problem that we have been studying.

Bibliography

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

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

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

[4] J. Berstel, A. Restivo (1981), Codes et sousmonoides fermes par conjugaison, *Sem. LITP* **81-45**, 10 pages.

[5] Biomolecular Computing Group at Binghampton, http://www.math.binghamton.edu/dennis/DARPA/index.html

[6] P. Bonizzoni (1998), *private communication.*

[7] 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.

[8] P. Bonizzoni, R. Zizza (2000), A characterization of regular splicing languages, *Tech. Rep.* **254-00**, University of Milan, 16 pages.

[9] P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza (2001), Developments on circular splicing, presented at WORDS01, Palermo (Italy).

[10] P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza (2001), On the power of circular splicing, *in preparation.*

[11] 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.

[12] P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza (2001) Circular splicing and regularity, *submitted.*

[13] P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza (1999), Linear and circular splicing, presented at WORDS99, Rouen (France).

[14] P. Bonizzoni, C. Ferretti, G. Mauri, R. Zizza (2000), Separating some splicing models, presented at Grammar Systems 2000, Austria.

[15] P. Bonizzoni, C. Ferretti, G. Mauri, R. Zizza (2001), Separating some splicing models, *Information Processing Letters* **76 (6)**, pp. 255 – 259.

[16] 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.

[17] K. Culik, T. Harju (1991), Splicing semigroups of dominoes and DNA, *Discrete Applied Mathematics*, **31**, pp. 261 – 277.

[18] M. Daley, L. Kari, G. Gloor, R. Sirimoney (1999) Circular contextual insertions/deletions with applications to biomolecular computing, *Proc. of SPIRE'99, 6th Int. Symp. on String Processing and Information Retrieval*, Concun, Mexico, IEEE Comp. Science Press, Los Alamos, California, pp. 47 – 54.

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

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

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

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

[23] C. De Felice (1996), An application of Hajós factorizations to variable-length codes, *Theoretical Computer Science* **164**, pp. 223 – 252.

[24] 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.

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

[26] R. Freund, L. Kari, G. Paun (1995), DNA Computing based on splicing: the existence of universal computers, *Journal of the ACM*, to appear.

[27] P. Frisco, *A bibliography of molecular computation and splicing systems*, http://www.wi.LeidenUniv.nl/~pier/dna.html

[28] L. Fuchs (1960), *Abelian groups*, Pergamon Press, Oxford, London, New York and Paris.

[29] R. W. Gatterdam (1992), Algorithms for splicing systems, *SIAM Journal of Computing*, **21:3**, pp. 507 – 520.

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

[31] E. Goode, D. Pixton (2001), Semi-simple splicing systems, *in* "Where Mathematics, Computer Science, Linguistics and Biology Meet" (C. Martin-Vide & V. Mitrana, Eds.), pp. 343 – 357, Kluwer Academic Publ., Dordrecht.

[32] J. Hartmanis (1995), On the Weight of Computations, *Bull. of EATCS* **55**, pp. 136 – 138.

[33] T. Head (1992), Splicing schemes and DNA, *in* "Lindenmayer Systems: Impacts on Theoretical Computer Science and Developmental Biology", pp. 371 – 383, Springer-Verlag, Berlin.

[34] 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.

[35] T. Head (1998), Splicing languages generated with one sided context, *in* "Computing with Bio-molecules: Theory and Experiments" (Gh. Paun, Ed.), Springer-Verlag, Singapore.

[36] T. Head (1998), Splicing representations of strictly locally testable languages, *Discrete Applied Math.*, **87**, pp. 139 – 147.

[37] 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.

[38] T. Head, Gh. Paun, D. Pixton (1996), Language theory and molecular genetics: generative mechanisms suggested by DNA re-

combination, *in* "Handbook of Formal Languages" (G. Rozenberg & A. Salomaa, Eds.), Vol. 2, pp. 295 – 360, Springer-Verlag.

[39] 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.

[40] J.E. Hopcroft, J.D. Ullman (1979), *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Mass.

[41] S. Hussini, L.Kari, S. Koustantinidis (2001) Coding Properties of DNA languages, *Proc. of DNA 7.*

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

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

[44] L. Kari (1999), DNA computing: an arrival of biological mathematics, *The Mathematical Intelligencer* **19:2**, pp. 9 – 22.

[45] L. Kari, Gh. Paun, G. Thierren (1997), At the crossroads of DNA Computing and Formal Languages: Characterizing Recursive Enumerable Languages using insertion and deletion systems, *in* Proc. of 3rd DIMACS Int. Workshop in DNA Based Computers, pp. 318 – 333.

[46] S.M. Kim (1997), An Algorithm for Identyfying Spliced Languages, *Proc. of Cocoon97*, Lecture Notes in Computer Science **1276**, pp. 403 – 411.

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

[48] E. Laun, K. Reddy (1999), Wet Splicing Systems, *Proc. of 3rd Annual Meeting on DNA Based Computers*, (H. Rubin and D. Woods, Eds.) DIMACS Series in Discrete Mathematics and Theoretical Computer Science, **48**, AMS, Providence RI.

[49] Life Science BioTech Dictionary, http://biotech.icmb.utexas.edu /search/dict-search.html

[50] Cellular Biology, http://www.mblab.gla.ac.uk/ julian/Dict.html

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

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

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

[54] On-line Medical Dict., http://www.graylab.ac.uk/omd/index.html

[55] Gh. Paun (1996), On the splicing operation, *Discrete Applied Math.* **70**, pp. 57 – 79.

[56] Gh. Paun (1996), Regular extended H systems are computationally universal, *Journal of Aut., Lang., Combinatorics.* **1:1**, pp. 27 – 36.

[57] Gh. Paun, A. Salomaa (1996), DNA computing based on the splicing operation, *Math. Japonica*, **43:3**, pp. 607 – 632

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

[59] G. Paun, G. Rozenberg, A. Salomaa (1998), *DNA computing, New Computing Paradigms*, Springer-Verlag.

[60] D. Perrin (1995), Symbolics Dynamics and Finite Automata, *Proc. of MFCS95*, Lecture Notes in Computer Science **969**.

[61] D. Perrin, Finite Automata, *in* "Handbook of Theoretical Computer Science" (J. Van Leeuwen, Ed.), Vol. B, pp. 1 – 57, Elsevier.

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

[63] D. Perrin, M. P. Schützenberger (1976), Codes et sous-monoides possedant des mots neutres, *in* "Theoretical Computer Science", Lecture Notes in Computer Science **48**, pp. 270 – 281.

[64] D. Pixton (2000), Splicing in abstract families of languages, *Theoret. Comp. Science* **234**, pp. 135 – 166.

[65] D. Pixton (1996), Linear and Circular Splicing Systems, *in* "Proc. of 1st Int. Symp. on Int. in Neural and Biological Systems", pp. 181 – 188.

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

[67] C. Reis, G. Thierren (1979), Reflective star languages and codes, *Information and Control* **42**, pp. 1 – 9.

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

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

[70] R. Siromoney (2000), Distributed Circular Systems, presented at Grammar Systems 2000, Austria.

[71] 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.

[72] D. G. Thomas, K. G. Subramanian, V. R. Dare, T. Robinson, B. Le Saec (2000), Learning algorithms for subclasses of circular DNA molecules, Poster session in DNA 2000, Leiden 2000.

[73] P. Wilhelm, K. Rothemund(1996), A DNA restriction enzyme implementation of Turing Machines, *in* "DNA Based Computers", Vol. 27 of *DIMACS* Series in Discrete Mathematics and Theoretical Computer Science, AMS.

[74] T. Yokomory, S. Kobayaski, C. Ferretti (1999), On the power of circular splicing systems and DNA computability, *Proc. of IEEE Int. Conf. Evol. Comp.* ICEC99.

[75] R. Zizza (1999), On the power of classes of splicing systems, *Thesis Proposal.*

Appendix: A flash of biochemical background

For the sake of completeness, we collect here a bit of basically definitions on biochemistry, only with respect to what we have used in the thesis. The material of this section is taken from [54, 49, 50, 19].

- **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).

- **Circular DNA.** DNA arranged as a closed circle. Characteristic of prokaryotes, but also found in mitochondria and some viral genomes.

- **Gene.** Originally defined as the physical unit of heredity, it is probably best defined as the unit of the inheritance that occupies a specific locus on a chromosome (self-replicating genetic structures of cells containing the cellular DNA that bears in its nucleotide sequence the linear array of genome). They are formed from DNA.

- **DNA replication** The process whereby a copy of a DNA molecule is made and thus the genetic informations it contains is duplicated. The parental double stranded DNA molecule is replicated semi-conservatively, i.e., each copy contains one of the original strands paired with a newly sybthesized strand that is complementary in terms of AT and GC base pairing. Though in this sense conceptually simple, mechanistically a complex process involving a number of enzymes.

- **Polymerase Chain Reaction (PCR).** The first pratical system for *in vitro* amplification of DNA, and as such one of the most important recent developments in molecular biology. 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). Because the copies of templates can function as templates themselves, repeated PCR can be used to exponentially multiply the template. It is realized by a series of temperature 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.

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

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

- **Recombinant DNA.** Spliced DNA formed from two or more different sources that have been cleaved by restriction enzymes and joined by ligases.

- **Plasmids.** Autonomously replicating circular DNA molecules, distinct from the normal bacterial genome and nonessential for cell survival under nonselective conditions. Some plasmids are capable of integrating into the host genome. A number of artificially constructed plasmids are used as cloning vectors.

- **Restriction Fragment length polymorphism (RFLP)**. Tecnique, also known as DNA fingerprinting, that allows familiar relationship to be estabilished by comparing the characteristic polymorphic patterns that are obtained when certain regions of genomic DNA are amplified (typically by PCR) and cut with certain restriction enzymes. In principle, an individual can be identified unambiguously by RFLP. Similarly if a polymorphism can be identified close to the locus of a genetic defect, it provides a valuable marker for tracing the inheritance of the defect.

Splicing in cartoons

Be careful! The last pages of my thesis are devoted to a nice side effect I have received working on splicing.

I have presented the results obtained in my PhD Thesis step by step, in some exciting conferences, like Words99, DNA6, Words01, and some meetings regarding Italian Murst Project. As youngest in the working group, I have prepared the talks and, trying to explain splicing operation to a not specialized audience, I have used to give an informal description by depicting some "cartoons". I am very happy to include some of them in my PhD Thesis, as a reminder of those marvellous days. Someone told me that, thanks to them, I was able to convince not theoricians that splicing theory and formal language theory may be amusing and coloured!