Theoretical Computer Science 436 (2012) 2–22

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Splicing systems and the Chomsky hierarchy☆

Jean Berstel [a], Luc Boasson [b], Isabelle Fagnot [a,c,*]

[a] LIGM, Université Paris-Est Marne-la-Vallée, 5, boulevard Descartes, Champs-sur-Marne, F-77454 Marne-la-Vallée Cedex 2, France

[b] LIAFA, Université Paris Diderot–Paris 7 and CNRS, Case 7014, 75205 Paris Cedex 13, France

[c] Université Paris Diderot–Paris 7, Case 7014, 75205 Paris Cedex 13, France

ARTICLE INFO

Article history:
Received 4 February 2011
Received in revised form 16 January 2012
Accepted 7 March 2012
Communicated by N. Jonoska

Keywords:
Formal languages
Splicing system
Chomsky hierarchy

ABSTRACT

In this paper, we prove decidability properties and new results on the position of the family of languages generated by (circular) splicing systems within the Chomsky hierarchy. The two main results of the paper are the following. First, we show that it is decidable, given a circular splicing language and a regular language, whether they are equal. Second, we prove the language generated by an alphabetic splicing system is context-free. Alphabetic splicing systems are a generalization of simple and semi-simple splicing systems already considered in the literature.

1. Introduction

Splicing systems were introduced by Head [12–14] as a model of recombination. The basic operation is to cut words into pieces and to reassemble the pieces in order to get another word.

There are several variants of splicing systems for circular or linear words [14]. In this paper, we consider Păun's circular splicing, and we introduce a new variant that we call flat splicing. In both cases, the system is described by an *initial set* of words and a *finite set of rules*. The language generated is the closure of the initial set under the application of splicing rules.

A splicing rule is a quadruplet of words, usually written as $\alpha\#\beta\$\delta\#\gamma$. The words α, β, γ, δ are called the *handles* of the rule. A rule indicates where to cut and what to paste. More precisely, in a circular splicing system, given a rule $\alpha\#\beta\$\delta\#\gamma$ and two circular words, the first of the form $u\alpha\cdot\beta v$ and the second of the form $\gamma w\delta$, we cut the first word between α and β, the second word between δ and γ and stick α with γ as well as δ with β in order to get the new circular word $u\alpha\cdot\gamma w\delta\cdot\beta v$, see Fig. 1. The case of flat splicing systems, which involves linear words, is similar, see Fig. 2. In order to emphasize the position where to cut and the condition on what to paste, we prefer to write $\langle\alpha|\gamma - \delta|\beta\rangle$ instead of $\alpha\#\beta\$\delta\#\gamma$. This indicates more clearly that one word is cut between α and β, and that the word to be pasted is in $\gamma A^*\delta$.

Our purpose, in introducing flat splicing systems, is to get a direct approach to standard results in formal language theory. Circular systems are handled, in a second step, by full linearization.

Our definition of flat splicing system differs from various so-called linear splicing systems appearing in the literature [14,17,18,20]. For example, in Păun linear splicing, the rule $\alpha|\beta\$\gamma|\delta$, from two words $x\alpha\beta y$ and $z\gamma\delta t$, produces the two words $x\alpha\delta t$ and $z\gamma|\beta y$. The other linear systems (Head's and Pixton's) behave roughly on the same way.

Pixton has considered the nature of the language generated by a splicing system, with some assumptions about the splicing rules (symmetry, reflexivity and self-splicing). He proves that the language generated by a splicing system is regular

☆ Part of this work has been done during a sabbatical leave of the third author from University Paris 7, and during a stay at DIA (Dipartimento di Informatica e Applicazioni) at Università di Salerno, Italy.

* Corresponding author at: LIGM, Université Paris-Est Marne-la-Vallée, 5, boulevard Descartes, Champs-sur-Marne, F-77454 Marne-la-Vallée Cedex 2, France. Tel.: +33 1 60 95 75 50.

E-mail addresses: Isabelle.Fagnot@univ-mlv.fr, fagnot@univ-mlv.fr (I. Fagnot).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.03.008

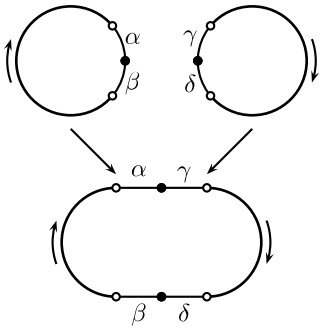

Fig. 1. Circular splicing.

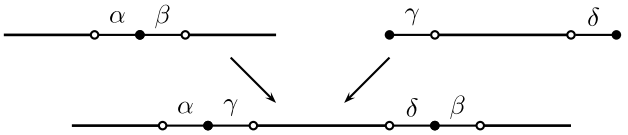

Fig. 2. Flat splicing.

(resp. context-free), provided the initial set is regular (resp. context-free). More generally, if the initial set is in some full AFL, then the language generated by the system is also in this full AFL. Without the additional assumptions on the rules, it is known that one may generate non-regular languages even with a finite initial set (Siromoney et al. [19]). A survey of recent developments along these lines appears in [2].

In this paper, we prove decidability properties and new results on the position of splicing systems and their languages within the Chomsky hierarchy. We introduce a special class of splicing rules called alphabetic rules. A rule is *alphabetic* if its four handles are letters or the empty word. A splicing system is alphabetic when all its rules are alphabetic. Special cases of alphabetic splicing systems, called *simple* or *semi-simple* systems, have been considered in the literature [8,7,5]. In a semi-simple system, all rules $\alpha\#\beta\$\delta\#\gamma$ satisfy the condition that the words $\alpha\beta$ and $\gamma\delta$ are letters. In a simple system, one requires in addition that these letters are equal, that is $\alpha\beta = \gamma\delta$.

We show that alphabetic systems have several remarkable properties that do not hold for general systems.

We consider first the problem of deciding whether the language generated by a splicing system is regular. The problem is still open, but has been solved in special cases [6,9] . Our contribution is the following (Theorem 3.1). It is decidable, given a splicing language and a regular language, whether they are equal. The corresponding inclusion problems are still open. We also show (Remark 3.5) that it is decidable whether a given regular language is an alphabetic splicing language. This is related to another problem that we do not consider here, namely to give a characterization of those regular languages that are splicing languages, or vice versa. For recent results see [6], and for a survey see [2,20].

The next problem we consider concerns the comparison of the family of splicing languages with the Chomsky hierarchy. We first prove (Theorem 4.1) that splicing languages are always context-sensitive. Next, we prove, and this is the main result of the paper (Theorem 5.3), that alphabetic splicing languages are context-free. The proof of this result needs several intermediate results and fills a good deal of the article.

The relation between circular and flat alphabetic splicing systems is described in Proposition 8.3. It follows easily that the main result also holds for circular splicing (Theorem 8.4).

The paper is organized as follows. We start by introducing the new type of splicing systems called flat splicing systems: these systems behave like circular systems, but operate on linear words.

In Section 3, we prove that it is decidable, given a splicing language and a regular language, whether they are equal (Theorem 3.1). Section 4 contains the proof that splicing languages are always context-sensitive.

Section 5 defines *alphabetic* splicing systems and states the main result (Theorem 5.3) namely that the language generated by a flat or circular alphabetic splicing system is context-free, even if the initial set is context-free. This section and the next two contains the proof of the main theorem for flat splicing systems (Theorem 7.11).

Section 8 describes the relationship between flat and circular splicing systems and their languages. It contains the proof of the main theorem for circular splicing systems (Theorem 8.4).

All proofs are effective.

The proofs of the two main theorems use an old result from formal language theory [15] that presents *generalized context-free grammars* and shows that they generate context-free languages. As it seems not to be well known, for sake of completeness, we include a proof of this result, together with an example, in an Appendix.

The results of the paper were announced by the third author in [10].

2. Definitions

2.1. Words, circular words

As usual, an *alphabet* A is a finite set of *letters*. A word $u = u_0u_1 \dots u_{n-1}$ is a finite sequence of letters. When it is useful to compare words with circular words defined below, they will be called *linear* words.

Two words u and v are conjugate, denoted by $u \sim v$ if there exist two words x and y such that $u = xy$ and $v = yx$. This is an equivalence relation. A *circular word* is an equivalence class of $\sim$, that is an element of the quotient of A^* by the relation $\sim$. The equivalence class of u will be noted $\tilde{}u$. We also say that $\tilde{}u$ is the *circularization* of u. For example, $\tilde{}abb = \{abb, bab, bba\}$ is a circular word. A circular word can also be viewed as in Fig. 1 as a word written on a circle. A set of circular words is a *circular language*.

Let C be a language of circular words. Its *full linearization*, denoted by $Lin(C)$, is the language $Lin(C) = \{u \in A^* \mid \tilde{}u \in C\}$.

Let L be a language of linear words. Its *circularization* $\tilde{}L$ is equal to $\tilde{}L = \{\tilde{}u \mid u \in L\}$.

A language L of circular words is *regular* (resp. *context-free*, resp. *context-sensitive*) if its full linearization is regular (resp. context-free, resp. context-sensitive).

Let G be a context-free grammar, the language generated by G will be denoted by L_G. Let S be a non-terminal symbol, we will denote $L_G(S)$ the language produced by the grammar G with S as axiom.

2.2. Splicing systems

We begin by presenting flat splicing systems which are new systems. Then we make a short description of circular splicing systems. These systems are well known, see e.g. [6]. We also give some common properties of the two types of systems as well as stressing the difference between both.

Flat splicing systems are of interest for proving language-theoretic results because they allow us to separate operations on formal languages and grammars from the operation of circular closure (circularization). It appears that proofs for linear words are sometimes simpler because they rely directly on standard background on formal languages.

2.2.1. Flat splicing systems

A *flat splicing system*, or a *splicing system* for short, is a triplet $\mathcal{S} = (A, I, R)$, where A is an alphabet, I is a set of words over A, called the *initial set* and R is a finite set of *splicing rules*, which are quadruplets $\langle \alpha|\gamma - \delta|\beta\rangle$ of words over A. The words α, β, γ and δ are called the *handles* of the rule.

Let $r = \langle \alpha|\gamma - \delta|\beta\rangle$ be a splicing rule. Given two words $u = x\alpha \cdot \beta y$ and $v = \gamma z\delta$, applying r to the pair (u, v) yields the word $w = x\alpha \cdot \gamma z\delta \cdot \beta y$. (The dots are used only to mark the places of cutting and pasting, they are not parts of the words.) This operation is denoted by $u, v \vdash_r w$ and is called a *production*. Note that the first word (here u) is always the one in which the second word (here v) is inserted.

Example 2.1. 1. Consider the splicing rule $r = \langle ab|aa - b|c\rangle$. We have the production $bab \cdot cc, aaccb \vdash_r bab \cdot aaccb \cdot cc$.

2. Consider the splicing rule $\langle b|a - a|b\rangle$. Note that we cannot produce the word $b \cdot a \cdot b$ from the word $b \cdot b$ and the singleton a, because the rule requires that the inserted word has at least two letters. On the contrary, the rule $\langle b|\varepsilon - a|b\rangle$ does produce the word bab from the words bb and a.

3. For the rule $r = \langle \varepsilon|a - a|b\rangle$, the production $\cdot bbc, aba \vdash_r aba \cdot bbc$, is in fact a concatenation.

4. As a final example, the rule $\langle \varepsilon|\varepsilon - \varepsilon|\varepsilon\rangle$ permits all insertions (including concatenations) of a word into another one.

The *language generated* by the flat splicing system $\mathcal{S} = (A, I, R)$, denoted $\mathcal{F}(\mathcal{S})$, is the smallest language L containing I and closed by R, i.e., such that for any couple of words u and v in L and any rule r in R, then any word such that $u, v \vdash_r w$ is also in L.

Example 2.2. Consider the splicing system over $A = \{a, b\}$ with initial set $I = \{ab\}$ and the unique splicing rule $r = \langle a|a - b|b\rangle$. It generates the context-free and non-regular language $\mathcal{F}(\mathcal{S}) = \{a^n b^n \mid n \geq 1\}$. (This example is a straightforward adaptation of Example 2.3 below.)

A splicing system is *finite* (resp. *regular, context-free, context-sensitive*) if its initial set is finite (resp. regular, context-free, context-sensitive).

A rule $r = \langle \alpha|\gamma - \delta|\beta\rangle$ is *alphabetic* if its four handles α, β, γ and δ are letters or the empty word. A splicing system is alphabetic if all its rules are alphabetic.

2.2.2. Splicing and insertion systems are incomparable

Insertion systems, as described in [16], look quite similar to flat splicing systems. This justifies to compare both generating devices.

An *insertion system* is given by a finite set R of rules of the form $(g|\alpha|d)$, where g, α, d are words, and a finite set I of words called the initial set. The system generates a language as follows : given a word w that contains a factor gd, that is of the form $w = xgdy$, then the application of the rule $(g|\alpha|d)$ generates the word $w' = xg\alpha dy$. The set of words generated by the system is the set of words obtained by repeated application of the insertion rules, starting with the words in the initial set I.

The two main differences between splicing systems and insertion systems are following :

- in an insertion system, only a finite number of words can be inserted (the words α in the rules).
- The words α which are inserted do not necessarily belong to the language the system generates.

In [16], it is proved (Theorem 6.4) that *alphabetic* insertion systems, that is insertion systems where the contexts g, d of the rules have length at most 1, always generate context-free languages. This is very close to our result (Theorem 5.3) stating that alphabetic splicing systems generate context-free languages.

However, despite the fact that the families of languages generated by the two types of systems have a non-empty intersection, we show below that the two types of systems generate incomparable families of languages, and so the two results quoted are really different.

2.2.2.1. A splicing system generating a language which is not generated by an insertion system.
Let A be the alphabet $\{a, b, c, d, x, y\}$ and define the following splicing system:

$(r1)$ $\langle c|x-y|\varepsilon\rangle$
$(r2)$ $\langle \varepsilon|c-y|d\rangle$
$(r3)$ $\langle a|c-d|\varepsilon\rangle$
$(r4)$ $\langle \varepsilon|a-d|b\rangle$
$(r5)$ $\langle c|a-b|\varepsilon\rangle$
$(r6)$ $\langle \varepsilon|c-b|d\rangle$.

Let P be the language generated by this system, with initial set $I = \{a, b, c, d, xy\}$. We prove that P is not generated by any insertion system.

For this, we start with some easy preliminary remarks.

First, only words of length at least two can be inserted in another word; this is because the rules contain no ε in their central part. It follows immediately that all words obtained by applying at least one rule contain the factor xy, and consequently all words of P of length at least 2 contain the factor xy. Along the same lines, if a word u is inserted in a word v and if v has length at least 2, the result will contain at least two letters x (and two letters y). Hence, words generated in P containing at most one letter x, are obtained by insertions of words u (of length at least 2) in a word of length at most 1. So, looking for these words, and starting with $u = xy$, one can only produce cxy by using rule $(r1)$. Then, by rule $(r2)$, one gets the word $cxyd$. Then one has to use successively the rules $(r3)$, $(r4)$, $(r5)$, $(r6)$ and repeat them in this exact sequence. This leads to the following conclusion:

The set of words generated by the splicing system and containing exactly one occurrence of the letter x is

$$L = M \cup cM \cup cMd \cup acMd \quad \text{where } M = \{(ac)^n xy(db)^n \mid n \geq 0\}.$$

Now assume, arguing by contradiction, that the language P is generated by some insertion system. Then, the words $w_n = (ac)^n xy(db)^n$ must be generated by this system. For a large enough n, such a word w_n cannot be in the initial set of the insertion system. Hence this word factorizes into $w_n = ur\alpha sv$ in such a way that $ursv$ is in P and the insertion system contains a rule $(r|\alpha|s)$. This ensures that w_n is obtained by inserting the word α in the shorter word $ursv$ of the language. The only possibility is to cut the word $(ac)^n xy(db)^n$ into $ursv = (ac)^n xy(db)^{n-1}d$ and $\alpha = b$. This implies that the considered insertion system contains a rule of the form$(r|b|s)$ with $s = \varepsilon$ and r a suffix of $(ac)^n xy(db)^{n-1}d$; however, with a rule of this kind, one generates also words like $(ac)^{m+n}xy(db)^{n-1}dbb(db)^m$ by inserting a letter b at the right of the $(m)^{th}$ block db. As this word is not in P, we get a contradiction.

2.2.2.2. An insertion system generating a language which is not generated by a splicing system.
Over $A = \{a, x, y\}$, consider the (rational) language $K = xa^*y$. It is generated by the insertion system with initial set $F = \{xy\}$ and the insertion rules

$(r1)$ $(x|a|y)$
$(r2)$ $(x|a|a)$.

On the other hand, all words of K contain exactly one letter x. Since the insertion of a word of K in another one would lead to a word containing at least two letters x, this set is not obtained by a finite splicing system.

2.2.3. Circular splicing systems

A circular splicing system is a triplet $\mathcal{S} = (A, I, R)$, where A is an alphabet, I is a set of circular words on A, called *initial set* and R is a finite set of *splicing rules*, which are quadruplets $\langle \alpha|\gamma-\delta|\beta\rangle$ of *linear* words on A. The words α, β, γ and δ are called the *handles* of the rule. In the literature (see e.g. [6]), a rule is written as $\alpha\#\beta\$\delta\#\gamma$. Again, a rule is *alphabetic* if its four handles α, β, γ and δ are letters or the empty word. A circular splicing system is alphabetic if all its rules are alphabetic.

If $r = \langle \alpha|\gamma-\delta|\beta\rangle$ is a splicing rule then the circular words $\tilde{}u = \tilde{}(\beta x\alpha)$ and $\tilde{}v = \tilde{}(\gamma y\delta)$ produce the circular word $\tilde{}w = \tilde{}(\beta x\alpha \cdot \gamma y\delta)$. We will denote this production by $\tilde{}u, \tilde{}v \vdash_r \tilde{}w$. The *language generated* by the circular splicing

system is the smallest language C of circular words containing I and closed by R, i.e., such that for any couple of words $\tilde{}u$ and $\tilde{}v$ in C and any rule r in R, any circular word $\tilde{}w$ such that $\tilde{}u, \tilde{}v \vdash_r \tilde{}w$ is also in C. This set of circular words is denoted by $\mathcal{C}(\mathcal{S})$.

The notion of *finite, regular, context-free, context-sensitive* circular splicing system are alike to those in flat splicing systems.

Example 2.3 (*[19]*)**.** Let $\mathcal{S} = (A, I, R)$ be the (finite alphabetic) circular splicing system defined by $I = \{\tilde{}(ab)\}$ and $R = \{\langle a|a{-}b|b\rangle\}$. It produces the context-free language $\mathcal{C}(\mathcal{S}) = \{\tilde{}(a^n b^n) \mid n \geq 1\}$.

2.2.4. Relationship between flat and circular splicing systems

The following two already known remarks hold for flat and circular splicing systems, and will be used to simplify our proofs [20].

Remark 2.4. A production $u, v \vdash_r w$, where $u = \varepsilon$ or $v = \varepsilon$, even when it is permitted, is useless. Indeed, one has $\varepsilon, v \vdash_r v$ and $u, \varepsilon \vdash_r u$. As a consequence, given a splicing system $\mathcal{S} = (A, I, R)$ one has $\varepsilon \in \mathcal{F}(\mathcal{S})$ (resp. $\varepsilon \in \mathcal{C}(\mathcal{S})$) if and only if $\varepsilon \in I$. So we can assume that $\varepsilon \notin I$ without loss of generality.

Remark 2.5. A production $u, v \vdash_r w$, where $|w| = 1$, even when it is permitted, is useless. Indeed, since $|u| + |v| = |w|$, one has in this case $w = u$ or $w = v$. As a consequence, given a splicing system $\mathcal{S} = (A, I, R)$, and a letter $a \in A$, one has $a \in \mathcal{F}(\mathcal{S})$ (resp. $a \in \mathcal{C}(\mathcal{S})$) if and only $a \in I$. However, we cannot assume that $a \notin I$ without possibly changing the language it generates.

Remark 2.6. Let $\mathcal{S} = (A, I, R)$ be a circular splicing system. The full linearization of $\mathcal{C}(\mathcal{S})$ is the closure of the linear language $Lin(I)$ under the composition of the two operations of splicing and circularization.

Remark 2.7. Despite the previous remark, it does not suffice, in general, to just consider a single circularization. If we consider the flat splicing system $\mathcal{S}' = (A, Lin(I), R)$ with the same splicing rules, we have not $\mathcal{C}(\mathcal{S}) = \tilde{}\mathcal{F}(\mathcal{S}')$. Nevertheless, the inclusion $\tilde{}\mathcal{F}(\mathcal{S}') \subseteq \mathcal{C}(\mathcal{S})$ is always true.

For example, consider the circular splicing system $\mathcal{S} = (A, I, R)$ with $A = \{a, b\}$, $I = ab$ and $R = \{\langle a|a{-}\varepsilon|b\rangle\}$ and the flat splicing system $\mathcal{S}' = (A, I' = Lin(I), R)$. As $I' = \{ab, ba\}$, the flat splicing system $\mathcal{S}'$ will produce only the language $L = \{ba\} \cup \{a^n b^n \mid n \geq 1\}$ while the circular splicing system $\mathcal{S}$ will, for example, produce also the word *aabbab* by the two productions $ab, ab \vdash_r aabb = abba$, and $ab, abba \vdash_r aabbab$. This word *aabbab* is not in the circularization of the language L.

3. A decision problem

In this section, we prove the following result.

Theorem 3.1. *Given a regular circular (resp. flat) splicing system $\mathcal{S}$ and a regular language K, it is decidable whether $\mathcal{C}(\mathcal{S}) = K$ (resp. $\mathcal{F}(\mathcal{S}) = K$).*

Proof. We assume that neither I nor K contains ε, since otherwise it suffices, according to Remark 2.4, to check that ε is contained in both sets.

Let $\mathcal{S} = (A, I, R)$. Let $\mathcal{A} = (A, Q, q_o, Q_F)$ be a deterministic automaton recognizing K, with Q the set of states, q_o the initial state and Q_F the set of final states. The transition function is denoted by "$\cdot$" in the following way: for a state q and a word v, $q \cdot v$ denotes the state that is reached by v from q.

For any state $q \in Q$, we define $G_q = \{v \mid q_o \cdot v = q\}$ and $D_q = \{v \mid q \cdot v \in Q_F\}$. The set G_q is the set of all words which label paths from q_0 to q, and D_q is the set of all words which label paths from q to a terminal state. Both sets are regular.

Next, let $P = \{w \in A^* \mid u, v \vdash_r w, \text{ with } r \in R, u, v \in K\}$. The set P is the set of the words that can be obtained by splicing two words of K.

For each rule $r = \langle \alpha|\gamma{-}\delta|\beta\rangle$, let

$$\begin{aligned} K_r &= \{w \in A^* \mid u, v \vdash_r w, \text{ with } u, v \in K\} \\ &= \{x\alpha \cdot \gamma z\delta \cdot \beta y \mid x\alpha \cdot \beta y \in K, \gamma z\delta \in K, x, y, z \in A^*\}. \end{aligned}$$

It is easily checked that

$$K_r = \bigcup_{q \in Q} (G_q \cap A^*\alpha)(K \cap \gamma A^*\delta)(D_q \cap \beta A^*).$$

This expression shows that each language K_r is regular, and so is $P = \bigcup_{r \in R} K_r$ because R is finite.

We first consider flat splicing system. The algorithm consists in checking three inclusions. We claim that $\mathcal{F}(\mathcal{S}) = K$ if and only if the following three inclusions hold.

(1) $I \subseteq K$,
(2) $P \subseteq K$,
(3) $K \setminus P \subseteq I$.

Take the claim for granted. Then the equality $\mathcal{F}(\mathcal{S}) = K$ is decidable since the three inclusions, that involve only regular languages, are decidable.

Now we prove the claim, namely that $\mathcal{F}(\mathcal{S}) = K$ if and only if the above-mentioned three inclusions hold.

If $\mathcal{F}(\mathcal{S}) = K$, then (1), (2) and (3) are obviously true.

Conversely, assume now that these three inclusions hold. Since $I \subseteq K$ by (1) and since K is closed under the rules of splicing of R by (2), obviously $\mathcal{F}(\mathcal{S}) \subseteq K$.

Next, we prove the reverse inclusion $K \subseteq \mathcal{F}(\mathcal{S})$ by induction on the length of the words in K. Let $w \in K$. Since $P \subset K$ by (2), one has $K = P \cup (K \setminus P)$. If $w \in K \setminus P$, then by (3), $w \in I$ and therefore $w \in \mathcal{F}(\mathcal{S})$. Otherwise, there are words $u, v \in K$ of shorter length such that $u, v \vdash_r w$ for some $r \in R$. By induction, $u, v \in \mathcal{F}(\mathcal{S})$ and consequently $w \in \mathcal{F}(\mathcal{S})$.

This proves the result for flat splicing.

For circular splicing systems, we assume that the language $\tilde{}I$ and $\tilde{}K$ are given by a linearization of the circular language, that, by commodity, we call I and K. Recall that the full linearization of $\mathcal{C}(\mathcal{S})$ is the closure of the linear language $Lin(\tilde{}I)$ under the composition of the two operations of splicing and circularization (Remark 2.6.)

It then suffices to check, in addition, that K and I are closed under conjugacy (or to compute their closure) and to replace K_r by $Lin(\tilde{}K_r)$, in order to obtain a set P that is also closed under conjugacy. The circular closure is regular since with the previous notation we have $Lin(\tilde{}K) = \bigcup_{q\in Q} D_q \cdot G_q$. By this, we ensure that $P = Lin(\{w \in A^* \mid u, v \vdash_r w, \text{ with } r \in R, u, v \in \tilde{}K\})$. The rest of the proof being unchanged. □

Remark 3.2. There are two related problems which are still open. The first is to decide whether the language generated by a splicing system is regular, and the second is to decide whether a regular language can be generated by a splicing system. We shall see below that the second problem is decidable in the case of what we call alphabetic splicing systems.

Remark 3.3. The two inclusion problems, for flat and circular system, i.e., the problem of deciding whether $\mathcal{F}(\mathcal{S}) \subseteq K$ or whether $K \subseteq \mathcal{F}(\mathcal{S})$ (resp. $\mathcal{C}(\mathcal{S}) \subseteq K$ or $K \subseteq \mathcal{C}(\mathcal{S})$) are still open.

Remark 3.4. The characterization of the family of regular languages which can be obtained by a circular splicing system, is still open. However, partial results have been obtained by Bonizzoni et al. [3,4,6]. In particular, a complete characterization of languages over one letter generated by a splicing system is given in [4]. For another class of splicing systems, called marked system, the regularity of the languages generated has been proved decidable [9]. Recently, a description of the languages generated by a family of alphabetic splicing systems called semi-simple systems has been given in [6], see also [20].

Remark 3.5. Given a regular language K over an alphabet A, it is decidable whether it can be generated by a finite alphabetic splicing system. (The problem is meaningless for regular systems.) First, remark that given a set of rules R, if there exists a finite initial set I such that $\mathcal{C}(A, I, R) = K$, we have also $\mathcal{C}(A, K \setminus P, R) = K$ with $K \setminus P$ finite, P being defined as in the above-mentioned proof. It suffices to remark that $K \setminus P \subseteq I$ (inclusion (3) of the proof) ensures that $K \setminus P$ is finite and to check that the three inclusions still hold when I is replaced by $K \setminus P$.

Now, observe that there are only finitely many alphabetic splicing rules over A. So there are only finitely many sets of such rules over A. We then only need to enumerate these sets and for each set R of alphabetic splicing rules over A, define P as in the proof of Theorem 3.1; if $P \subseteq K$ and $K \setminus P$ is finite, then the test is positive and the splicing system $\mathcal{S} = (A, K \setminus P, R)$ generates K. If there is no such set R, then K cannot be generated by a finite alphabetic splicing system.

4. Splicing languages are context-sensitive

We will see that the highest level in Chomsky hierarchy which can be obtained by splicing systems with a finite initial set and a finite set of rules is the context-sensitive level. This result remains true when the initial set is context-sensitive.

Before proving this property, we give an example of a splicing language which is not context-free.

4.1. A splicing language which is not context-free

In this paragraph, an example of a flat linear splicing system having a finite initial set which produces a non-context-free example is presented. This example is then easily adapted to give a circular counter-part.

We first consider flat splicing, then we will consider the circular one.

Let A be the alphabet $\{0, 1, 2, 3, \blacktriangleright, \blacktriangleleft\}$ and set $u = 0123$. Let $\mathcal{S} = (A, I, R)$ be the flat splicing system with

$$I = \{\blacktriangleright u \blacktriangleleft, 0, 1, 2, 3\}$$

and with R composed of the rules

$$\begin{aligned} a_1 &= \langle \blacktriangleright | 0 - \varepsilon | u \rangle, & a_2 &= \langle 0u | 0 - \varepsilon | u \rangle, \\ b_1 &= \langle 0 | 1 - \varepsilon | u \blacktriangleleft \rangle, & b_2 &= \langle 0 | 1 - \varepsilon | u01u \rangle, \\ c_1 &= \langle \blacktriangleright 01 | 2 - \varepsilon | u \rangle, & c_2 &= \langle 012u01 | 2 - \varepsilon | u \rangle, \\ d_1 &= \langle 012 | 3 - \varepsilon | u \blacktriangleleft \rangle, & d_2 &= \langle 012 | 3 - \varepsilon | uuu \rangle. \end{aligned}$$

This splicing system produces the language

$$\begin{aligned}\mathcal{F}(\mathcal{S}) &= I \cup \{\blacktriangleright(u)^{2^n}\blacktriangleleft \mid n \geq 0\}\\ &\cup \{\blacktriangleright(0u)^p(u)^q\blacktriangleleft \mid p+q=2^n, n\geq 0\}\\ &\cup \{\blacktriangleright(0u)^p(01u)^q\blacktriangleleft \mid p+q=2^n, n\geq 0\}\\ &\cup \{\blacktriangleright(012u)^p(01u)^q\blacktriangleleft \mid p+q=2^n, n\geq 0\}\\ &\cup \{\blacktriangleright(012u)^p(uu)^q\blacktriangleleft \mid p+q=2^n, n\geq 0\}. \end{aligned} \tag{4.1}$$

This splicing system will produce the word $\blacktriangleright u^{2^{(n+1)}}\blacktriangleleft$ from the word $\blacktriangleright u^{2^n}\blacktriangleleft$ by roughly adding a word u before each of its occurrences. This is done by inserting first the letters '0' from left to right, then the letter '1' from right to left, then the letter '2' from left to right, and finally the letters '3' from right to left.

More precisely, the system generates the word $\blacktriangleright uu\blacktriangleleft$ from the word $\blacktriangleright u\blacktriangleleft$ by the following computation: (We write here $x \xrightarrow{a_1,0} y$ instead of $x, 0 \vdash_{a_1} y$, and so on.)

$$\blacktriangleright u\blacktriangleleft \xrightarrow{a_1,0} \blacktriangleright 0u\blacktriangleleft \xrightarrow{b_1,1} \blacktriangleright 01u\blacktriangleleft \xrightarrow{c_1,2} \blacktriangleright 012u\blacktriangleleft \xrightarrow{d_1,3} \blacktriangleright 0123u\blacktriangleleft.$$

Then, given a word $\blacktriangleright u^p\blacktriangleleft$, with $p = 2^n$, $n \geq 1$, the rules a_1 and a_2 generate a left-to-right sweep inserting the symbol 0 in head of each u:

$$\blacktriangleright u^p\blacktriangleleft \xrightarrow{a_1,0} \blacktriangleright (0u)u^{p-1}\blacktriangleleft \xrightarrow{a_2,0} \cdots \xrightarrow{a_2,0} \blacktriangleright (0u)^{p-1}u\blacktriangleleft \xrightarrow{a_2,0} \blacktriangleright (0u)^p\blacktriangleleft.$$

The rules b_1 and b_2 generate a right-to-left sweep which inserts a symbol 1 in head of each u. This gives

$$\blacktriangleright (0u)^p\blacktriangleleft \xrightarrow{b_1,1} \blacktriangleright (01u)(0u)^{p-1}\blacktriangleleft \xrightarrow{b_2,1} \cdots \xrightarrow{b_2,1} \blacktriangleright (01u)^{p-1}0u\blacktriangleleft \xrightarrow{b_2,1} \blacktriangleright (01u)^p\blacktriangleleft.$$

Similarly, the rules c_1 and c_2 are used to insert a symbol 2 in head of each u, again in a left-to-right sweep. This gives the word $\blacktriangleright (012u)^n\blacktriangleleft$. Finally, the last two rules, d_1 and d_2, insert a 3 in head of each u. The final result is $\blacktriangleright u^{2p}\blacktriangleleft = \blacktriangleright u^{2^{n+1}}\blacktriangleleft$.

We can check that no other word other that those produced in this computation can be yielded by noting that at each step no other rule can be applied.

The intersection of the language $\mathcal{F}(\mathcal{S})$ with the regular language $\blacktriangleright (u)^*\blacktriangleleft$ is equal to $\{\blacktriangleright (u)^{2^n}\blacktriangleleft \mid n \geq 0\}$. The latter language is not context-free.

Concerning circular splicing systems, recall that a circular language is context-sensitive if and only if its full linearization is context-sensitive. If we take the circular splicing system with the same rules and the same initial language, we can check that the language $\mathcal{C}(\mathcal{S})$ is such that $\mathcal{C}(\mathcal{S}) \cap \blacktriangleright (u)^*\blacktriangleleft = \mathcal{F}(\mathcal{S}) \cap \blacktriangleright (u)^*\blacktriangleleft$. (This can be done by only computing the words with at most one '$\blacktriangleleft$' and one '$\blacktriangleright$', the set of these words is the circularization of $\mathcal{F}(\mathcal{S})$.) Thus, we also can produce a language which is not context-free with a circular splicing system with finite initial set.

4.2. Splicing languages are always context-sensitive

Theorem 4.1. *The language generated by a context-sensitive circular (resp. flat) splicing system is context-sensitive.*

The proof uses bounded automata. Recall that a *k-linear bounded automaton* (k-LBA) is a non-deterministic Turing machine with a tape of only kn cells, where n is the size of the input. We will use in the sequel the following characterization of context-sensitive languages. A language is context-sensitive if and only if it is recognized by a k-LBA (see, for example, [11]). It is known that it is always possible to recognize a context-sensitive language with a 1-LBA.

Proof. We start with the case of a flat system. Let $\mathcal{S} = (A, I, R)$ be a flat splicing system. Let $\mathcal{T}$ a 1-LBA recognizing I. We construct a 3-LBA machine $\mathcal{U}$ which recognizes the language $\mathcal{F}(\mathcal{S})$.

Let u be the word written on the tape at the beginning of the computation. Let # be a new symbol. The machine works as follows.

During the computation the word written on the tape will have the form

$$u_1\#u_2\#\cdots\#u_{n-1}\#u_n,$$

where the u_i are words on the alphabet A.

At the beginning the tape contains the word u.

Repeat the following operation as long as possible.

(1) If the tape is void, stop and return "yes".
(2) If u_n is in the set I (this test is performed by machine $\mathcal{T}$), remove u_n along with the symbol # which may precede u_n.
(3) Choose randomly a rule $r = \langle\alpha|\gamma - \delta|\beta\rangle$ in R, and choose randomly, if it exists, a decomposition of u_n of the form $u_n = x\alpha\gamma y\delta\beta z$ such that neither $x\alpha\beta z$ nor $\gamma y\delta$ are empty word. Remove the subword $\gamma y\delta$ from u_n and place it at the right after a # symbol. Then shift the string $\beta z\#\gamma y\delta$ so that we have on the tape $u_1\#u_2\#\cdots\#u_{n-1}\#x\alpha\beta z\#\gamma y\delta$. If no choice exists, stop the computation.

If no computation succeeds, then the word is rejected. The length of the tape is always less that $3|u|$, because except when we decompose the word u_n at step (3), we have at most $|u|$ subwords in the iterated decomposition of u, with total length $|u|$, along with at most $|u| - 1$ separating#-symbols. When we compute the decomposition in step (3), we have in addition a copy of a part of the last word u_n whose length is at most $|u|$.

In the case of a circular splicing system, the method is almost the same. It requires an additional step between the steps (2) and (3), which consists in choosing randomly one of the conjugates of u_n, possibly retaining u_n itself. The computation of the conjugates is obtained by choosing a letter in the word, removing the part strictly before that letter and placing it at the right of the last letter of u_n, and then shift the conjugates obtained right after the last symbol #. The bound on the length of the tape is again $3|u|$ for similar reasons. □

5. Alphabetic splicing systems

We recall that a rule in a splicing system is said *alphabetic* if its handles have length at most one. A splicing system is said *alphabetic* if all its rules are alphabetic.

The splicing systems of Examples 2.3 and 2.2 are alphabetic. They generate a non-regular language, although they have a finite initial set. Let us give another example.

Example 5.1. Let $\mathcal{S} = (A, I, R)$ be the flat splicing system defined by $A = \{a, \bar{a}\}$, $I = \{a\bar{a}\}$ and $R = \{\langle\varepsilon|\varepsilon - \varepsilon|\varepsilon\rangle\}$. It generates the Dyck language. Recall that the Dyck language over $\{a, \bar{a}\}$ is the language of parenthesized expressions, a, $\bar{a}$ being viewed as a pair of matching parentheses (see, for example, [1]).

The circular splicing system $\mathcal{S} = (A, I, R)$ defined by $A = \{a, \bar{a}\}, I = \{^\frown(a\bar{a})\}$ and $R = \{\langle\varepsilon|\varepsilon - \varepsilon|\varepsilon\rangle\}$ generates the language $\hat{D}$ of words having as many a as $\bar{a}$. Indeed, the system allows to insert $a\bar{a}$ and $\bar{a}a$ anywhere, hence we get the set of words having as many a as $\bar{a}$. This language $\hat{D}$ is the circularization of the Dyck language [1].

Remark 5.2. All examples given so far show that alphabetic splicing systems generate always a context-free languages, and this is indeed the main result of the paper. Observe however that we cannot get all context-free languages and even all regular languages as splicing languages with a finite initial set. For example, we easily can see that the language $L = \{a^*c \mid n \geq 0\}$ cannot be obtained by such a splicing system: consider indeed the fact that all words of L have exactly only one c, so inserting a word of L into another word of L will produce a word out of L.

5.1. Main theorem

We now state the main theorem, namely that alphabetic rules and a context-free initial set can produce only context-free languages.

Theorem 5.3. (i) *The language generated by a circular alphabetic context-free splicing system is context-free.*
(ii) *The language generated by a flat alphabetic context-free splicing system is context-free.*

This theorem is effective, that is, we can actually construct a context-free grammar which generates the language produced by the splicing system. The rest of the paper is devoted to the (long) proof of this theorem. Before embarking on it, we give here the idea behind the proof.

We begin by the flat case. We have seen (Example 2.1) that some productions are in fact concatenations in contrast with "proper insertions" where we really put a word between two non-empty parts of another one. As these concatenations cannot be treated like "proper insertions", we are going to separate both types of productions.

In order to do this, Section 6 introduces pure splicing systems, which are splicing systems where there are only "proper insertions". Here, it is proved that the language generated by a context-free pure splicing system is context-free. The proof uses some results on context-free languages which are recalled in Section 6.1.

In the next section (Section 7), we first define concatenation systems and prove that (alphabetic) concatenation systems produce only context-free languages. Then we define heterogeneous systems, these systems combine both "proper insertions" and concatenations. The weak commutation lemma (Lemma 7.8) is then proved; this lemma explained that roughly we can do all concatenations before any insertions. This section ends with the proof of the main theorem for flat splicing systems.

Section 8 describes the relationship between flat and circular splicing systems and their languages. It contains the proof of the main theorem for circular splicing systems.

The proofs that concatenation systems and alphabetic pure systems generate context-free languages are done by giving explicitly the grammars. These grammars deviate from the standard form of grammars by the fact that the sets of derivation rules may be infinite, provided they are themselves context-free sets. It is an old result from formal language theory [15] that generalized context-free grammars of this kind still generate context-free languages. For the sake of completeness, we include a proof of this result, together with an example, in an Appendix.

We start with a technical normalization of splicing systems which will enable us to manage the ε handles more easily.

5.2. Complete set of rules

Completion of rules is a tool to manage the usage of the empty word ε among the handles $\alpha, \beta, \gamma, \delta$ of an alphabetic rule

$$r = \langle \alpha | \gamma - \delta | \beta \rangle$$

in a production

$$u, v \vdash_r w. \tag{5.1}$$

Assume first that $\delta = \varepsilon$. (The case where $\gamma = \varepsilon$ is symmetric.) In this case, the production (5.1) is valid provided v starts with γ (and of course if u has an appropriate factorization $u = x\alpha\beta y$). Let d be the final letter of v. Then the same result is obtained with the rule

$$r_d = \langle \alpha | \gamma - d | \beta \rangle,$$

with only one, but noticeable exception: this is the case where v is a single letter, that is $v = \gamma$. Observe that this may happen only if v is in the initial set of the system.

In other words, a production

$$r = \langle \alpha | \gamma - \varepsilon | \beta \rangle$$

is mandatory if and only if $\gamma \in I$. For all words $v \neq \gamma$, the production (5.1) is realized by the use of the rule r_d where d is the final letter of v. Thus a rule with $\delta = \varepsilon$ can be replaced by the set of rules r_d, for $d \in A$ with one exception.

Assume next that $\beta = \varepsilon$. (The case where $\alpha = \varepsilon$ is symmetric.) In this case, the production

$$u, v \vdash_r w$$

is valid provided α occurs in u (and v begins with γ and ends with δ). This holds in particular when α is the final letter of u. In this case, one gets

$$w = uv.$$

In other words, the application of the rule reduces to a simple concatenation. If however u has another occurrence of α, that is if $u = x\alpha y$ for some $y \neq \varepsilon$, then the rule r can be replaced by the appropriate rule $r_d = \langle \alpha | \gamma - \delta | d \rangle$, where d is the initial letter of y.

In conclusion, the use of a rule

$$r = \langle \alpha | \gamma - \varepsilon | \beta \rangle \quad (\text{resp. } r = \langle \alpha | \varepsilon - \delta | \beta \rangle)$$

can always be replaced by the use of a rule

$$r = \langle \alpha | \gamma - d | \beta \rangle \quad (\text{resp. } r = \langle \alpha | c - \delta | \beta \rangle)$$

for letters $c, d \in A$, except – and this is the only case – when the word to be inserted is a single letter which is in the initial set.

On the contrary, the use of a rule

$$r = \langle \alpha | \gamma - \delta | \varepsilon \rangle \quad (\text{resp. } r = \langle \varepsilon | \gamma - \delta | \beta \rangle)$$

can be replaced by the use of a rule

$$r = \langle \alpha | \gamma - \delta | b \rangle \quad (\text{resp. } r = \langle a | \gamma - \delta | \beta \rangle)$$

for letters $a, b \in A$, except when the result is a concatenation $w = uv$ (resp. $w = vu$).

The rules shown above have only one ε, the rules with multiple ε handles have similar properties.

Example 5.4. Let $\mathcal{S} = (A, I, R)$ with $A = \{a, b, c\}$, $I = \{abc, abb\}$ and R composed of the single rule $r = \langle b | a - b | \varepsilon \rangle$. The rule r permits the production $ab \cdot c, abb \vdash_r ab \cdot abb \cdot c$. This production could also be realized with the rule $r' = \langle b | a - b | c \rangle$ obtained from r by replacing ε by c. Similarly, the production $ab \cdot b, abb \vdash_r ab \cdot abb \cdot b$ could also be realized with the rule $r'' = \langle b | a - b | b \rangle$. Conversely, all productions that can be realized with r' and r'' can be made with r.

We can thus check that the system $\mathcal{S}' = (A, I, R')$ with the set of rules $R' = \{\langle b | a - b | \varepsilon \rangle, \langle b | a - b | a \rangle, \langle b | a - b | b \rangle, \langle b | a - b | c \rangle\}$ produces the same language as the system $\mathcal{S}$ does.

However, the production $abb \cdot, abb \vdash_r abb \cdot abb$, cannot be obtained by use of a production without ε-handle. So, the system $\mathcal{S}'' = (A, I, R'')$ with $R'' = \{\langle b | a - b | a \rangle, \langle b | a - b | b \rangle, \langle b | a - b | c \rangle\}$ does not produce the same language as the system $\mathcal{S}$ does.

We say that a splicing system $\mathcal{S} = (A, I, R)$ is *complete* if for any rule $r = \langle \alpha_1 | \alpha_3 - \alpha_4 | \alpha_2 \rangle$ in R, whenever one or several of the α_i are equal to the empty word, then the set R contains all rules obtained by replacing some or all of the empty handles by all letters of the alphabet.

For example, the system $\mathcal{S} = (A, I, R')$ is complete. The *completion* of a splicing system consists in adding to the system the rules that makes it complete. Completion is possible for alphabetic splicing systems without changing the language it produces.

Lemma 5.5. *For any alphabetic splicing system $\mathcal{S} = (A, I, R)$, the complete alphabetic splicing system $\hat{\mathcal{S}} = (A, I, \hat{R})$ obtained by completing the set of productions generates the same language.*

The proof is left to the reader.

Observe that completion may increase considerably the number of rules of a splicing system. Thus, over a k-letter alphabet, completing a rule with one ε-handle adds k rules, and if the rule has two ε-handles, completion adds $k^2 + 2k$ rules.…

In the proof of Theorem 5.3, i.e., in Sections 7, 8 we will assume that splicing systems are complete.

Remark 5.6. Complete systems may simplify some verifications. Thus, in order to verify that one may insert a letter a between some letters d and b, it suffices to check that one of $\langle d|a{-}\varepsilon|b\rangle$ or $\langle d|\varepsilon{-}a|b\rangle$ is in the set of splicing rules. Otherwise we would also have to check whether one of $\langle \varepsilon|a{-}\varepsilon|b\rangle$ or $\langle \varepsilon|a{-}\varepsilon|\varepsilon\rangle$ or $\langle d|\varepsilon{-}\varepsilon|b\rangle$, . . . is in the set of splicing rules.

6. Pure splicing systems

In this section, we consider a subclass of splicing systems called pure systems, and we prove (Theorem 6.2) that these systems generate context-free languages. We start with a description of a theorem for context-free languages that will be useful.

6.1. A theorem on context-free languages

We recall here, for the convenience of the reader, the notion of generalized context-free grammars along with one theorem on such grammars. A proof of this theorem and an example are given in the Appendix.

A *generalized grammar* G is a quadruplet (A, V, S, R), where A is a terminal alphabet, V is a non-terminal alphabet, and $S \in V$ is the axiom. The set of rules R is a possibly infinite subset of $V \times (A \cup V)^*$. For each, $v \in V$, define $M_v = \{m \mid v \to m \in R\}$. In an usual context free grammar, the sets M_v are finite. The grammar G is said to be a *generalized context-free grammar* if the languages M_v are all context-free.

Derivations are defined as usual. More precisely, given $v, w \in (A \cup V)^*$, we denote by $v \to w$ the fact that v *directly derives* w and by $v \xrightarrow{*} w$ the fact that v *derives* w. The *language generated* by G, denoted L_G, is the set of words over A derived from S, i.e., $L_G = \{u \in A^* \mid S \xrightarrow{*} u\}$.

It will be convenient, in the sequel, to use the notation $v \to \sum_{m \in M_v} m$ or $v \to M_v$ as shortcuts for the set of rules $\{v \to m \mid m \in M_v\}$.

Thus, the only difference between usual and generalized context-free grammars is that for the latter the set of productions may be infinite, and in this case it is itself context-free.

A theorem, due to Kràl [15], states that the language recognized by such a grammar remains context-free.

Theorem 6.1 ([15])**.** *The language generated by a generalized context-free grammar is context-free.*

A proof of this theorem is given in the Appendix.

6.2. Pure alphabetic splicing systems

A splicing rule $r = \langle \alpha|\gamma{-}\delta|\beta\rangle$ is *pure* if both α and β are nonempty. If the rule is alphabetic, this means that α and β are letters. A splicing system is pure if all its rules are pure.

Note that a pure splicing system produce only "proper insertions" in the sense that a word is inserted between two non-empty parts of another one.

Theorem 6.2. *The language generated by an alphabetic context-free pure splicing system is context-free.*

Proof. Let $\mathcal{S} = (A, I, R)$ an alphabetic context-free pure system. We suppose that the set R is complete.

We construct a generalized context-free grammar G with axiom S, terminal alphabet A and with non-terminals S and ${}^aB^b$, ${}_aW_b$ for $a, b \in A$, and V_a for $a \in A$.

The variable ${}_aW_b$ is used to derive words with at least two letters that begin with a letter a and end with a letter b. The variable V_a is used to derive the word a if it is in the set I.

A symbol ${}^aB^b$ is always preceded by a letter a or by a letter V_a or by a letter ${}_cW_a$, with $c \in A$, and is always followed by a letter b by or a letter V_b or by a letter ${}_bW_d$, with $d \in A$. Roughly speaking, the symbol ${}^aB^b$ denotes words for which eventually there is a letter a preceding it and a letter b following it.

We define an operation

$$\mathrm{Ins} : A^+ \to \left(A \cup \bigcup_{a,b \in A} {}^aB^b\right)^+$$

by $\mathrm{Ins}(x) = x$ for $x \in A$, and on words $a_1a_2a_3\cdots a_{n-1}a_n$ where $a_1, \ldots, a_n \in A$ and $n \geq 2$, by setting

$$\mathrm{Ins}(a_1a_2a_3\cdots a_{n-1}a_n) = a_1\ {}^{a_1}B^{a_2}\ a_2\ {}^{a_2}B^{a_3}\ a_3 \ldots a_{n-1}\ {}^{a_{n-1}}B^{a_n}\ a_n.$$

The derivation rules of G are divided into the three following groups. (Here a and b are any letters in A.)

The first group contains derivation rules that separate words of I according to their initial and final letters, and single out one-letter words.

$$S \to {}_aW_b,$$
$$S \to V_a,$$
$${}_aW_b \to \mathrm{Ins}(I \cap aA^*b),$$
$$V_a \to I \cap a.$$

We use here the convention that a derivation rule of the last type is not added if $I \cap a$ is empty. Similarly, the third sets in these derivation rules may be empty. Observe that these sets may also be context-free.
The second group reflects the application of the rules in R. It is composed of

$${}^aB^b \to {}^aB^c\ {}_cW_d\ {}^dB^b, \quad \text{for } \langle a|c{-}d|b\rangle \in R,$$
$${}^aB^b \to {}^aB^c\ V_c\ {}^cB^b, \quad \text{for } \langle a|c{-}\varepsilon|b\rangle \in R \text{ or } \langle a|\varepsilon{-}c|b\rangle \in R.$$

The third group of derivation rules is used to replace the variables ${}^aB^b$ by the empty word.

$${}^aB^b \to \varepsilon.$$

By Theorem 6.1, the language generated by G is context-free.

We claim that $L_G = \mathcal{F}(\mathcal{S})$. In order to do this we consider the grammar G without the derivations of the third group, and prove that this grammar produces $\mathrm{Ins}(\mathcal{F}(\mathcal{S}))$.

More formally, we denote by L'_G the language obtained without applying the productions of the third type, and by $L'_G({}_aW_b)$ and by $L'_G(V_a)$ the languages obtained when starting with the variable ${}_aW_b$ (resp. with V_a), and we prove that $L'_G = \mathrm{Ins}(\mathcal{F}(\mathcal{S}))$.

First, we prove the inclusion $L'_G \subseteq \mathrm{Ins}(\mathcal{F}(\mathcal{S}))$. For this, we prove by induction on the length of the leftmost derivations in G' that for all letters $a, b \in A$, we have $L'_G({}_aW_b) \subseteq \mathrm{Ins}(\mathcal{F}(\mathcal{S}) \cap aA^*b)$ and that $L'_G(V_a) \subseteq \mathrm{Ins}(\mathcal{F}(\mathcal{S}) \cap a)$.

A derivation $X \xrightarrow{*} v$ is called *terminal* if v does not contains any occurrence of variables other than ${}^aB^b$, for $a, b \in A$. It is easy to check that the length of terminal derivations are always odd.

The only terminal derivations of length one are

$${}_aW_b \to \mathrm{Ins}(I \cap aA^*b),$$
$$V_a \to I \cap a,$$

and the inclusion is clear.

Assume that the hypotheses of induction hold for derivations of length less than k and let u be a word obtained by a derivation of length k. Since the length of the derivation is greater than 1, the derivation starts with a derivation step $S \to {}_aW_b$ for some $a, b \in A$.

The last derivation step have one of the following form

$${}_eW_f \to x, \quad \text{with } x \in \mathrm{Ins}(I \cap eA^*f),$$
$$V_e \to e, \quad \text{with } e \in I,$$

And the derivation step which produces the symbol ${}_eW_f$ or the symbol V_e is respectively of the form

$${}^cB^d \to {}^cB^e\ {}_eW_f\ {}^fB^d$$
$${}^cB^d \to {}^cB^e\ V_e\ {}^eB^c$$

for suitable letters c, d, e, f.

As the derivation is leftmost, this latter step is the step before the last one.

In the first case, there are words v, w such that

$${}_aW_b \xrightarrow{*} v\,{}^cB^d\,w \to v\,{}^cB^e\,{}_eW_f\,{}^fB^d\,w \to v\,{}^cB^e x\,{}^fB^d\,w = u.$$

By induction, $v\,{}^cB^d\,w \in \mathrm{Ins}(\mathcal{F}(\mathcal{S}) \cap aA^*b)$. Since the derivation rule ${}^cB^d \to {}^cB^e\,{}_eW_f\,{}^fB^d$ is in G, there is a splicing rule $\langle c|e{-}f|d\rangle$ in R. Consequently, the word u is in $\mathrm{Ins}(\mathcal{F}(\mathcal{S}) \cap aA^*b)$. The second case is similar. This proves the inclusion $L'_G \subseteq \mathrm{Ins}(\mathcal{F}(\mathcal{S}))$.

Now, we prove the inclusion $\mathrm{Ins}(\mathcal{F}(\mathcal{S})) \subseteq L'_G$. For this, we prove that for all letters $a, b \in A$, we have $\mathrm{Ins}(\mathcal{F}(\mathcal{S}) \cap aA^*b) \subseteq L'_G({}_aW_b)$ and $\mathrm{Ins}(\mathcal{F}(\mathcal{S}) \cap a) \subseteq L'_G(V_a)$.

We observe that for a letter a, one has $\mathcal{F}(\mathcal{S}) \cap a = I \cap a = \mathrm{Ins}(\mathcal{F}(\mathcal{S}) \cap a)$. The letter a is thus obtained by the derivation $V_a \to I \cap a$. Thus we have $\mathrm{Ins}(\mathcal{F}(\mathcal{S}) \cap a) \subseteq L'_G(V_a)$ for all letters $a \in A$.

Let us prove the inclusions $\mathrm{Ins}(\mathcal{F}(\mathcal{S}) \cap aA^*b) \subseteq L_{G'}({}_aW_b)$ by induction on the number of splicing rules used for the production of a word in $\mathcal{F}(\mathcal{S}) \cap aA^*b$.

Let $u \in \mathcal{F}(\mathcal{S}) \cap aA^*b$. If no splicing rule is used, then $u \in I \cap aA^*b$. The word $\mathrm{Ins}(u)$ is obtained by the application of the corresponding derivation rule ${}_aW_b \to \mathrm{Ins}(u)$ which is in the set ${}_aW_b \to \mathrm{Ins}(I \cap aA^*b)$. Thus $u \in L'_G({}_aW_b)$.

Assume that the inductive hypothesis holds for the words obtained by less than k splicing operations, and that u is obtained by application of $k \geq 1$ splicing operations. We consider the last insertion that leads to u: there exist three nonempty words v, w and x and a pure rule $r \in R$, such that $v \cdot w, x \vdash_r u = v \cdot x \cdot w$, and moreover vw and x are words of $\mathcal{F}(\mathcal{S})$ obtained by less than k splicing operations.

Two cases may occur, for suitable letters e and f:

$$x \in \mathcal{F}(\mathcal{S}) \cap eA^*f,$$
$$x \in \mathcal{F}(\mathcal{S}) \cap e.$$

Consider the first case. Let c be the last letter of v, and let d be the first letter of w. Then $r = \langle c|e{-}f|d\rangle$. By induction hypothesis, we have ${}_aW_b \xrightarrow{*} \mathrm{Ins}(v)\,{}^cB^d\,\mathrm{Ins}(w)\,(= \mathrm{Ins}(vw))$ and ${}_eW_f \xrightarrow{*} \mathrm{Ins}(x)$. Moreover, the rule r shows that the derivation rule ${}^cB^d \to {}^cB^e\,{}_eW_f\,{}^fB^d$ is in the grammar G. Thus combining these three derivations, we obtain

$$\begin{aligned} {}_aW_b &\xrightarrow{*} \mathrm{Ins}(v)\,{}^cB^d\,\mathrm{Ins}(w) \to \mathrm{Ins}(v)\,{}^cB^e\,{}_eW_f\,{}^fB^d\,\mathrm{Ins}(w) \\ &\xrightarrow{*} \mathrm{Ins}(v)\,{}^cB^e\,\mathrm{Ins}(x)\,{}^fB^d\,\mathrm{Ins}(w). \end{aligned}$$

Thus $\mathrm{Ins}(u) \in L'_G({}_aW_b)$. The second case is similar.

This shows the inclusion $\mathrm{Ins}(\mathcal{F}(\mathcal{S})) \subseteq L'_G$. Consequently $\mathrm{Ins}(\mathcal{F}(\mathcal{S})) = L'_G$, and quite obviously, we can deduce $\mathcal{F}(\mathcal{S}) = L_G$. □

The following example is designed to illustrate all the features of the formal proof. So it is not so easy to follow.

Example 6.3. Consider the pure splicing system

$$\mathcal{S} = (A, I, R)$$

with $A = \{a, b, c\}$, $I = c^*ab \cup c$, and with R composed of the rules

$$r = \langle c|\varepsilon{-}b|a\rangle, \qquad r' = \langle c|\varepsilon{-}b|b\rangle, \qquad r'' = \langle c|\varepsilon{-}b|c\rangle, \qquad r''' = \langle a|a{-}b|b\rangle.$$

This splicing system generates the language $\mathcal{F}(\mathcal{S}) = c(c \cup L)^+L \cup \{c\}$, with $L = \{a^nb^n \mid n \geq 1\}$.

For the construction of the grammar for $\mathcal{F}(\mathcal{S})$, we add the completions of the rules r, r' and r''. We also discard tacitly useless variables. Now, we observe that $I \cap aA^*b = ab$, $I \cap cA^*b = c^+ab$, $I \cap c = c$, and that the other intersections are empty. Thus, the first group of derivation rules of the grammar is the following.

$$\begin{aligned} S &\to {}_aW_b \mid {}_cW_b \mid V_c \\ {}_aW_b &\to a\,{}^aB^b\,b, \quad (= \mathrm{Ins}(I \cap aA^*b) = \mathrm{Ins}(ab)), \\ {}_cW_b &\to (c\,{}^cB^c)^*c\,{}^cB^a\,a\,{}^aB^b\,b, \quad (= \mathrm{Ins}(I \cap cA^*b) = \mathrm{Ins}(c^+ab)), \\ V_c &\to c. \end{aligned}$$

We observe by inspection, that there is no derivation rule starting with ${}_bW_x$, ${}_xW_a$ or ${}_xW_c$, for $x \in A$, and similarly for V_a, V_b. This leaves only the following second group of rules.

$$\begin{aligned} &{}^aB^b \to {}^aB^a\,{}_aW_b\,{}^bB^b \quad \text{(because of rule } r'''\text{)} \\ &{}^cB^a \to {}^cB^a\,{}_aW_b\,{}^bB^a \quad \text{(due to rule } \langle c|b{-}b|a\rangle\text{, completion of } r\text{)} \\ &{}^cB^a \to {}^cB^c\,{}_cW_b\,{}^bB^a \quad \text{(due to rule } \langle c|c{-}b|a\rangle\text{, completion of } r\text{)} \\ &{}^cB^b \to {}^cB^a\,{}_aW_b\,{}^bB^b \quad \text{(due to rule } \langle c|a{-}b|b\rangle\text{, completion of } r'\text{)} \\ &{}^cB^b \to {}^cB^c\,{}_cW_b\,{}^bB^b \quad \text{(due to rule } \langle c|c{-}b|b\rangle\text{, completion of } r'\text{)} \\ &{}^cB^c \to {}^cB^a\,{}_aW_b\,{}^bB^c \quad \text{(due to rule } \langle c|a{-}b|c\rangle\text{, completion of } r''\text{)} \\ &{}^cB^c \to {}^cB^c\,{}_cW_b\,{}^bB^c \quad \text{(due to rule } \langle c|c{-}b|c\rangle\text{, completion of } r''\text{)}. \end{aligned}$$

When looking for the final grammar (including the third group), we may observe that the ${}^cB^b$ is no right part of a rule, we can then discard the rules having ${}^cB^b$ as a left part.

We also remark that the variables ${}^bB^x$ for $x \in A$, and ${}^aB^a$ only produce the empty word. Also they can be replaced by ε everywhere in the grammar. The variable V_c produces only c, and can be replaced by it. Also, it is easily seen that ${}^cB^a$ and ${}^cB^c$

generate the same language. This leads to the following grammar, where we write, for easier reading, X for ${}_aW_b$ and Y for ${}_cW_b$, and T for ${}^cB^a$ and ${}^cB^c$, U for ${}^aB^b$.

$$S \to X \mid Y \mid c$$
$$X \to aUb$$
$$Y \to (cT)^+aUb$$
$$U \to X \mid \varepsilon$$
$$T \to TX \mid TY \mid \varepsilon.$$

Now, we replace aUb by X in the third line, and then replace U by its production everywhere. We also replace the production of T by $(X + Y)^*$, and replace the T

$$S \to X \mid Y \mid c$$
$$X \to aXb \mid ab$$
$$Y \to (c(X + Y)^*)^+X = c(c + X + Y)^*X.$$

We then observe that the language over C, X generated by Y is exactly $c(c + X)^*X$, we thus obtain

$$S \to X \mid Y \mid c$$
$$X \to aXb \mid ab$$
$$Y \to c(c + X)^*X.$$

Now we replace X by the language it generates: $L = \{a^nb^n \mid n \geq 1\}$, and we conclude. This generalized context-free grammar generates the language $\mathcal{F}(\mathcal{S}) = c(c \cup L)^*L \cup L \cup \{c\}$, with $L = \{a^nb^n \mid n \geq 1\}$.

7. Concatenation systems

We introduce a classification of the productions generated in a splicing system by defining two kinds of productions called proper insertions and concatenations.

Let $r = \langle\alpha|\gamma-\delta|\beta\rangle$ be a splicing rule. The production $x\alpha \cdot \beta y, \gamma z\delta \vdash_r x\alpha \cdot \gamma z\delta \cdot \beta y$ is a *proper insertion* if $x\alpha \neq \varepsilon$ and $\beta y \neq \varepsilon$, it is a *concatenation* otherwise. If r is a pure rule, then its productions are always proper insertions.

Of course, the rule r can produce a concatenation only if $\beta = \varepsilon$ or $\alpha = \varepsilon$. However, such rules can be used for both kinds of productions. Consider for example the rule $r = \langle a|c-d|\varepsilon\rangle$. Then the production $aa\cdot, cad \vdash_r aa \cdot cad$ is a concatenation, while the production $a \cdot a, cad \vdash_r a \cdot cad \cdot a$ is a proper insertion. We consider now rules which are not pure, and we restrict their usage to concatenations. This leads to the notion of concatenation systems. We then show that alphabetic context-free concatenation systems only generate context-free languages.

7.1. Concatenation systems

A *concatenation system* is a triplet $\mathcal{T} = (A, I, R)$, where A is an alphabet, I is a set of words over A, called the *initial set* and R is a finite set of *concatenation rules*. A concatenation rule r is a quadruplet of words over A. It is denoted $r = [\alpha-\beta|\gamma-\delta]$, to emphasize the special usage which is made of such a rule.

A concatenation rule $r = [\alpha-\beta|\gamma-\delta]$ can be applied to words u and v provided $u \in \alpha A^*\beta$ and $v \in \gamma A^*\delta$. Applying r to the pair (u, v) gives the word $w = uv$. This is denoted by $u, v \models_r w$ and is called a *concatenation production*.

The *language generated* by the system $\mathcal{T}$, denoted by $\mathcal{K}(\mathcal{T})$, is the smallest language containing I and closed under the application of the rules of R.

Again, the system $\mathcal{T}$ is alphabetic if every rule in R have handles of length at most one. It is context-free if the initial set I is context-free. The notion of complete set is similar to the one for splicing rules.

7.2. Alphabetic concatenation

This section is devoted to the proof of the following theorem.

Theorem 7.1. *The language generated by an alphabetic context-free concatenation system is context-free.*

Proof. Let $\mathcal{T} = (A, I, R)$ be an alphabetic context-free concatenation system. We suppose that the set R is complete. Set $K = \mathcal{K}(\mathcal{T})$.

We construct a grammar $G = (A, V, S, R)$ which produce K. The grammar is quite similar to that built for Theorem 6.2. The grammar G has A as set of terminal symbols, the set of non-terminal symbols being $V = \{S\} \cup \{{}_aW_b \mid a, b \in A\} \cup \{V_a \mid a \in A\}$. The axiom is S.

As in the proof of Theorem 6.2, the purpose of the variables is the following. The symbol ${}_aW_b$ is used to derive words of length at least 2 that start with the letter a and end with the letter b, that is the set $K \cap aA^*b$. Similarly, the symbol V_a will be used to derive the word a if it is in K.

The derivation rules of the grammar G are divided in two groups. In the following, a and b are any letters in A.

The first group contains derivation rules that separate words according to their initial and final letters, and single out one-letter words:

$$S \to {}_aW_b,$$
$$S \to V_a,$$
$${}_aW_b \to I \cap aA^*b,$$
$$V_a \to I \cap a.$$

The second group of rules deals with concatenations:

$${}_aW_b \to {}_aW_c\ {}_dW_b, \quad \text{for } [a{-}c|d{-}b] \in R,$$
$${}_aW_b \to V_a\ {}_cW_b, \quad \text{for } [\varepsilon{-}a|c{-}b] \in R \text{ or } [a{-}\varepsilon|c{-}b] \in R,$$
$${}_aW_b \to {}_aW_c\ V_b, \quad \text{for } [a{-}c|\varepsilon{-}b] \in R \text{ or } [a{-}c|b{-}\varepsilon] \in R,$$
$${}_aW_b \to V_a\ V_b, \quad \text{for } [\varepsilon{-}a|\varepsilon{-}b] \in R \text{ or } [a{-}\varepsilon|\varepsilon{-}b] \in R \text{ or } [a{-}\varepsilon|b{-}\varepsilon] \in R \text{ or } [\varepsilon{-}a|b{-}\varepsilon] \in R.$$

By construction, the language L_G generated by G is context-free by Theorem 6.1.

We claim that $L_G = K$. We first prove the inclusion $L_G \subseteq K$. For this, we show, by induction on the length of the derivation in G, that for all letters $a, b \in A$, we have $L_G({}_aW_b) \subseteq K \cap aA^*b$ and that $L_G(V_a) \subseteq K \cap a$.

The only terminal derivations of length 1 are

$${}_aW_b \to w \quad \text{with } w \in I \cap aA^*b \subseteq K \cap aA^*b,$$
$$V_a \to a \quad \text{with } a \in I \cap a \subseteq K \cap a.$$

Thus the inclusion holds in this case.

Assume that the hypotheses of induction are true for derivations of length less that k and let u a word obtained by a derivation of length k. Since $k \geq 2$, the first derivation rule is one of the second group, and the derivation has one of the forms

$${}_aW_b \to {}_aW_c\ {}_dW_b \overset{*}{\to} u$$
$${}_aW_b \to V_a\ {}_cW_b \overset{*}{\to} u$$
$${}_aW_b \to {}_aW_c\ V_b \overset{*}{\to} u$$
$${}_aW_b \to V_a\ V_b \overset{*}{\to} u$$

for some $a, b, c, d \in A$. In the first case, we have $u = u_1u_2$, with ${}_aW_c \overset{*}{\to} u_1$ and ${}_dW_b \overset{*}{\to} u_2$, both derivations having length strictly less than k. By the inductive hypotheses, $u_1 \in K \cap aA^*c$ and $u_2 \in K \cap dA^*b$. Moreover, since ${}_aW_b \to {}_aW_c\ {}_dW_b$ is a derivation rule in G, one has $[a{-}c|d{-}b] \in R$. This ensures that $(K \cap aA^*c)(K \cap dA^*b) \subseteq K$ and then $(K \cap aA^*c)(K \cap dA^*b) \subseteq K \cap aA^*b$. Consequently, $u = u_1u_2$ is in $K \cap aA^*b$. The other cases are similar. This proves the inclusion $L_G \subseteq K$.

We now prove the converse inclusion $K \subseteq L_G$. For this, we prove that for all letters $a, b \in A$, we have $K \cap aA^*b \subseteq L_G({}_aW_b)$ and that $K \cap a \subseteq L_G(V_a)$.

It is easy to see, that if $a \in K$ then $V_a \to I \cap a = \{a\}$. Thus $K \cap a \subseteq L_G(V_a)$, for all letter a in A.

The inclusions $K \cap aA^*b \subseteq L_G({}_aW_b)$ are proved by induction on the number of the concatenation operations used. Let $u \in K \cap aA^*b$.

If u is obtained without any concatenation, then $u \in I \cap aA^*b$, and since ${}_aW_b \to I \cap aA^*b$ is a derivation rule in G, we have $u \in L_G({}_aW_b)$.

Assume that the inductive hypothesis holds for words obtained by less than k concatenations, and that u is obtained by k concatenations. Then there exist two words u_1 and u_2 such that $u = u_1u_2$ and such that u_1 and u_2 are obtained by less than k concatenations. There are four cases to consider, according to the concatenation rule used to produce u from u_1 and u_2. The cases are the following.

$$u_1 \in K \cap aA^*c, \qquad u_2 \in K \cap dA^*b,$$
$$u_1 \in K \cap aA^*c, \qquad u_2 \in K \cap b,$$
$$u_1 \in K \cap a, \qquad u_2 \in K \cap dA^*b,$$
$$u_1 \in K \cap a, \qquad u_2 \in K \cap b.$$

Consider the first case (the others are similar). Since $u \in K$, there is a concatenation rule $[a{-}c|d{-}b]$ in R. Consequently, there exists in G a derivation rule ${}_aW_b \to {}_aW_c\ {}_dW_b$. By induction hypothesis, there is a derivation ${}_aW_c \overset{*}{\to} u_1$, and a derivation

${}_dW_b \xrightarrow{*} u_2$. It follows that

$$ {}_aW_b \to {}_aW_c\ {}_dW_b \xrightarrow{*} u_1u_2, $$

and subsequently that $u \in L_G({}_aW_b)$. This proves the inclusion $K \subseteq L_G$, and thus the claim. Since L_G is context-free, the language K is also context-free. This completes the proof. □

Remark 7.2. The example of Section 4.1 shows that Theorem 6.2 does not hold for systems which are pure but not are not alphabetic. However, Theorem 7.1 holds for concatenation systems without the requirement that they are alphabetic. The proof is quite analogous to the alphabetic case. Indeed, if k is the maximum length of an handle, instead of indexing variables W by letters, it suffices to index them by words of length k, and to index variables V by words of length strictly inferior to $2k$. There are finitely many such variables; then the prof goes along the same lines.

Example 7.3. Consider the concatenation system $\mathcal{T} = (A, I, R)$ over the alphabet $A = \{a, b, c\}$, with $I = \{ab, c\}$, and with R composed of the concatenation rules

$$ [\varepsilon{-}c|\varepsilon{-}b], $$
$$ [\varepsilon{-}c|x{-}b] \quad \text{for } x \in A. $$

The completion of the system gives the concatenation rules

$$ [\varepsilon{-}c|\varepsilon{-}b], $$
$$ [\varepsilon{-}c|x{-}b] \quad \text{for } x \in A $$
$$ [y{-}c|\varepsilon{-}b] \quad \text{for } y \in A $$
$$ [y{-}c|x{-}b] \quad \text{for } x, y \in A. $$

According to the construction of the previous proof, these concatenation rules give the derivation rules

$$ {}_cW_b \to V_c\ V_b \tag{7.1} $$
$$ {}_cW_b \to V_c\ {}_xW_b \quad \text{for } x \in A \tag{7.2} $$
$$ {}_yW_b \to {}_yW_c\ V_b \quad \text{for } y \in A \tag{7.3} $$
$$ {}_yW_b \to {}_yW_c\ {}_xW_b \quad \text{for } x, y \in A. \tag{7.4} $$

The first group of derivation rules is composed only of

$$ S \to {}_xW_y \quad \text{for } x, y \in A $$
$$ S \to V_c $$
$$ {}_aW_b \to ab $$
$$ V_c \to c $$

because of the set I of initial words. Since there is no derivation rule starting with V_b, the derivation rules (7.1) and (7.3) are useless and can be removed. Similarly, there is no derivation rule starting with ${}_yW_c$, so the derivation rules (7.4) can be removed. For the same reason, the variable ${}_bW_b$ can be removed. Finally, we get the grammar

$$ S \to {}_aW_b \mid {}_cW_b \mid V_c $$
$$ {}_aW_b \to ab $$
$$ V_c \to c $$
$$ {}_cW_b \to V_c\ {}_aW_b \mid V_c\ {}_cW_b. $$

The language obtained is $c^*ab + c$.

Remark 7.4. The language $\mathcal{K}(\mathcal{T})$ generated by a concatenation system $\mathcal{T} = (A, I, R)$ may not be regular, even if I is finite. Consider indeed the system given by $I = \{ab, a, b, c, d\}$ and

$$ R = \{r_1 = [\varepsilon{-}c|a{-}b], r_2 = [c{-}b|d{-}\varepsilon], r_3 = [\varepsilon{-}a|c{-}d], r_4 = [a{-}d|b{-}\varepsilon]\}. $$

The language obtained is $\mathcal{K}(\mathcal{T}) = L \cup cL \cup cLd \cup acLd$ where L denotes the $L = \{(ac)^n ab(db)^n \mid n \geq 0\}$. Indeed, we can check that at each step we can apply only one rule, using in this order $r_1, r_2, r_3, r_4r_1, r_2, \ldots$; yielding the words cab, $cabd$, $acabd$, $acabdb$, $cacabdb$, ... This language is clearly not regular.

7.3. Heterogeneous systems

A system is a *heterogeneous system* if all its rules are either pure rules or concatenation rules.

The aim of heterogeneous systems is to separate the splicing rules according to their usage. A pure rule is used for a proper insertion, that is for producing a word $w = xvy$ from words $u = xy$ and v, with $x, y \neq \varepsilon$. On the contrary, a concatenation rule produces the word $w = uv$ or $w = vu$, this handles precisely the case where $x = \varepsilon$ or $y = \varepsilon$.

The following proposition shows that for any flat alphabetic splicing system, there is an alphabetic heterogeneous system with same initial set I which generates the same language.

Proposition 7.5. *Let $\mathcal{S} = (A, I, R)$ be a complete alphabetic flat splicing system, and let $\mathcal{S}' = (A, I, R' \cup R'')$ be the heterogeneous system with same initial set I, where R' is the set of pure rules of R, and*

$$R'' = \{[\varepsilon - \alpha | \gamma - \delta] \mid \langle \alpha | \gamma - \delta | \varepsilon \rangle \in R\} \cup \{[\gamma - \delta | \beta - \varepsilon] \mid \langle \varepsilon | \gamma - \delta | \beta \rangle \in R\}.$$

Then S and S' generate the same language.

Proof. When using a rule of the completed set R, either it is a proper insertion and it is simulated by a rule of R', or it is a concatenation and it is simulated by the corresponding rule of R''. Hence $\mathcal{F}(\mathcal{S}) \subseteq \mathcal{F}(\mathcal{S}')$.

Conversely, using a rule of R' or R'' is simulated by using the corresponding rule of R. □

Example 7.6. Let $\mathcal{S}$ be the flat splicing system (A, I, R) with $A = \{a, b, c\}$, $I = \{ab, c\}$, and $R = \{\langle a | a - b | b \rangle, \langle c | \varepsilon - b | \varepsilon \rangle\}$.

We complete R. The complete set of rules for R is

$$\begin{array}{ll} \langle a | a - b | b \rangle & \\ \langle c | \varepsilon - b | \varepsilon \rangle & \\ \langle c | x - b | y \rangle & \text{for } x, y \in \{a, b, c\} \\ \langle c | x - b | \varepsilon \rangle & \text{for } x \in \{a, b, c\} \\ \langle c | \varepsilon - b | x \rangle & \text{for } x \in \{a, b, c\}. \end{array}$$

The heterogeneous system $\mathcal{S}'$ corresponding to $\mathcal{S}$ is the system $\mathcal{S}' = (A, I, R')$ with R' is composed of the pure rules

$$\begin{array}{ll} \langle a | a - b | b \rangle & \\ \langle c | x - b | y \rangle & \text{for } x, y \in \{a, b, c\} \\ \langle c | \varepsilon - b | x \rangle & \text{for } x \in \{a, b, c\} \end{array}$$

and with the concatenation rules

$$\begin{array}{ll} [\varepsilon - c | x - b] & \text{for } x \in \{a, b, c\} \\ [\varepsilon - c | \varepsilon - b] & \end{array}$$

which give the concatenation rules of Example 7.3.

7.4. Weak commutation of concatenations and proper insertions

Given a heterogeneous system $\mathcal{S} = (A, I, R)$, a *production sequence* is a sequence $[\pi_1; \pi_2; \ldots; \pi_n]$ of productions such that, setting

$$\pi_k = (u_k, v_k \vdash_{r_k} w_k) \quad \text{for } 1 \leq k \leq n,$$

each u_k and v_k is either an element of I, or is equal to one of the words $u_1, v_1, w_1, \ldots, u_{k-1}, v_{k-1}, w_{k-1}$. Each word w_i is a *result* of the production sequence. The length of the sequence is n. By convention, there is a production sequence of length 0 for each $w \in I$, denoted by $[w]$. Its result is w.

Example 7.7. Consider the pure system over $A = \{a, b\}$ with initial set $I = \{ab\}$ and the unique splicing rule $r = \langle a | a - b | b \rangle$. In this system, the only splicing sequence of length 0 is $[ab]$. Both production sequences (we omit the reference to r)

$$[ab, ab \vdash a^2b^2;\ \ a^2b^2, a^2b^2 \vdash a^4b^4]$$

and

$$[ab, ab \vdash a^2b^2;\ \ ab, a^2b^2 \vdash a^3b^3;\ \ a^3b^3, ab \vdash a^4b^4]$$

have both a^4b^4 as a result. While a^3b^3 is a result of only the second one.

Clearly, the language $\mathcal{F}(\mathcal{S})$ generated by a heterogeneous system $\mathcal{S}$ is the set of the results of all its production sequences.

We show that, in an alphabetic splicing system, one always can choose a particular type of production sequence for the computation of a word, namely a sequence where the concatenations are performed before proper insertions. This is stated in the following lemma.

Lemma 7.8. *Let $\mathcal{S} = (A, I, R)$ be an alphabetic heterogeneous splicing system. Given a sequence of proper insertions and concatenation productions with a result u, there exists another sequence with the same result u, using the same rules of proper insertions and concatenations, and such that all concatenation productions occur before any proper insertion production.*

Proof. We need only to prove that if a sequence contains a subsequence consisting of a proper insertion followed immediately by a concatenation, then it is possible to replace these two productions by one concatenation followed by some proper insertions using the same rules and whose set of results contains the results of the subsequence.

Let $r_1 = \langle \alpha | \gamma - \delta | \beta \rangle$ be a pure rule and let $r_2 = [\zeta - \eta | \mu - \nu]$ be a concatenation rule, and assume that there is a production sequence $\sigma = [\pi_1; \pi_2]$, with

$$\pi_1 = (u, v \vdash_{r_1} w), \qquad \pi_2 = (p, s \models_{r_2} t),$$

where u, v, w, p, s, t are words and $t = ps$. We assume that u, v, p, s are all non-empty. If neither p nor s is equal to w we replace the sequence $[\pi_1;\ \pi_2]$ by $[\pi_2;\ \pi_1]$, and we get the result.

Assume now that $p = w$ or $s = w$. Since π_1 is a proper insertion, there exists a factorization $u = u_1 \cdot u_2$, with u_1 and u_2 non-empty words, such that $w = u_1 \cdot v \cdot u_2$, and the production π_1 can be rewritten as $\pi_1 = (u_1 \cdot u_2, v \vdash_{r_1} u_1 \cdot v \cdot u_2)$.

There are two cases to be considered.

1. $p = w, s \neq w$ (or the symmetric case $p \neq w, s = w$);
2. $p = s = w$.

Case 1: In this case, we have

$$\pi_1 = (u_1 \cdot u_2, v \models_{r_1} u_1 \cdot v \cdot u_2), \qquad \pi_2 = (u_1 v u_2, s \Vdash_{r_2} u_1 v u_2 \cdot s),$$

and, in view of the production π_2, one has $u_1 \in \zeta A^*$ and $u_2 \in A^* \eta$ (here we use the fact that u_1 and u_2 are non-empty, and that ζ and η have at most one letter).

We replace the sequence $[\pi_1; \pi_2]$ by $[\pi_3;\ \pi_4;\ \pi_1]$, where

$$\pi_3 = (u_1 u_2, s \models_{r_2} u_1 u_2 \cdot s), \qquad \pi_4 = (u_1 \cdot u_2 s, v \vdash_{r_1} t = u_1 \cdot v \cdot u_2 s).$$

The last production π_1 being necessary only to keep the result $w = u_1 \cdot v \cdot u_2$.

The concatenation π_3 is valid because $u_1 u_2 \in \zeta A^* \eta$ and $s \in \mu A^* \nu$.

Case 2: In this case,

$$\pi_1 = (u_1 \cdot u_2, v \vdash_{r_1} u_1 \cdot v \cdot u_2),$$
$$\pi_2 = (u_1 v u_2, u_1 v u_2 \models_{r_2} u_1 v u_2 \cdot u_1 v u_2),$$

and the sequence $[\pi_1;\ \pi_2]$ is replaced by $[\pi_3;\ \pi_4;\ \pi_5;\ \pi_1]$, where

$$\pi_3 = (u_1 u_2, u_1 u_2 \models_{r_2} u_1 u_2 \cdot u_1 u_2),$$
$$\pi_4 = (u_1 \cdot u_2 u_1 u_2, v \vdash_{r_1} u_1 \cdot v \cdot u_2 u_1 u_2),$$
$$\pi_5 = (u_1 v u_2 u_1 \cdot u_2, v \vdash_{r_1} u_1 v u_2 u_1 \cdot v \cdot u_2).$$

This proves the lemma. □

Remark 7.9. The proposition does not hold anymore if the rules are not alphabetic. Consider, for example, the rules $r_1 = \langle a | x - y | b \rangle$ and $r_2 = [z - t | ax - \varepsilon]$. Then the splicing sequence

$$[a \cdot b, xy \vdash_{r_1} a \cdot xy \cdot b;\ \ zt, axyb \models_{r_2} zt \cdot axyb]$$

cannot be replaced by a sequence where proper insertions occur after concatenations.

The following theorem is an immediate corollary of the previous lemma.

Proposition 7.10. *For any language L generated by an alphabetic heterogeneous system $\mathcal{S} = (A, I, R)$, there exist a set of alphabetic concatenation rules R' and a set of pure alphabetic rules R'', such that*

$$L = \mathcal{F}(A, \mathcal{K}(A, I, R'), R'').$$

We have eventually all the ingredients to prove the main theorem in the case of a flat system. Indeed, given an alphabetic splicing system, we can transform it into an heterogeneous system by separating the proper insertion and concatenation productions (Theorem 7.10). As we can make the concatenations first and the insertions after, we prove that the language produced by the concatenations only is context-free (Theorem 7.1) , this latter language used as initial language along with the proper insertions produces the final language which is also context-free (Theorem 6.2).

Theorem 7.11. *Let $\mathcal{S} = (A, I, R)$ be a flat alphabetic context-free splicing system. Then $\mathcal{F}(\mathcal{S})$ is context-free.*

Proof. By Theorem 7.10, $\mathcal{F}(\mathcal{S}) = \mathcal{F}(A, \mathcal{K}(A, I, R'), R'')$. The language $L = \mathcal{K}(A, I, R')$ is context-free in view of Theorem 7.1. The language $\mathcal{F}(A, L, R'')$ is context-free by Theorem 6.2. Thus $\mathcal{F}(\mathcal{S})$ is context-free. □

Example 7.12. Consider again the splicing system $\mathcal{S} = (A, I, R)$ with $A = \{a, b, c\}$, $I = \{ab, c\}$, and $R = \{\langle a|a{-}b|b\rangle, \langle c|\varepsilon{-}b|\varepsilon\rangle\}$. The homogeneous system corresponding to $\mathcal{S}$ is given in Example 7.6. The associated concatenation system $\mathcal{T} = (A, I, R')$ has the concatenation rules R' composed of

$$[\varepsilon{-}c|x{-}b] \quad \text{for } x \in \{a, b, c\}$$
$$[\varepsilon{-}c|\varepsilon{-}b]$$

and we have seen in Example 7.3 that it generates the language $\mathcal{K}(T) = c^*ab \cup c$. The pure system has the set R'' of rules consisting in

$$\langle a|a{-}b|b\rangle$$
$$\langle c|x{-}b|y\rangle \quad \text{for } x, y \in \{a, b, c\}$$
$$\langle c|\varepsilon{-}b|x\rangle \quad \text{for } x \in \{a, b, c\}.$$

This system is the completion of the system of Example 6.3, it generates the context-free language $\mathcal{F}(\mathcal{S}) = c(c \cup L)^+L \cup \{c\}$, with $L = \{a^nb^n \mid n \geq 1\}$.

8. Circular splicing

Recall that a circular splicing system $\mathcal{S} = (A, I, R)$ is composed of an alphabet A, an initial set I of circular words, and a finite set R of rules. A rule $r = \langle\alpha|\gamma{-}\delta|\beta\rangle$ is applied to two circular words $^\sim u$ and $^\sim v$, provided there exist words x, y such that $u \sim \beta x\alpha$ and $v \sim \gamma y\delta$ and produces the circular word $^\sim\beta x\alpha\gamma y\delta$.

Example 8.1. Consider the circular splicing system over $A = \{a, b\}$, with initial set $I = \{^\sim ab\}$ and with the single rule $\langle a|a{-}b|b\rangle$. The rule expresses the fact that a word starting with the letter a and ending with a letter b can be inserted, in a circular word, between a letter a followed by a letter b. As a consequence, the set generated by the system is the $^\sim\{a^nb^n \mid n \geq 1\}$.

We now show, on this example, that an alphabetic circular splicing system, operating on circular words and generating a circular language, can always be simulated by a flat heterogeneous splicing system. This system has the same initial set (up to full linearization), but has an augmented set of rules, obtained by a kind of conjugacy of the splicing rules. To be more precise, we introduce the following notation. Given an alphabetic rule $\langle\alpha|\gamma{-}\delta|\beta\rangle$, we denote by $^\sim r$ the set

$$^\sim r = \{\langle\alpha|\gamma{-}\delta|\beta\rangle, \langle\delta|\beta{-}\alpha|\gamma\rangle, [\beta{-}\alpha|\gamma{-}\delta], [\gamma{-}\delta|\beta{-}\alpha]\}.$$

The rules of the flat splicing system simulating the circular system are the sets $^\sim r$, for all rules r of the circular system. We illustrate the construction on the previous example.

Example 8.2. Consider the circular splicing system over $A = \{a, b\}$, initial set $^\sim I = \{^\sim ba\}$ and with the single rule $\langle a|a{-}b|b\rangle$. As seen before, (see Example 2.3), this system generates $L = \{^\sim a^nb^n \mid n > 0\}$.

If we "linearize" the system without any care, we obtain the flat splicing system over $A = \{a, b\}$, initial set $I' = \{ba\}$ and with the single rule $\langle a|a{-}b|b\rangle$ in which, clearly, the rule cannot be applied, and consequently the language generated by the system reduces to I. Note that if we use the complete linearization of I, $Lin(I)$ as initial set without changing the set of rules, we obtained $\{a^nb^n \mid n > 0\}$, which is a linearization of L, but not its full-linearization. Moreover, there are cases, when this fact will not be true (see Example 2.7).

To obtain really the full linearization of L, we transform the system into a heterogeneous system as follows.

(1) The initial set is now the circular class of I, namely the set $^\sim I = \{ab, ba\}$.
(2) The rule $r = \langle a|a{-}b|b\rangle$ is replaced by $^\sim r$; this gives, by conjugacy, one new pure rule $\langle b|b{-}a|a\rangle$ and two concatenation rules $[a{-}b|b{-}a]$ and $[b{-}a|a{-}b]$.

The use of only the concatenation rules produces the set $\{ab, ba, abba, baab\}$. Note that this set is not closed under conjugacy. Then, the repeated application the two pure rules produces the set

$$\{a^nb^{n+m}a^m \mid n + m > 0\} \cup \{b^na^{n+m}b^m \mid n + m > 0\}.$$

This set is now closed under conjugacy; it is the language generated with the four flat rules. Moreover, it is exactly the linearization of the set of circular words $= ^\sim\{a^nb^n \mid n \geq 1\}$ generated by the circular splicing system.

We prove the following result which shows that the example holds in the general case.

Proposition 8.3. *Let $\mathcal{S} = (A, I, R)$ be a circular alphabetic splicing system, and let $\mathcal{S}' = (A, Lin(I), R')$ be the flat heterogeneous splicing system defined by $R' = \bigcup_{r\in R} {}^\sim r$. Then $Lin(\mathcal{C}(\mathcal{S})) = \mathcal{F}(\mathcal{S}')$.*

Proof. We prove first the inclusion $Lin(\mathcal{C}(\mathcal{S})) \subseteq \mathcal{F}(\mathcal{S}')$. The proof is by induction on the minimal number of steps used for generating $^\sim w \in \mathcal{C}(\mathcal{S})$. If the number of step is null, we have $^\sim w \in I$, thus $Lin(^\sim w) \subseteq Lin(I) \subseteq \mathcal{F}(\mathcal{S}')$.

Suppose now that for any word $^\sim w$ of $\mathcal{C}(\mathcal{S})$ generated in at most k steps, we have $Lin(^\sim w) \subseteq \mathcal{F}(\mathcal{S}')$. Let w a word generated in $k + 1$ steps (and not generated in less steps). There exist two circular words $^\sim u$ and $^\sim v$ generated in at most

k steps such that there exist a rule $r = \langle\alpha|\gamma-\delta|\beta\rangle$ in the circular system $\mathcal{S}$, words x, y such that $u \sim \beta x\alpha$ and $v \sim \gamma y\delta$ and such that $w \sim \beta x\alpha\gamma y\delta$. By induction all words in $\tilde{}u$, $\tilde{}v$ are in $\mathcal{F}(\mathcal{S}')$ and we have to show that any word in $\tilde{}w$ is in $\mathcal{F}(\mathcal{S}')$, by the use of the rules in $\tilde{}r$. First, w is obtained, in $\mathcal{S}'$, from $\beta x\alpha$ and $\gamma y\delta$ by the concatenation rule $[\beta-\alpha|\gamma-\delta]$, so $w \in \mathcal{F}(\mathcal{S}')$. Next, if $z \sim w$ and $z \neq w$, then $z = st$ and $w = ts$ for some nonempty words s, t.

If t is a prefix of βx, then there is a factorization $x = x'x''$ such that $t = \beta x'$, $s = x''\alpha\gamma y\delta$. Consequently, $z = x''\alpha\gamma y\delta\beta x'$, showing that z is obtained, in the system $\mathcal{S}'$, from $x''\alpha\beta x'$ and $\gamma y\delta$ by the rule r. Since $x''\alpha\beta x' \sim u$, it follows that $z \in \mathcal{F}(\mathcal{S}')$.

If $t = \beta x\alpha$, then $s = \gamma y\delta$ and $z = \gamma y\delta\beta x\alpha$. In this case, z is obtained by the concatenation rule $[\gamma-\delta|\beta-\alpha]$.

Finally, if $\beta x\alpha\gamma$ is a prefix of t, then there is a factorization $y = y'y''$ such that $t = \beta x\alpha\gamma y'$ and $s = y''\delta$. Consequently, $z = y''\delta\beta x\alpha\gamma y'$, showing that z is obtained from $y''\delta\gamma y'$ and $\beta x\alpha$ by the rule $\langle\delta|\beta-\alpha|\gamma\rangle$. Since $y''\delta\gamma y' \sim v$, it follows again that $z \in \mathcal{F}(\mathcal{S}')$.

Now, we prove the reverse inclusion, that is, $\mathcal{F}(\mathcal{S}') \subseteq Lin(\mathcal{C}(\mathcal{S}))$.

Again, the proof is by induction. The induction is on the minimal number of steps used for generating $w \in \mathcal{F}(\mathcal{S}')$. If the number of step is zero, we have $w \in Lin(I)$, thus $w \subseteq Lin(\mathcal{C}(\mathcal{S}))$.

Suppose now that for any word w of $\mathcal{F}(\mathcal{S}')$ generated in at most k steps, we have $w \subseteq Lin(\mathcal{C}(\mathcal{S}))$. Let w a word generated in $k+1$ steps (and not generated in less steps). There exist two words u and v of $\mathcal{F}(\mathcal{S}')$ and generated by at most k steps, a rule r in R' such that $u, v \vdash_r w$ in the system $\mathcal{S}'$. By induction u and v are in $Lin(\mathcal{C}(\mathcal{S}))$, and than $\tilde{}u$ and $\tilde{}v$ are in $\mathcal{C}(\mathcal{S})$ Moreover, by construction of the set R', there exist a rule $r' = \langle\alpha|\gamma-\delta|\beta\rangle$ in R such that $r \in \tilde{}r'$. We have now four cases.

If r is the rule of concatenation $[\beta-\alpha|\gamma-\delta]$, then there exist words x and y such that $u = \beta x\alpha$, $v = \gamma y\delta$ and $w = \beta x\alpha\gamma y\delta$. we have then $\tilde{}u, \tilde{}v \vdash_{r'} \tilde{}w$ in $\mathcal{S}$.

If r is the rule of concatenation $[\gamma-\delta|\beta-\alpha]$, then there exist words x and y such that $u = \gamma x\delta$, $v = \beta y\alpha$ and $w = \gamma x\delta\beta y\alpha$. we have then $\tilde{}v, \tilde{}u \vdash_{r'} \beta x\alpha\gamma y\delta \sim \tilde{}w$ in $\mathcal{S}$.

If r is the pure rule $\langle\alpha|\gamma-\delta|\beta\rangle$, then there exist words x, y and z such that $u = x\alpha\beta y$, $v = \gamma z\delta$ and $w = x\alpha\gamma z\delta\beta y$. Remark, then that $\tilde{}u \sim \beta yx\alpha$, $v \sim \gamma z\delta$ and $w \sim \beta yx\alpha\gamma z\delta$, and we again have $\tilde{}v, \tilde{}u \vdash_{r'} \tilde{}w$ in $\mathcal{S}$.

The case where r is the pure rule $\langle\delta|\beta-\alpha|\gamma\rangle$, is very similar to the previous one.

In each case, $\tilde{}w$ is produced from $\tilde{}u$ and $\tilde{}v$ by the rule r' in the system $\mathcal{S}$. Subsequently, $\tilde{}w$ is in $\mathcal{C}(\mathcal{S})$, and then w is in $Lin(\mathcal{C}(\mathcal{S}))$.

We have thus proved the inclusion $\mathcal{F}(\mathcal{S}') \subseteq Lin(\mathcal{C}(\mathcal{S}))$. We now can state $Lin(\mathcal{C}(\mathcal{S})) = \mathcal{F}(\mathcal{S}')$. □

The proof of the proposition relies heavily on the fact that the system is alphabetic.

As a consequence of the proposition, we obtain the following theorem, which is our main theorem in the circular case.

Theorem 8.4. *Let $\mathcal{S} = (A, I, R)$ be a circular alphabetic context-free splicing system. Then $Lin(\mathcal{C}(\mathcal{S}))$ is a context-free language.*

Proof. By Proposition 8.3, $Lin(\mathcal{C}(\mathcal{S})) = \mathcal{F}(\mathcal{S}')$, where $\mathcal{S}' = (A, Lin(I), R')$ is the flat heterogeneous splicing system defined by $R' = \bigcup_{r\in R} \tilde{}r$. Since I is context-free, the language $Lin(I)$ is context-free. By Theorem 7.11, the language generated by $\mathcal{S}'$ is context-free. □

Acknowledgments

We are thankful to Olivier Carton for many stimulating discussions during this work. The third author also wishes to thank Clelia de Felice and Rosalba Zizza for inviting her in Salerno, for interesting discussions, and pointing out useful references. We are also thankful to the anonymous referee for his careful reading and his numerous detailed comments which greatly improve the correctness and the presentation of the text.

Appendix. A theorem on generalized context-free grammars

For sake of completeness, we give here a proof of Theorem 6.1, together with an example.

The proof makes use of the substitution theorem that we recall below, along with the notion of context-free substitutions.

Let A and B be two alphabets. A *substitution* from A^* to B^* is a mapping σ from m A^* into subsets of B^* such that $\sigma(\varepsilon) = \{\varepsilon\}$ and

$$\sigma(xy) = \sigma(x)\sigma(y)$$

for all $x, y \in A^*$. The product of the right-hand side is the product of subsets of B^*. The substitution is called *finite* (resp. *regular, context-free, context-sensitive*) if all the languages $\sigma(a)$, for a letter of A, are *finite* (resp. *regular, context-free, context-sensitive*).

It is a well known result (called the substitution theorem, see, for example, [11]) that the language $\sigma(L)$ is context-free if L is a context-free language and σ is a context-free substitution.

The proof of Theorem 6.1 is by induction on the number of non-terminals, this induction is presented through two lemmas. The first deals with the case of generalized context-free grammar with a single non-terminal symbol, and the second shows how to reduce the number of non-terminal symbols in the general case.

Lemma A.1. *Let $G = (A, \{S\}, S, R)$ be a generalized context-free grammar with a single non-terminal symbol S. The language generated by G is context-free.*

Proof. Let L be the language generated by G and set $M_S = \{m \mid S \to m \in R\}$. Let $H = (A \cup \{S\}, V, X, P), S \notin V$, be a usual context-free grammar that generates M_S. The language L is generated by the usual context-free grammar $G' = (A, V \cup \{S\}, S, P \cup \{S \to X\})$. □

Example A.2. Let the grammar G with a single non-terminal symbol $G = (A, \{S\}, S, R)$ with

$$R = \left\{ S \quad \to \quad a \mid Sb^n(c^k d)^n, n \geq 1, k \geq 0 \right\}.$$

According to the sketch of the proof given above, we define a grammar $H = (A \cup \{S\}, \{X, Y, Z\}, X, P)$, with

$$P = \begin{cases} X & \to & a \mid SY \\ Y & \to & bYZ \mid bZ \\ Z & \to & cZ \mid d \end{cases}$$

we can check that H generate the language $M_S = \{a\} \cup \{Sb^n(c^*d)^n \mid n \geq 1\}$. we define now $G' = (A, V \cup \{S\}, S, P')$ with

$$P' = \begin{cases} S & \to & X \\ X & \to & a \mid SY \\ Y & \to & bYZ \mid bZ \\ Z & \to & cZ \mid d. \end{cases}$$

The grammar G' generates the same language as G, that is the language

$$\{ab^{n_1}(c^*d)^{n_1}b^{n_2}(c^*d)^{n_2} \cdots b^{n_k}q(c^*d)^{n_k} \mid k \geq 1, n_i \geq 1, 1 \leq i < k\}.$$

The second lemma below tells us that we can reduce the problem to grammars with a single non-terminal symbol.

Lemma A.3. *Let G be a generalized context-free grammar with at least two non-terminal symbols. There is a generalized context-free grammar with a single non-terminal symbol which generates the same language as G does.*

Proof. Let $G = (A, V, S, R)$ with V of cardinal at least 2. Let $X \in V$, with $X \neq S$. Define a grammar G_X with one non-terminal symbol by $G_X = (A \cup V \setminus \{X\}, \{X\}, X, R_X)$ with $R_X = \{X \to m \mid X \to m \in R\}$. Let M_X the language generated by G_X. The language M_X is context-free by Lemma A.1.

Define the substitution σ_X on $A \cup V$ by

$$\sigma_X(\alpha) = \begin{cases} M_X, & \text{if } \alpha = X \\ \{\alpha\}, & \text{otherwise.} \end{cases}$$

Define now the grammar $H = (A, V \setminus \{X\}, S, P)$ with $P = \{v \to \sigma_X(m) \mid v \to m \in R, v \in V \setminus \{X\}\}$.

The grammar H generates the same language as G. It has one variable less than G and the set of its right-members is still context-free by the substitution theorem.

Now it suffices to iterate the process in order to obtain a grammar with one non-terminal symbol. □

Example A.4. Let $G = (A, V, S, R)$ be the generalized grammar defined by $A = \{a, b, c, d\}$, $V = \{S, T, U\}$, and

$$R = \begin{cases} S & \to & ST \mid a \\ T & \to & b^nU^n, n \geq 1 \\ U & \to & cU \mid d. \end{cases}$$

In the first step, we choose to remove the non-terminal T. Following the sketch of the proof given above, we define a grammar G_T with a single non-terminal symbol T by $G_T = (A \cup \{S, U\}, \{T\}, T, \{T \to b^nU^n, n \geq 1\})$. Let M_T be the language generated by G_T. Clearly, $M_T = \{b^nU^n \mid n \geq 1\}$ and M_T is context-free.

Next, we define a substitution σ_T on $A \cup V$ by

$$\sigma_T(\alpha) = \begin{cases} M_T & \text{if } \alpha = T, \\ \{\alpha\} & \text{otherwise,} \end{cases},$$

and the grammar $H = (A, \{S, U\}, S, P)$ with two non-terminal symbols S, U by $P = \{v \to \sigma_T(m) \mid v \to m \in R, v \in V \setminus \{T\}\}$, i.e.

$$P = \begin{cases} S & \to & a \mid Sb^nU^n, n \geq 1 \\ U & \to & cU \mid d. \end{cases}$$

The grammars H and G generate the same language.

To obtain a grammar with only one variable, we iterate the process by eliminating the variable U from H, and we obtain the grammar $H' = (A, \{S\}, S, P')$ with the unique variable S and with

$$P' = \left\{ S \quad \to \quad a \mid Sb^n(c^*d)^n, n \geq 1. \right\}$$

This is the grammar G of Example A.2 above.

References

[1] Jean Berstel, Transductions and context-free languages, B. G. Teubner, Stuttgart, 1979. Available at http://www-igm.univ-mlv.fr/~berstel/LivreTransductions/LivreTransductions.html.
[2] Paola Bonizzoni, Clelia de Felice, Gabriele Fici, Rosalba Zizza, On the regularity of circular splicing languages: a survey and new developments, Natural Computing 9 (2) (2010) 397–420.
[3] Paola Bonizzoni, Clelia de Felice, Giancarlo Mauri, Rosalba Zizza, DNA and circular splicing, DNA Computing (2000) 117–129.
[4] Paola Bonizzoni, Clelia de Felice, Giancarlo Mauri, Rosalba Zizza, Decision problems for linear and circular splicing systems, Developments in Language Theory (2002) 78–92.
[5] Paola Bonizzoni, Clelia de Felice, Rosalba Zizza, Circular languages generated by complete splicing systems and pure unitary languages, in: S. Barry Cooper, Vincent Danos (Eds.), Fifth Workshop on Developments in Computational Models–Computational Models From Nature, in: Electronic Proceedings in Theoretical Computer Science, vol. 9, 2009, pp. 22–31.
[6] Paola Bonizzoni, Clelia de Felice, Rosalba Zizza, A characterization of (regular) circular languages generated by monotone complete splicing systems, Theoretical Computer Science 411 (48) (2010) 4149–4161.
[7] Rodica Ceterchi, An algebraic characterization of semi-simple splicing, Fundam. Inform. 73 (1–2) (2006) 19–25.
[8] Rodica Ceterchi, Carlos Martín-Vide, K.G. Subramanian, On some classes of splicing languages, in: Natasa Jonoska, Gheorghe Paun, Grzegorz Rozenberg (Eds.), Aspects of Molecular Computing, in: Lecture Notes in Computer Science, vol. 2950, Springer-Verlag, 2004, pp. 84–105. Essays Dedicated to Tom Head on the Occasion of His 70th Birthday.
[9] Clelia de Felice, Gabriele Fici, Rosalba Zizza, A characterization of regular circular languages generated by marked splicing systems, Theoretical Computer Science 410 (47–49) (2009) 4937–4960.
[10] Isabelle Fagnot, Splicing and Chomsky hierarchy, in: Preproceedings Journées Montoises, 8–11 September, Liège, 2004.
[11] Michael A. Harrison, Introduction to Formal Language Theory, Addison-Wesley, 1978.
[12] Tom Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, Bulletin of Mathematical Biology 49 (1987) 737–759.
[13] Tom Head, Splicing schemes and DNA, in: Lindenmayer Systems; Impact on Theoretical Computer Science and Developmental Biology, Springer Verlag, Berlin, 1992, pp. 371–383.
[14] Tom Head, Gheorghe Păun, Dennis Pixton, Language theory and molecular genetics: Generative mechanisms suggested by DNA recombination, in: Handbook of Formal Languages, Vol 2, Springer, Verlag, 1996, pp. 295–360.
[15] Jaroslav Král, A modification of a substitution theorem and some neccessary and sufficient conditions for sets to be context-free, Mathematical Systems Theory 4 (2) (1970) 129–139.
[16] Gheorghe Paun, Grzegorz Rozenberg, Arto Salomaa, DNA computing — new computing paradigms, in: Texts in Theoretical Computer Science, Springer-Verlag, 1998.
[17] Dennis Pixton, Linear and circular splicing systems, in: First International Symposium on Intelligence in Neural and Biological Systems, IEEE, Washington, 1985, pp. 181–188.
[18] Dennis Pixton, Regularity of splicing languages, Discrete Applied Mathematics 69 (1–2) (1996) 101–124.
[19] Rani Siromoney, K.G. Subramanian, V. Rajkumar Dare, Circular DNA and splicing systems, in: ICPIA, 1992, pp. 260–273.
[20] Rosalba Zizza, Splicing systems, Scholarpedia 5 (7) (2010) 9397.