0

Contents

Chapter 1

Introduction

1.1 Synopsis

Operations on languages are intensively studied in formal language theory. One of the main goals of the theory is to represent a family of languages as the closure of some atomic languages with respect to some operations. The theory of abstract families of languages (AFL) deals with operations, many operations appear in formal language theory applications, and so on.

The operations studied so far can be roughly divided into three classes: *set operations* (union, intersection, complementation), *algebraic operations* (morphism, substitution) and purely *language theoretical operations* (catenation, quotient, Kleene closure). In one of the author's papers, [13], some *"arithmetic" operations* like compact, literal and generalized subtraction, multiplication, power and factorial are defined and studied. The so-called subtraction operations turn out to be generalizations of the right and left quotient of languages.

This Thesis continues and enlarges the work in [13] which mainly consisted of the closure properties of the families in the Chomsky hierarchy under the defined operations. Here we study in detail some generalizations of both catenation and quotient together with their variants. The generalizations of catenation will be referred to as *insertion operations* and the generalizations of quotient as *deletion operations*.

As an example, the simplest and most natural generalization of catenation is the *sequential insertion*. Given two words u and v, instead of

catenating v at the right extremity of u, the new operation inserts v in an arbitrary place in u. Notice that the sequential insertion of two words is a finite set of words and their catenation is an element of this set. The operation is extended to languages in the natural fashion.

Besides considering insertion in addition to deletion operations, the main novelty in this work, when compared to [13], is that for each defined operation, a *parallel variant* will be (whenever possible) defined. For example, the *parallel insertion* of a word v into a word u consists of the word obtained by simultaneously inserting v in between all the letters and also at the extremities of u.

The iterated version of catenation is the catenation (Kleene) closure. It is natural to define a counterpart for insertion. The *iterated insertion* proves to be more powerful than the catenation closure. For example, all families in the Chomsky hierarchy are closed under catenation closure but the iterated parallel insertion of a letter into itself already produces a non-context-free language.

A more exotic variant of insertion is obtained if we combine the ordinary insertion with the commutative variant. The commutative variant of a word v is the set consisting of all words obtained by arbitrarily permuting the letters of v. The *permuted insertion* of v into u will then consist of inserting into u all the words from the commutative variant of v.

Observe that even though the above operations generalize the catenation, catenation cannot be obtained as a particular case of any of them. This happens because we cannot force the insertion to take place at the right extremity of the word. This brings up the notion of *controlled insertion*: each letter determines what can be inserted after it. The catenation can be now obtained by using a marker and the controlled insertion next to the marker.

Notice, finally, that all the previously defined types of insertion were "compact". The word to be inserted was treated "as a whole". A "scattered" type of insertion can be considered as well. Instead of inserting a word, we sparsely insert its letters. If the inserted letters are in the same order as in the original, we obtain the well-known *shuffle* operation. Else, the *permuted scattered insertion* is obtained.

For each of the aboved mentioned variants of insertion, the dual deletion operation is also investigated.

Take, for example, the *sequential deletion* which is the dual of the sequential insertion. The sequential deletion is the simplest and most natural generalization of left (right) quotient. The sequential deletion of v from u

consists of erasing v not only from the left (right) extremity of u, but from an arbitrary place in u. If v is not a subword of u the result of the deletion is the empty set. The insertion of a language into another is a *total* operation in the sense that essentially all words from both languages contribute to the result. The deletion of a language from another is a *partial* operation in this sense. The words from the second language which are not subwords of any word in the first, as well as the words from the first language which do not contain any words from the second as subwords, do not contribute to the result.

Some words about the organization of the Thesis.

Section 1.2 contains some basic formal language notions and notations. The section is to be consulted only when need arises.

Section 1.3 contains some background and motivations of the work.

In Chapters 2 and 3, the insertion and deletion operations are formally defined and investigated. These chapters mainly contain closure properties of the families in the Chomsky hierarchy under the above operations. Some other results are presented as well. For example, certain ways in which the operations can be expressed in terms of other classical operations are investigated. Somewhat unexpected results are obtained in connection with the deletion operations. The language obtained by sequentially (resp. iterated sequentially) deleting an arbitrary language from a regular one is always regular. An open problem occurs in connection with the iterated parallel deletion: It is not yet known whether or not the family of regular languages is closed under this operation. Moreover, it is not even known if the same family is closed under iterated parallel deletion with singletons.

Chapter 4 is dedicated to the in-depth study of some operations and their restrictions.

Section 4.1 analyzes the generative power of some operations. We focus on classes of languages which contain the singleton letters, λ and $\emptyset$ and are closed under some of the previously defined operations. The operations are controlled ones and include one insertion operation, one deletion operation and an iterative one. Finally, the mirror image operator and the union with λ are added for technical reasons. The classes thus obtained properly contain the family of regular languages. Moreover, the one whose operations are sequential is included in the family of context-free languages, and the one whose operations are parallel is included in the family of context-sensitive languages and contains non-context-free languages.

Section 4.2 focuses on the particular case where the result of the insertion of two languages is regular. The main theorem of the section states

that if there exists a language L_2 such that the result of the sequential insertion of L_2 into L_1 is a regular language R, then L_2 can be replaced by a regular R'. The language R' can be obtained from R and L_1 by using a deletion operation. Moreover, the regular language R' is the largest language which, when inserted into L_1, gives R, and it can be effectively constructed if L_1 is regular or context-free.

An analogous result holds if the languages L_2 and regular R are given, and there exists L_1 such that the sequential insertion of L_2 into L_1 equals R. Similar theorems are true also for catenation.

Section 4.3 contains an analysis of the particular case of the sequential deletion, where the language to be deleted is a singleton. The operation thus obtained is called the *derivative*. The derivative of a language L by the word w is the sequential deletion of w from L. It is a generalization of the notions of the left and the right derivative. Section 4.3 contains a sufficient condition under which a language gives rise to the same derivative with respect to two different words. Moreover, it is shown that the language consisting of all words, with respect to which a given regular language has the same derivative, is regular. Finally, languages with the maximal number of derivatives are studied. The main result shows that for each integer n there exists a minimal finite deterministic automaton with n states, which recognizes a language with the maximal number of derivatives.

In Section 4.4 the special case of operations where all the operands involved are singletons is focused on. Necessary and sufficient conditions under which the result of sequential insertion or deletion of two words is a singleton set, are given. Also the situation when the insertion and deletion are inverse to each other, which is not generally the case, is studied. Namely, necessary and sufficient conditions under which, after inserting u into w and then deleting u from the result we obtain again w, are given.

Chapter 5 deals with decidability results concerning insertion and deletion. For $\diamond$ denoting an insertion or deletion operation, problems of the following type are considered:

1 Given a regular language R and languages L_1, L_2, is $L_1 \diamond L_2 = R$?

2 Given languages L_1, L_2, is $L_1 \diamond L_2$ regular?

3 Given a regular language R and a language L_1, does there exists a language L_2 such that $L_1 \diamond L_2 = R$?

4 Given a regular language R and a language L_2, does there exist a language L_1 such that $L_1 \diamond L_2 = R$?

Problems of the type 1 and 2 are studied in Section 5.1. In the cases where the problems turn out to be decidable, the closure properties of the family of regular languages under $\diamond$ and the effectiveness of the proofs are invoked. In the other cases, a reduction to well-known undecidable problems is used.

Problems 3 for insertion (resp. deletion) operations are studied in Section 5.2 (resp. 5.4). Problems 4 for insertion (resp. deletion) are investigated in Section 5.3 (resp. 5.5). For the problems 3 and 4 also the existence of a singleton language is investigated. In the cases where the problems turn out to be decidable, the method for showing their decidability is based on theorems similar to the ones proved in Section 4.2. Consider, for example, problem 3 for sequential insertion. The algorithm for deciding it will start with the construction of the language R', mentioned above in the paragraph concerning Section 4.2. As the family of regular languages is closed under sequential deletion, R' is regular and can be effectively constructed. The algorithm then decides whether or not $L_1 \diamond R' = R$ and the answer is the answer to our problem.

For the problems shown to be undecidable, different methods depending on the operation have been applied, in order to reduce the problems to known undecidable ones.

Appendix **A** contains the names of the operations that occur throughout the Thesis, together with their abbreviations and notations.

Appendix **B** presents in a concise form the closure results obtained in Chapters 2, 3 and 4.

Appendix **C** summarizes the decidability results obtained in Chapter 5.

1.2 Basic definitions and notations

The aim of this section is to present the basic formal language notions and notations used in the sequel. For details, the reader is referred to the monographs listed in Bibliography and especially to [12].

For a set S, $\mathrm{card}(S)$ denotes its cardinality. We often identify a singleton $\{v\}$ with its element v. The family of subsets of S is denoted by $\mathcal{P}(S)$ or 2^S.

An *alphabet* is a finite nonempty set; its elements are called *letters* or *symbols*. If $\Sigma = \{a_1, \ldots a_n\}$ is an alphabet then any sequence $w = a_{i_1} \ldots a_{i_k}$, $k \geq 0$, $a_{i_j} \in \Sigma$, $1 \leq j \leq k$ is called a *string (word)* over Σ. The length of the word w is denoted with $\lg(w)$ and, by definition, equals k. The word "consisting" of zero letters is denoted by λ and is called the

empty word. Obviously, $\lg(\lambda) = 0$. The word consisting of repeating the letter a_i j times is abbreviated a_i^j, with the convention $a_i^0 = \lambda$. The set of all words over Σ is denoted Σ^* and the set of all nonempty words, $\Sigma^* - \{\lambda\}$, is denoted Σ^+. For a word u over an alphabet Σ and a letter $a \in \Sigma$, $N_a(u)$ will denote the number of occurrences of the letter a in u.

The set Σ^* is a monoid under the operation of catenation defined by:

$$uv = a_{i_1} \dots a_{i_r} a_{j_1} \dots a_{j_s},$$

$u = a_{i_1} \dots a_{i_r}$, $v = a_{j_1} \dots a_{j_s}$, $r, s \geq 0$, $a_{i_q}, a_{j_p} \in \Sigma$ for $1 \leq q \leq r$, $1 \leq p \leq s$. The fact that r, s can be also zero means that the words u and v can be empty. The catenation operation is associative and λ is the neutral element of the monoid.

A *language* (over Σ) is any subset of Σ^*.

Language operations. Since languages are sets, we may define the set-theoretic operations of union, intersection, difference, and complement in the usual fashion:

$$\begin{aligned}
&L_1 \cup L_2 = \{w|\ w \in L_1 \text{ or } w \in L_2\},\\
&L_1 \cap L_2 = \{w|\ w \in L_1 \text{ and } w \in L_2\},\\
&L_1 - L_2 = \{w|\ w \in L_1 \text{ and } w \notin L_2\},\\
&L^c = \Sigma^* - L,
\end{aligned}$$

where L is over the alphabet Σ.

The *catenation* of two languages L_1 and L_2, denoted $L_1 L_2$, is defined by

$$L_1 L_2 = \{uv|\ u \in L_1, v \in L_2\}.$$

Catenation of languages is associative because catenation of words is associative. Consequently, we may define L^i, $i \geq 1$, to be the language obtained by catenating i copies of L. Furthermore, we define L^0 to be the language $\{\lambda\}$ consisting of the empty word λ. For any language L,

$$L\emptyset = \emptyset L = \emptyset, L\lambda = \lambda L = L.$$

The *catenation closure* (or *iteration*) of a language L is defined to be the union of all the powers of L:

$$L^* = \bigcup_{i=0}^{\infty} L^i.$$

The λ-free catenation closure of L is defined to be the union of all positive powers of L:

$$L^+ = \bigcup_{i=1}^{\infty} L^i.$$

Clearly, L^+ equals either L^* or $L^* - \lambda$, depending on whether $\lambda \in L$ or $\lambda \notin L$. A language that does not contain λ is termed λ-*free*.

The *left quotient* of a language L_1 by a language L_2 is defined by

$$L_2 \backslash L_1 = \{v |\ uv \in L_1 \text{ for some } u \in L_2\}.$$

The *right quotient* is defined similarily:

$$L_1 / L_2 = \{v |\ vu \in L_1 \text{ for some } u \in L_2\}.$$

The *mirror image* of a language is the collection of the mirror images of its words, that is,

$$\mathrm{Mi}(L) = \{\mathrm{Mi}(w) |\ w \in L\},$$

where $\mathrm{Mi}(a_1 a_2 \dots a_n) = a_n \dots a_2 a_1$, $n \geq 0$, $a_i \in \Sigma$, $1 \leq i \leq n$.

We now define the operations of *substitution* and *morphism*. For each letter a of an alphabet Σ, let $\sigma(a)$ be a language over an alphabet Σ_a. Define, furthermore,

$$\sigma(\lambda) = \lambda,\ \sigma(uv) = \sigma(u)\sigma(v) \text{ for } u, v \in \Sigma^*.$$

Such a mapping σ of Σ^* into $2^{\Sigma'^*}$, where Σ' is the union of the alphabets Σ_a is called a *substitution*. For a language L over Σ, we define

$$\sigma(L) = \{w |\ w \in \sigma(u) \text{ for some } u \in L\}.$$

A substitution is *regular* iff each of the languages $\sigma(a)$ is regular. A substitution is λ-*free* iff none of the languages $\sigma(a)$ contains the empty word. A family of languages is *closed under substitution* iff whenever L is in the family and σ is such that each $\sigma(a)$ is in the family, then $\sigma(L)$ is in the family.

A substitution σ such that each $\sigma(a)$ consists of a single word w_a is called a *morphism*. Thus, a morphism is a mapping of Σ^* into Σ'^*. A morphism is λ-*free* iff none of the words w_a is λ.

Assume that L is a language over an alphabet Σ. A morphism h of Σ^* is termed a *k-linear erasing* with respect to L iff for each $w \in L$:

$$\lg(w) \leq k \lg(h(w)).$$

A family $\mathcal{L}$ of languages is said to be *closed under linear erasing* iff $h(L) \in \mathcal{L}$ whenever $L \in \mathcal{L}$, k is an integer and h is a k-linear erasing with respect to L.

A *Chomsky grammar* is an ordered quadruple

$$G = (N, T, S, P),$$

where N and T are disjoint alphabets, $S \in N$ and P is a finite set of ordered pairs (u, v) such that v is a word over the alphabet $N \cup T$ and u is a word over $N \cup T$ containing at least one letter of N. The elements of N are called *nonterminals* and those of T *terminals*; S is called the *initial letter* or the *axiom*. Elements (u, v) of P are called *rewriting rules* or *productions* and are written $u \longrightarrow v$.

In general, the nonterminals will be denoted by upper case letters from the English alphabet, the terminals by lower case letters from the beginning of the English alphabet and the words by lower case letters from the end of the English alphabet or with lower case Greek letters.

The following relation with respect to the grammar G is defined over $(N \cup T)^*$: for $u, v \in (N \cup T)^*$ we write $u \Longrightarrow_G v$ (or simply $u \Longrightarrow v$ if there is no danger of confusion) iff $u = \alpha x \beta$, $v = \alpha y \beta$, $\alpha, \beta \in (N \cup T)^*$ and $x \longrightarrow y \in P$. We say that u *directly derives* v with respect to the grammar G. The reflexive and transitive closure of $\Longrightarrow$ is denoted $\Longrightarrow^*$. The word u *derives* v in the grammar G iff $u \Longrightarrow^* v$.

The *language generated by* G is

$$L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}.$$

The words $v \in (N \cup T)^*$ which satisfy $S \Longrightarrow^* v$ are called *sentential forms*. Two grammars are called *equivalent* iff they generate the same language.

For $i = 0, 1, 2, 3$, a grammar $G = (N, T, S, P)$ is of the *type* i iff the restrictions *(i)* on the set P, as given below, are satisfied:

(0) No restrictions;

(1) Each rule in P is of the form $uAv \longrightarrow uwv$, $u, v, w \in (N \cup T)^*$, $A \in N$, $w \neq \lambda$, with the possible exception of the rule $S \longrightarrow \lambda$ whose occurrence in P implies, however, that S does not occur on the right side of any rule in P;

(2) Each rule in P is of the form $A \longrightarrow w$, $A \in N$, $w \in (N \cup T)^*$;

(3) Each rule of P is of one of the two forms $A \longrightarrow aB$, $A \longrightarrow a$, $A, B \in N$, $a \in T$.

For $i = 0, 1, 2, 3$, a language is of *type* i iff it is generated by a grammar of type i. The type i languages $i = 0, 1, 2, 3$ will be called respectively *recursively enumerable, context-sensitive, context-free, regular* languages. The family of all languages of type i will be denoted respectively RE, CS, CF, REG. The following inclusion chain, called the Chomsky hierarchy, holds:

$$\text{REG} \subset \text{CF} \subset \text{CS} \subset \text{RE},$$

where every inclusion is proper. The families RE, CS, CF, REG are the basic families of the hierarchy.

A grammar is called *length-increasing* iff every production $u \longrightarrow v$ satisfies the condition $\lg(u) \leq \lg(v)$, with the possible exception of the production $S \longrightarrow \lambda$ whose occurrence in P implies, however, that S does not occur on the right side of any production.

Automata. The grammars generate strings, starting from the axiom. The automata recognize strings, the definitional method being thus the inverse one. They start from an arbitrary string and, after processing it, conclude whether or not it belongs to the language defined.

A *finite deterministic automaton* A is a 5-tuple $A = (S, \Sigma, s_0, F, P)$, where:

(i) S is a finite nonempty set of *states*.

(ii) Σ is a finite nonempty set of *inputs*.

(iii) $s_0 \in S$ is the *initial state*.

(iv) $F \subseteq S$ is the set of *final states*.

(v) P is the *set of rules* having the form

$$s_i a \longrightarrow s_j \quad s_i, s_j \in S,\ a \in \Sigma. \qquad (*)$$

Furthermore, for each pair (s_i, a) where $s_i \in S$ and $a \in \Sigma$, there is exactly one production (*) in P.

The following derivation relation with respect to the automaton A is defined over $S\Sigma^*$: for $s, s' \in S$, $a \in \Sigma$, $u \in \Sigma^*$ we write $s\ au \Longrightarrow s'u$ iff $sa \longrightarrow s'$ is a production in P. The reflexive and transitive closure of $\Longrightarrow$ is denoted $\Longrightarrow^*$. The language *accepted* or *recognized* by the automaton A is defined by

$$L(A) = \{w \in \Sigma^* |\ s_0 w \Longrightarrow^* s_1 \text{ for some } s_1 \in F\}.$$

A *finite nondeterministic automaton* is defined as a deterministic one with the exception that in *(v)* the second sentence is omitted. The language accepted by a finite nondeterministic automaton is defined as before.

Since the finite deterministic and nondeterministic automata accept the same languages, that is, the regular languages (see [12], pp.27-29 or [5], pp.22-24), we shall not distinguish between them unless it becomes necessary, but shall simply refer to both as *finite automata*.

A *generalized sequential machine* (shortly, *gsm*) is a 6-tuple

$$g = (\Sigma_1, \Sigma_2, S, s_0, F, P),$$

where

(i) S is a finite nonempty set of *states*.

(ii) Σ_1 is a finite nonempty *input alphabet*.

(iii) Σ_2 is a finite nonempty *output alphabet*.

(iv) $s_0 \in S$ is the *initial state*.

(v) $F \subseteq S$ is the set of *final states*.

(vi) The *productions* in P are of the form

$$s_i a \longrightarrow w s_j,\ s_i, s_j \in S,\ a \in \Sigma_1,\ w \in \Sigma_2^*. \qquad (**)$$

If, in addition, $w \neq \lambda$ in all productions, then we speak of a *λ-free gsm*. A gsm which is not λ-free is sometimes called *gsm with erasing*. For a gsm g and a word u over its input alphabet Σ_1, we denote

$$g(u) = \{w |\ s_0 u \Longrightarrow^* w s' \text{ for some } s' \in F\}.$$

Let L be a language over Σ_1. Then g *translates* or *maps* L into the language

$$g(L) = \{w |\ w \in g(u) \text{ for some } u \in L\}.$$

We say that the mapping thus defined is a *gsm mapping*.

If in the preceding definition the symbol a in relation (**) is allowed to denote also the empty word then the 6-tuple is called a *sequential transducer*. The image of a word and the image of a language through a sequential transducer are defined similarily as for gsm's. Mappings of languages thus defined are referred to as *rational transductions*.

Decidability. A *decision problem* consists of an infinity of instances, to each of which the answer is either "yes" or "no". Such a problem is *decidable* iff there exists an algorithm (effective procedure) which for any instance of the problem outputs the answer for that instance. If no such algorithm exists, the problem is *undecidable*.

Consider a class $\mathcal{G}$ of devices for defining languages. Such a $\mathcal{G}$ might consist, for example, of all grammars of some type. Here are some typical decision problems for a class $\mathcal{G}$ of grammars:

The equivalence problem. Given two grammars G_1, G_2 in $\mathcal{G}$, is $L(G_1) = L(G_2)$?

The emptiness (resp. infinity) problem. Given a grammar G in $\mathcal{G}$, is the language $L(G)$ empty (resp.infinite)?

The inclusion problem. Given two grammars G_1, G_2 in $\mathcal{G}$, is $L(G_1)$ included in $L(G_2)$?

The membership problem. Given a grammar G in $\mathcal{G}$ and a word w, does the word w belong to the language $L(G)$?

1.3 Background: a broader view

The basic notions used for specifying languages are of *algorithmic* or *operational* character: automata (*accepting* devices) or grammars (*generating* devices). This duality reflects the original motivations coming from computer science or linguistics. A deeper theory and more involved proofs called for alternative definitional devices, where the operations have a classical mathematical character. Early examples of such definitional devices are the sequential functions for automata (see [11], pp.48-53) or substitutions and morphisms for grammars.

This Thesis goes into the fundamentals of the substitution operation, aiming thus to throw light on the mechanisms of generating languages. So far in the literature a substitution has been defined as an operation on an

alphabet. A substitution is never applied to λ (except for the convention that λ is always mapped into λ). Our work can be viewed as an attempt to understand the substitution on the empty word.

Let L_1, L_2 be two languages over an alphabet Σ. The operation of *sequential insertion* of a language L_2 into a language L_1, can be viewed as a nonstandard modification of the notion of substitution. It maps all letters of Σ into themselves and the empty letter into L_2, with the following additional convention. For each word w, only one of the empty words occurring in it is substituted. The result of applying this "substitution" to L_1 consists of words obtained from words in L_1 in which an arbitrary word from L_2 has been inserted. Note that the sequential insertion is also a generalization of the catenation operation.

The operation of *parallel insertion* is closer to the classical notion of substitution. It is defined as the substitution above, but with a modified convention: For each word w, between all the letters and also at the extremities, only one λ occurs. The effect of the substitution applied to L_1 will be the insertion of words belonging to L_2 between all letters and also at the extremities of words belonging to L_1.

The exact effect of the classical substitution that maps all letters into themselves and λ into a language L_2 would be the insertion of arbitrary many words of L_2 between letters and at the extremities of words in L_1. According to the definitions mentioned above, this would amount to the parallel insertion of L_2^* into L_1.

Another way to look at operations of controlled insertion is the following. Such an insertion can be viewed as a *production* of the form $a \longrightarrow aw$, where the word w comes from the language to be inserted next to a. The mode of controlled insertion determines how the productions are going to be applied. The controlled *parallel* insertion resembles, thus, the rewriting process of OL systems (see [10]). However, it gives rise to something different from OL systems because the productions are always of the special form and, on the other hand, there may be infinitely many productions. The relation between controlled *sequential* insertion and OS systems (see, for instance, [6]) is similar.

It will be seen in Chapter 5 that some of the decidability issues involved are connected with certain basic questions dealing with the combinatorics of words. In general, we introduce many new types of decision problems when dealing with insertion and deletion. The problems provide a suitable testing ground for many classical proof methods and techniques.

Many of the issues dealt with are also connected with cryptography (see

[1], [2], [9]). The connection of Section 4.4 with cryptography is evident. It studies necessary and sufficient conditions under which the original word w results, after first inserting a word u into w and then deleting u from the result. The cryptographic connotations are obvious: after encrypting the message with a key, the decryption has to provide the original message.

Chapter 2

Sequential and parallel insertion

2.1 Insertion

We will define in the following the operation of *sequential insertion* between words and languages, which is a generalization of the catenation operation. Given two words u and v, instead of catenating v at the right extremity of u, we are allowed to insert it in any place of u. The result of the sequential insertion will be thus a set of words instead of a single word. The cardinality of the set is at most $n+1$, where n is the length of u. The catenation uv is an element of this set, corresponding to the particular case where the insertion is done at the right extremity of u.

Definition 2.1 *Let L_1, L_2 be languages over the alphabet Σ. The sequential insertion (SIN) of L_2 into L_1 is defined as:*

$$L_1 \leftarrow L_2 = \bigcup_{u \in L_1, v \in L_2} (u \leftarrow v)$$

where

$$u \leftarrow v = \{u_1 v u_2 |\ u = u_1 u_2\}.$$

Example 2.1 Let $u = cd$, $v = a$. The sequential insertion of v into u is $u \leftarrow v = \{acd,\ cad,\ cda\}$. Notice that $uv = cda$ is an element of the set $u \leftarrow v$. □

The sequential insertion operation is not associative, as shown by the following example.

Example 2.2 Let us consider the languages $\{a\}, \{b\}, \{c\}$. We have

$$(a \leftarrow b) \leftarrow c = \{ab, ba\} \leftarrow c = \{cab, acb, abc, cba, bca, bac\},$$

whereas

$$a \leftarrow (b \leftarrow c) = a \leftarrow \{cb, bc\} = \{cba, acb, bca, abc\}.$$

□

In general, the following result holds:

Theorem 2.1 *If L_1, L_2, L_3 are languages over an alphabet Σ, then*

$$L_1 \leftarrow (L_2 \leftarrow L_3) \subseteq (L_1 \leftarrow L_2) \leftarrow L_3.$$

Proof. Let α be a word in $L_1 \leftarrow (L_2 \leftarrow L_3)$. There exist words $u \in L_1$, $\beta \in L_2 \leftarrow L_3$ such that $\alpha = u_1 \beta u_2$, $u = u_1 u_2$. This further implies that there exist $v \in L_2$, $w \in L_3$ such that $\beta = v_1 w v_2$, $v = v_1 v_2$. According to the definition of SIN, the word $u_1 v u_2$ belongs to the set $L_1 \leftarrow L_2$. Consequently, α which is an element of the set $u_1 v u_2 \leftarrow w$ belongs to $(L_1 \leftarrow L_2) \leftarrow L_3$. □

The sequential insertion operation is not commutative. For example, $a \leftarrow bc = \{bca, abc\}$, whereas $bc \leftarrow a = \{abc, bac, bca\}$.

A parallel variant of the sequential insertion will be defined in the sequel. The *parallel insertion* of a word v into a word u is the word obtained after inserting v between all the letters of u and at the right and left extremities of u. The definition can be easily transferred to languages. The result of the parallel insertion of a language L into a word u is obtained by simultaneously inserting words from L between all the letters of u, and also at its extremities.

Definition 2.2 *Let L_1, L_2 be languages over the alphabet Σ. The parallel insertion (PIN) of L_2 into L_1 is defined as:*

$$L_1 \Leftarrow L_2 = \bigcup_{u \in L_1} (u \Leftarrow L_2)$$

where

$$\begin{aligned} u \Leftarrow L_2 = \quad & \{v_0 a_1 v_1 a_2 v_2 \dots a_k v_k \mid k \geq 0, a_j \in \Sigma, 1 \leq j \leq k, \\ & v_i \in L_2, 0 \leq i \leq k \text{ and } u = a_1 a_2 \dots a_k\}. \end{aligned}$$

The case $k = 0$ corresponds to the situation $u = \lambda$, when only one word $v_0 \in L_2$ is inserted.

Example 2.3 Let $L_1 = \{cd\ \},\ L_2 = \{a, b\}$. The parallel insertion of L_2 into L_1 is:

$$L_1 \Leftarrow\!\!= L_2 = \{acada, acadb, acbda, acbdb, bcada, bcadb, bcbda, bcbdb\}.$$

□

Theorem 2.2 *The parallel insertion operation induces a monoid structure on* $\mathcal{P}(\Sigma^*)$.

Proof. The parallel insertion is an associative operation. The proof of this fact being straightforward but long, we will consider in the following only the singleton case. Let u, v, w be words in Σ^*. Assume that u and v are of the form:

$$\begin{array}{llll} u = a_1 a_2 \dots a_n, & n \geq 0, & a_i \in \Sigma, & 1 \leq i \leq n, \\ v = b_1 b_2 \dots b_m, & m \geq 0, & b_j \in \Sigma, & 1 \leq j \leq m. \end{array}$$

Then, the following equalities hold:

$$\begin{array}{l} (u \Leftarrow\!\!= v) \Leftarrow\!\!= w = (b_1 \dots b_m a_1 b_1 \dots b_m \dots\dots b_1 \dots b_m a_n b_1 \dots b_m) \Leftarrow\!\!= w = \\ w b_1 w \dots w b_m w a_1 w b_1 w \dots w b_m w \dots\dots w b_1 w \dots w b_m w a_n w b_1 w \dots w b_m w \\ = (v \Leftarrow\!\!= w) a_1 (v \Leftarrow\!\!= w) \dots (v \Leftarrow\!\!= w) a_n (v \Leftarrow\!\!= w) = u \Leftarrow\!\!= (v \Leftarrow\!\!= w). \end{array}$$

The neutral element of the monoid is $\{\lambda\}$. Indeed, for any language L we have $\lambda \Leftarrow\!\!= L = L \Leftarrow\!\!= \lambda = L$.

□

The monoid $(\mathcal{P}(\Sigma^*), \Leftarrow\!\!=)$ is not commutative. For example, $a \Leftarrow\!\!= b = \{bab\}$ whereas $b \Leftarrow\!\!= a = \{aba\}$.

We establish in the following the closure properties of the families in the Chomsky hierarchy under sequential and parallel insertion.

Theorem 2.3 *The families of regular, context-free and context-sensitive languages are closed under sequential insertion.*

Proof. Let Σ be an alphabet and let $\#$ be a letter which does not occur in Σ. Consider the λ-free gsm:

$$\begin{array}{ll} g = & (\Sigma, \Sigma \cup \{\#\}, \{s_0, s\}, s_0, \{s\}, P), \\ P = & \{s_0 a \longrightarrow a s_0 |\ a \in \Sigma\} \cup \{s_0 a \longrightarrow a \# s |\ a \in \Sigma\} \cup \\ & \{sa \longrightarrow as |\ a \in \Sigma\} \cup \{s_0 a \longrightarrow \# as |\ a \in \Sigma\}. \end{array}$$

Claim. For any language $L \subseteq \Sigma^+$ we have $g(L) = L \leftarrow \{\#\}$.

Intuitively, the gsm g works as follows: given a nonempty word u as an input, g leaves its letters unchanged, but inserts a marker $\#$ in an arbitrary position in u.

"$\subseteq$" Let v be a word in $g(L)$. There exist a word u in L and a derivation $s_0 u \Longrightarrow^* vs$ according to the rules of P. During the derivation only one marker-producing rule can be applied. Depending on whether this rule is of the type $s_0 a \longrightarrow a \# s$ or $s_0 a \longrightarrow \# as$, $a \in \Sigma$, the derivation $s_0 u \Longrightarrow^* vs$ will have the form:

$$\begin{array}{ll} s_0 u = & s_0 u_1 a u_2 \Longrightarrow^* u_1 s_0 a u_2 \Longrightarrow \\ & u_1 a \# s u_2 \Longrightarrow^* u_1 a \# u_2 s = vs, \\ & \text{respectively} \\ s_0 u = & s_0 u_1 a u_2 \Longrightarrow^* u_1 s_0 a u_2 \Longrightarrow \\ & u_1 \# as u_2 \Longrightarrow^* u_1 \# a u_2 s = vs, \end{array}$$

where $a \in \Sigma$, $u_1, u_2 \in \Sigma^*$ and $u = u_1 a u_2 \in L$. For scanning u_1 only rules of the type $s_0 b \longrightarrow b s_0$, $b \in \Sigma$, have been used. These rules are not needed if $u_1 = \lambda$. Afterwards, the rule $s_0 a \longrightarrow a \# s$ (respectively the rule $s_0 a \longrightarrow \# as$) has been applied. Finally, for scanning u_2, rules of the type $sb \longrightarrow bs$, $b \in \Sigma$, have been used. They are not needed if $u_2 = \lambda$. According to the definition of the SIN, in both cases the word v belongs to the set $(u_1 a u_2 \leftarrow \{\#\}) \subseteq L \leftarrow \{\#\}$. We conclude that $g(L) \subseteq L \leftarrow \{\#\}$.

"$\supseteq$" Let v be a word in $L \leftarrow \{\#\}$. Then v is of the form $v = u_1 \# u_2$ where $u = u_1 u_2 \in L$.

If $u_1 = \lambda$ then $u_2 \neq \lambda$ and we can construct the derivation:

$$s_0 u_2 = s_0 a u_2' \Longrightarrow \# as u_2' \Longrightarrow^* \# a u_2' s = vs,$$

where $a \in \Sigma, u_2' \in \Sigma^*$. After applying the rule $s_0 a \longrightarrow \# as$, rules of the type $sb \longrightarrow bs$, $b \in \Sigma$, have been used. If $u_2' = \lambda$, such rules are not needed. The existence of the above derivation shows that $v \in g(u) \subseteq g(L)$.

If $u_1 \neq \lambda$ we can construct the derivation:

$$s_0u_1u_2 = s_0u_1'au_2 \Longrightarrow^* u_1's_0au_2 \Longrightarrow^* u_1'a\#su_2 \Longrightarrow^* u_1\#u_2s = vs,$$

where $a \in \Sigma$, $u_1' \in \Sigma^*$. For scanning u_1' rules $s_0b \longrightarrow bs_0$, $b \in \Sigma$, have been used. They do not appear in the derivation if u_1' is empty. After the application of the rule $s_0a \longrightarrow a\#s$, only rules of the type $sb \longrightarrow bs$, $b \in \Sigma$, have been used. They are not needed if $u_2 = \lambda$. The derivation shows that $v \in g(u) \subseteq g(L)$.

Since we have shown in both cases that v belongs to $g(L)$, the proof of the second inclusion and therefore of the claim is complete.

Return to the proof of the theorem. Let $L_1, L_2 \subseteq \Sigma^*$ be two languages belonging to one of the families REG, CF, CS. Consider the λ-free substitution:

$$s' : (\Sigma \cup \{\#\})^* \longrightarrow 2^{\Sigma^*}, s'(\#) = L_2 - \{\lambda\}, s'(a) = a, \forall a \in \Sigma.$$

Using the above Claim it is easy to prove that the following relations hold:

$$\begin{array}{lll} L_1 \leftarrow L_2 = & s'(g(L_1)) \cup L_1 \cup L_2, & \text{if } \lambda \in L_1 \cap L_2, \\ L_1 \leftarrow L_2 = & s'(g(L_1)) \cup L_1, & \text{if } \lambda \in L_2 - L_1, \\ L_1 \leftarrow L_2 = & s'(g(L_1)) \cup L_2, & \text{if } \lambda \in L_1 - L_2, \\ L_1 \leftarrow L_2 = & s'(g(L_1)), & \text{if } \lambda \notin L_1 \cup L_2. \end{array}$$

The families REG, CF, CS are closed under λ-free gsm mappings, λ-free substitutions and union. Consequently, they are closed also under sequential insertion. □

Note. We have proved, in fact, a stronger result than the one stated in Theorem 2.3: Any family of languages closed under λ-free gsm mappings, λ-free substitutions and union is closed under sequential insertion. However, the statement of Theorem 2.3 was preferred, as we are concerned here only with the closure of the Chomsky families under the defined operations.

This will often be the case in Chapters 2 and 3: even though stronger results are actually proved, the theorems state only the closure properties of the Chomsky families under the defined operations.

Theorem 2.4 *The families of regular, context-free and context-sensitive languages are closed under parallel insertion.*

Proof. Let L_1, L_2 be languages over the alphabet Σ. Then, $L_1 \Leftarrow\!\!= L_2 = L_2 s(L_1)$, where s is the λ-free substitution defined by:

$$s : \Sigma^* \longrightarrow 2^{\Sigma^*},\ s(a) = aL_2, \text{for every } a \in \Sigma.$$

The theorem now follows from the closure of the families of regular, context-free and context-sensitive languages under catenation and λ-free substitution. □

2.2 Iterated insertion

In the same way as the sequential and parallel insertion are generalizations of the catenation operation, the *iterated sequential* and *parallel insertion* are generalizations of the catenation closure operation. However, the iterated SIN and PIN prove to be more powerful than the iterated catenation. Indeed, the family of regular (context-free) languages is closed under catenation closure, whereas it is not closed under iterated SIN (iterated PIN) of a language into itself.

Definition 2.3 *Let L_1, L_2 be languages over the alphabet Σ. The insertion of order n of L_2 into L_1 is inductively defined by the equations:*

$$\begin{aligned} L_1 \leftarrow^0 L_2 &= L_1, \\ L_1 \leftarrow^{i+1} L_2 &= (L_1 \leftarrow^i L_2) \leftarrow L_2, i \geq 0. \end{aligned}$$

The iterated sequential insertion (iterated SIN) of L_2 into L_1 is then defined as:

$$L_1 \leftarrow^* L_2 = \bigcup_{n=0}^{\infty} (L_1 \leftarrow^n L_2).$$

Example 2.4 Let $L_1 = \{\lambda, ()\}$. The result of iterated SIN of L_1 into itself will be the *Dyck language* of order one. □

The iterated SIN is not a commutative operation. For example, $\lambda \leftarrow^* b = b^*$ whereas $b \leftarrow^* \lambda = b$.

The iterated SIN is not an associative operation. In general, for L_1, L_2, L_3 arbitrary languages over an alphabet Σ, the sets $L_1 \leftarrow^* (L_2 \leftarrow^* L_3)$ and $(L_1 \leftarrow^* L_2) \leftarrow^* L_3$ are incomparable. This is illustrated by the following two examples.

Example 2.5 Let $L_1 = \{a\}$, $L_2 = \{b\}$ and $L_3 = \{cd\}$. The word $abcbd$ belongs to the set $a \leftarrow^* (b \leftarrow^* cd)$ but not to the set $(a \leftarrow^* b) \leftarrow^* cd$. □

Example 2.6 Let $L_1 = \{ab\}$, $L_2 = \{cd\}$, $L_3 = \{f\}$. The word $afbcd$ belongs to the set $(ab \leftarrow^* cd) \leftarrow^* f$, but does not belong to the set $ab \leftarrow^* (cd \leftarrow^* f)$. □

The *iterated parallel insertion* of a language L_2 into a language L_1 is defined by replacing in the previous definition the sequential insertion "$\leftarrow$" with the parallel insertion "$\Leftarrow$".

Example 2.7 If b is a singleton letter, the result of iterated PIN of b into itself is $b \Leftarrow^* b = \{b^{2^k-1} \mid k > 0\}$. □

The example used to show the noncommutativity of the iterated SIN can be used to prove that the iterated PIN is not a commutative operation. Indeed, the word bbb belongs to $\lambda \Leftarrow^* b$ but not to $b \Leftarrow^* \lambda$.

The iterated PIN is not associative. Moreover, for L_1, L_2, L_3 arbitrary languages over an alphabet Σ, the sets $L_1 \Leftarrow^* (L_2 \Leftarrow^* L_3)$, $(L_1 \Leftarrow^* L_2) \Leftarrow^* L_3$ are incomparable. This is shown by the following example.

Example 2.8 Let $L_1 = \{a\}$, $L_2 = \{b\}$, $L_3 = \{c\}$. The word $cbcab$ belongs to the set $a \Leftarrow^* (b \Leftarrow^* c)$ but not to the set $(a \Leftarrow^* b) \Leftarrow^* c$. On the other hand, the word cac belongs to $(a \Leftarrow^* b) \Leftarrow^* c$ but not to $a \Leftarrow^* (b \Leftarrow^* c)$. □

The folllowing results show the "range" of the iterated SIN operation: it can produce non-regular languages starting from finite ones, but CF and CS are still closed under it.

Theorem 2.5 *There exist a finite language L_1 and a word w such that $L_1 \leftarrow^* w$ is not a regular language.*

Proof. The iterated SIN of $\{()\}$ into $\{\lambda, ()\}$ is the Dyck language of order one, which is a non-regular context-free language. □

Corollary 2.1 *The family of regular languages is not closed under iterated sequential insertion.*

The following lemma proves a fact already exemplified by the languages in Example 2.4 and Theorem 2.5. Adding the empty word to a λ-free language to be inserted does not change the result of the iterated SIN.

Lemma 2.1 *If L_1, L_2 are languages over an alphabet Σ then:*

$$L_1 \leftarrow^* L_2 = L_1 \leftarrow^* (L_2 - \{\lambda\}).$$

Proof. The inclusion " $\supseteq$ " is obvious.

For the reverse inclusion it can be proved by induction on n that:

$$L_1 \leftarrow^n L_2 \subseteq L_1 \leftarrow^* (L_2 - \{\lambda\}).$$

$n = 0$. $L_1 \subseteq L_1 \leftarrow^* (L_2 - \{\lambda\})$.

$n \mapsto (n+1)$. Assume that the inclusion holds for numbers up to n and show that:

$$(L_1 \leftarrow^n L_2) \leftarrow L_2 \ \subseteq \ L_1 \leftarrow^* (L_2 - \{\lambda\}).$$

Using the induction hypothesis, all that remains to be shown is that:

$$[L_1 \leftarrow^* (L_2 - \{\lambda\})] \leftarrow L_2 \subseteq \ L_1 \leftarrow^* (L_2 - \{\lambda\}),$$

which holds because:

$[L_1 \leftarrow^* (L_2 - \{\lambda\})] \leftarrow L_2 =$
$\{[L_1 \leftarrow^* (L_2 - \{\lambda\})] \leftarrow (L_2 - \{\lambda\})\} \cup [(L_1 \leftarrow^* (L_2 - \{\lambda\})) \leftarrow \{\lambda\}] \ \subseteq$
$L_1 \leftarrow^* (L_2 - \{\lambda\})$.

Hence, both inclusions hold. □

Theorem 2.6 *The family of context-free languages is closed under iterated sequential insertion.*

Proof. Let L_1, L_2 be languages generated by the context-free grammars $G_i = (N_i, \Sigma_i, S_i, P_i),\ i = 1, 2$. We can assume, without loss of generality, that $N_1 \cap N_2 = \emptyset$.

Assume further that $G_i,\ i = 1, 2$, satisfy the following properties (see, for example [12], pp.55-56):

- S_i does not appear on the right side of any production of P_i ;
- if $\lambda \in L(G_i)$, the only production of P_i with λ as the right side is $S_i \longrightarrow \lambda$;
- all the productions of P_i (except eventually $S_i \longrightarrow \lambda$) are of the form $A \longrightarrow BC,\ A \longrightarrow a,\ A, B, C \in N_i,\ a \in \Sigma_i$.

According to the previous lemma, we can assume that L_2 is λ-free, that is, P_2 does not contain the rule $S_2 \longrightarrow \lambda$.

Let $G = (N, \Sigma, S, P)$ be the context-free grammar whose components are:

$$\begin{array}{rcl} N & = & N_1 \cup N_2, \\ \Sigma & = & \Sigma_1 \cup \Sigma_2, \\ S & = & S_1, \\ P & = & P_1 \cup P_2 \cup \{S_1 \longrightarrow S_2 |\ S_1 \longrightarrow \lambda \in P_1\} \cup \\ & & \{A \longrightarrow S_2 a, A \longrightarrow a S_2 |\ A \longrightarrow a \in P_1 \cup P_2\}. \end{array}$$

Claim. $L_1 \leftarrow\!\!\!\!\leftarrow^* L_2 = L(G)$.

" $\subseteq$ " We will prove by induction on n that $L_1 \leftarrow\!\!\!\!\leftarrow^n L_2 \subseteq L(G)$.

$n = 0$. Obvious, because $L_1 \leftarrow\!\!\!\!\leftarrow^0 L_2 = L_1$ and $S = S_1$, $P_1 \subseteq P$.

$n \mapsto (n+1)$. Assume the claim true for n and let α be a word in $L_1 \leftarrow\!\!\!\!\leftarrow^{(n+1)} L_2 = (L_1 \leftarrow\!\!\!\!\leftarrow^n L_2) \leftarrow\!\!\!\!\leftarrow L_2$. There exist words $uv \in L_1 \leftarrow\!\!\!\!\leftarrow^n L_2$ and $w \in L_2$ such that $\alpha = uwv$. According to the induction hypothesis, $uv \in L(G)$, therefore a derivation $S_1 \Longrightarrow^* uv$ exists in G.

Because of the properties satisfied by G we can assume –without loss of generality– that during the derivation, first all the nonterminal and afterwards all the terminal rules have been applied. Depending on which of the words u, v is empty, the derivation has one of the following forms:

(i) $S_1 \Longrightarrow^* X_1 X_2 \ldots X_k Y_1 Y_2 \ldots Y_m \Longrightarrow^* a_1 a_2 \ldots a_k b_1 b_2 \ldots b_m$,
where $u = a_1 a_2 \ldots a_k$ and $v = b_1 b_2 \ldots b_m$, $k \geq 1$, $m \geq 0$ (v can be also the empty word), a_i, b_j are terminals and X_i, Y_j are nonterminals for $1 \leq i \leq k$, $1 \leq j \leq m$.

We can construct the derivation, according to G:

$$S_1 \Longrightarrow^* X_1 X_2 \ldots X_k Y_1 Y_2 \ldots Y_m \Longrightarrow^* a_1 a_2 \ldots a_{k-1} X_k b_1 b_2 \ldots b_m \Longrightarrow^*$$

$$a_1 a_2 \ldots a_{k-1} a_k S_2 b_1 b_2 \ldots b_m \Longrightarrow^* uwv = \alpha,$$

where we have used the derivations $S_1 \Longrightarrow^* uv$ and $S_2 \Longrightarrow^* w$, replacing in the first one the step $X_k \longrightarrow a_k$ with $X_k \longrightarrow a_k S_2$. The existence of this derivation assures us that $\alpha \in L(G)$.

(ii) $S_1 \Longrightarrow Y_1 Y_2 \ldots Y_m \Longrightarrow^* b_1 b_2 \ldots b_m$, where $u = \lambda, m \geq 1, v = b_1 \ldots b_m$, Y_j are nonterminals and b_j are terminals for $1 \leq j \leq m$. In this case we can construct:

$$\begin{array}{rcl} S_1 & \Longrightarrow^* & Y_1 Y_2 \ldots Y_m \Longrightarrow^* Y_1 b_2 \ldots b_m \Longrightarrow \\ & & S_2 b_1 \ldots b_m \Longrightarrow^* w b_1 b_2 \ldots b_m = \alpha. \end{array}$$

obtained from the previous derivation by replacing the rule $Y_1 \longrightarrow b_1$ with $Y_1 \longrightarrow S_2 b_1$ and adding the derivation $S_2 \Longrightarrow^* w$. This shows that $\alpha \in L(G)$.

$(iii) S_1 \Longrightarrow^* \lambda$, where $uv = \lambda$, that implies $\lambda \in L_1$. The derivation

$$S_1 \Longrightarrow S_2 \Longrightarrow^* w = \alpha$$

where we have used the rule $S_1 \longrightarrow S_2$ which is in P because $\lambda \in L_1$, shows that $\alpha \in L(G)$.

The inductive step and therefore the inclusion has been proved.

" $\supseteq$ " Let u be a word in $L(G)$. We will show, by induction on n that $u \in L_1 \longleftarrow^* L_2$, where n is the number of rules that contain S_2 on their right side, used in the derivation of u.

$n = 0$. If no such rule has been used, $u \in L_1 \subseteq L_1 \longleftarrow^* L_2$.

$n \mapsto (n+1)$. Assume the statement true for values up to n, and let u be a word such that $S_1 \Longrightarrow^* u$ in G and $(n+1)$ rules containing S_2 on their right side have been used in the derivation.

Let us separate the last occurrence of such a rule.

If the rule is $S_1 \longrightarrow S_2$, the derivation is $S_1 \Longrightarrow S_2 \Longrightarrow^* u$, with $u \in L_2$ and $S_1 \longrightarrow \lambda \in P_1$. Because $\lambda \in L_1$, $L_2 \subseteq L_1 \longleftarrow^* L_2$ and therefore $u \in L_1 \longleftarrow^* L_2$.

If the rule is $A \longrightarrow aS_2$, then the derivation is:

$$\begin{array}{lll} S_1 & \Longrightarrow^* & u_1 A u_2 \Longrightarrow u_1 a S_2 u_2 \Longrightarrow^* \\ & & u_1' a v u_2' = u, \end{array}$$

where $S_2 \Longrightarrow^* v$ in G_2. By replacing the rule $A \longrightarrow aS_2$ with $A \longrightarrow a$ and applying the induction hypothesis, we deduce that $u_1' a u_2'$ belongs to $L_1 \longleftarrow^* L_2$. That implies further $u \in (\{u_1' a u_2'\} \longleftarrow \{v\})$, that is $u \in (L_1 \longleftarrow^* L_2) \longleftarrow L_2 \subseteq L_1 \longleftarrow^* L_2$.

If the rule is $A \longrightarrow S_2 a$, the proof is similar with the preceding.

In all cases we have obtained that $u \in L_1 \longleftarrow^* L_2$, which proves the second inclusion and therefore the claim. The closure of the family of context-free languages under iterated sequential insertion follows as $L_1 \longleftarrow^* L_2$ is generated by the context-free grammar G. □

Theorem 2.7 *The family of context-sensitive languages is closed under iterated sequential insertion.*

Proof. Let $L_1 = L(G_1)$, $L_2 = L(G_2)$ be two languages generated by the context-sensitive grammars $G_i = (N_i, \Sigma_i, S_i, P_i)$, $i = 1, 2$. Assume that $N_1 \cap N_2 = \emptyset$. Assume further that G_i , $i = 1, 2$ satisfy the conditions (see, for example, [12], pp.19-20):

- S_i does not occur on the right side of any production of P_i ;
- if $\lambda \in L(G_i)$ then the only rule which has the right side equal with λ is $S_i \longrightarrow \lambda$;
- every rule of P_i containing a terminal letter is of the form $A \longrightarrow a$ where $A \in N_i$ and $a \in \Sigma_i$.

According to Lemma 2.1 we can also assume that λ does not belong to L_2.

Let # be a new symbol which does not occur in any of the considered alphabets. Consider the context-sensitive grammar $G = (N, \Sigma, S_1, P)$ whose components are:

$$\begin{array}{ll} N = & N_1 \cup N_2, \\ \Sigma = & \Sigma_1 \cup \Sigma_2 \cup \{\#\}, \\ P = & P_1 \cup P_2 \cup \{S_1 \longrightarrow S_2 |\ S_1 \longrightarrow \lambda \in P_1\} \cup \\ & \{A \longrightarrow \#S_2\#a, A \longrightarrow a\#S_2\# |\ A \longrightarrow a \in P_1 \cup P_2\}. \end{array}$$

Define now the morphism $h : \Sigma^* \longrightarrow (\Sigma_1 \cup \Sigma_2)^*$ by $h(\#) = \lambda$, $h(a) = a$, $\forall a \in \Sigma_1 \cup \Sigma_2$. The role of the morphism h being obvious, it can be proved as in the preceding theorem, that $h(L(G)) = L_1 \leftarrow^* L_2$.

From the form of the rules of P we notice that the number of markers # in a word $u \in L(G)$ depends on the number of terminals. More precisely, we have $N_\#(u) \leq 2 \times \lg(h(u))$, for every word $u \in L(G)$. Consequently, $\lg(u) \leq 3 \times \lg(h(u))$, $\forall u \in L(G)$, that is, h is a 3-linear erasing with respect to $L(G)$.

The theorem now follows as the family of context-sensitive languages is closed under linear erasing.

□

The next result shows that the iterated PIN is more powerful than the iterated SIN: starting only with two one-letter words, the iterated PIN can produce a non-context-free language. However the family of context-sensitive languages is still closed under iterated PIN.

Theorem 2.8 *There exists a singleton language L such that $L \Leftarrow^* L$ is not a context-free language.*

Proof. See Example 2.7. □

Corollary 2.2 *The families of regular and context-free languages are not closed under iterated parallel insertion.*

Theorem 2.9 *The family of context-sensitive languages is closed under iterated parallel insertion.*

Proof. Let L_1, L_2 be languages generated by the context-sensitive grammars $G_i = (N_i, \Sigma_i, S_i, P_i)$, $i = 1, 2$. Assume that the grammars satisfy the conditions stated in Theorem 2.7. Assume further that λ does not belong to L_2.

Let $G = (N, \Sigma, S, P)$ be the context-sensitive grammar whose components are:

$$\begin{array}{rcl} N & = & N_1 \cup N_2 \cup \{S, X\}, \\ \Sigma & = & \Sigma_1 \cup \Sigma_2 \cup \{\$, \#\}, \\ P & = & (P_1 - \{S_1 \longrightarrow \lambda\}) \cup P_2 \cup \\ & & \{S \longrightarrow XS, S \longrightarrow \$S_1\#\} \cup \\ & & \{X\$ \longrightarrow \$S_2X\} \cup \{X\$\# \longrightarrow \$S_2\#|\ S_1 \longrightarrow \lambda \in P_1\} \cup \\ & & \{Xa \longrightarrow aS_2X|\ a \in \Sigma_1 \cup \Sigma_2\} \cup \\ & & \{Xa\# \longrightarrow aS_2\#|\ a \in \Sigma_1 \cup \Sigma_2\} \cup \\ & & \{S \longrightarrow \$\#|\ S_1 \longrightarrow \lambda \in P_1\}, \end{array}$$

where $S, X, \$, \#$, are new symbols which do not occur in any of the given vocabularies. Intuitively, the grammar works as follows. X^n represents the number of iterations, \$ and # are markers. After a sentential form of the type $X^n\$u\#$, $u \in L_1$ is reached, X starts to move to the right, producing an S_2 at the left extremity of u, and after every letter in u. When it reaches the right extermity of the sentential form, X dissapears. The introduced S_2's produce words of L_2. Thus, a word from $L_1 \Leftarrow\!\!= {}^* L_2$ is obtained. After n iterations of the process, we obtain a word in $\$(L_1 \Leftarrow\!\!= {}^n L_2)\#$.

Claim. $\$(L_1 \Leftarrow\!\!= {}^* L_2)\# \ = \ L(G)$.

" $\subseteq$ " We will show, by induction on n, that:

$$\$(L_1 \Leftarrow\!\!= {}^n L_2)\# \ \subseteq \ L(G).$$

$n = 0$. Let $\$u\# \in \$L_1\#$. We can construct the derivations according to G:

$$S \Longrightarrow \$S_1\# \Longrightarrow {}^* \$u\#, \text{ if } u \neq \lambda,$$

or

$$S \Longrightarrow \$\#, \text{ if } u = \lambda,$$

which prove that $\$u\# \in L(G)$.

$n \mapsto (n+1)$. Assume the claim true for n and let $\$u\#$ be a word in $\$(L_1 \Leftarrow^{n+1} L_2)\# = \$((L_1 \Leftarrow^n L_2) \Leftarrow L_2)\#$. According to the definition of $\Leftarrow^n$, there exist words $w \in L_1 \Leftarrow^n L_2$, $v_i \in L_2$ such that:

$$u = v_0 a_1 v_1 \dots a_k v_k, w = a_1 a_2 \dots a_k, k \geq 0,$$
$$a_j \in \Sigma_1 \cup \Sigma_2, 1 \leq j \leq k, v_i \in L_2, 0 \leq i \leq k.$$

Taking into account the induction hypothesis, the word $\$w\#$ belongs to $L(G)$, therefore there exists a derivation $S \Longrightarrow^* \$w\#$ in G. We can construct now either the derivation:

$$\begin{aligned} S \quad \Longrightarrow \quad & XS \Longrightarrow^* X\$a_1 a_2 \dots a_k \# \Longrightarrow^* \\ & \$S_2 a_1 S_2 \dots a_k S_2 \# \Longrightarrow^* \\ & \$v_0 a_1 v_1 \dots a_k v_k \# = \$u\#, \end{aligned}$$

if $w \neq \lambda$, or the derivation:

$$S \Longrightarrow XS \Longrightarrow X\$\# \Longrightarrow \$S_2 \# \Longrightarrow^* \$v_0 \# = \$u\#$$

if $w = \lambda$, which prove that $\$u\# \in L(G)$.

" $\supseteq$ " Let $\$u\#$ be a word in $L(G)$, and $S \Longrightarrow^* \$u\#$ a derivation that produces it.

We will show by induction on n that $\$u\#$ is in $\$(L_1 \Leftarrow^* L_2)\#$, where n is the number of applications of the rule $S \longrightarrow XS$ during the derivation.

$n = 0$. If this rule has not been applied, the derivation has one of the folowing forms:

- $S \Longrightarrow \$\#$, where $u = \lambda$ and $S_1 \longrightarrow \lambda \in P_1$;
- $S \Longrightarrow \$S_1\# \Longrightarrow^* \$u\#$, where $u \in L_1$.

In both cases we have that $\$u\# \in \$L_1\# \subseteq \$(L_1 \Leftarrow^* L_2)\#$.

$n \mapsto (n+1)$. Assume the statement true for n and let $\$u\#$ be a word in $L(G)$ such that $(n+1)$ rules $S \longrightarrow XS$ have been applied during the derivation $S \Longrightarrow^* \$u\#$. The first rule used during this derivation is necessarily $S \longrightarrow XS$.

If the next rule to be used is $S \longrightarrow \$\#$, then the derivation is:

$$S \Longrightarrow XS \Longrightarrow X\$\# \Longrightarrow \$S_2\# \Longrightarrow^* \$u\#, \ u \in L_2.$$

The existence of $S \longrightarrow \$\#$ in P implies that $\lambda \in L_1$ and therefore $\$u\# \in \$(L_1 \Leftarrow L_2)\# \subseteq \$(L_1 \Leftarrow^* L_2)\#$.

Else, the derivation continues as explained in the following. The nonterminal S can produce only sentential forms whose left extremity is \$ or X. As our X can be rewritten only if it is followed by a terminal in $\Sigma_1 \cup \Sigma_2$ or \$, we deduce that the derivation has the form:

$$S \Longrightarrow XS \Longrightarrow^* X\$u'\# \Longrightarrow^* \$u\#,\ u' \in (N_1 \cup N_2 \cup \{X\} \cup \Sigma_1 \cup \Sigma_2)^+.$$

Because, after crossing \$, X can jump only over terminal letters of $\Sigma_1 \cup \Sigma_2$, we can assume that $u' \in (\Sigma_1 \cup \Sigma_2)^+$. This further implies that $S \Longrightarrow^* \$u'\#$ is a terminal derivation in $L(G)$, and we can apply now the induction hypothesis, which assures that $u' \in L_1 \Leftarrow^* L_2$. The only way the derivation

$$X\$u'\# \Longrightarrow^* \$u\#$$

can develop is that X produces an S_2 when crossing every terminal letter, and when it reaches the end marker # it disappears. In turn, every S_2 produces a word of L_2. The terminal letters act as separators, preventing illegitimate contexts. We can conclude that:

$$\begin{array}{rcl} S & \Longrightarrow & XS \Longrightarrow^* X\$u'\# = X\$a_1a_2 \ldots a_k\# \Longrightarrow^* \\ & & \$v_0a_1v_1 \ldots a_kv_k\# \ = \ \$u\#. \end{array}$$

where $k \geq 1, v_i \in L_2$, $0 \leq i \leq k$, $a_j \in (\Sigma_1 \cup \Sigma_2)$, $1 \leq j \leq k$, $u' \in L_1 \Leftarrow^* L_2$, which implies $\$u\# \in \$((L_1 \Leftarrow^* L_2) \Leftarrow L_2)\# \subseteq \$(L_1 \Leftarrow^* L_2)\#$.

This completes the proof of the inclusion $L(G) \subseteq \$(L_1 \Leftarrow^* L_2)\#$ and therefore, the proof of the claim.

If we consider now the morphism $h : \Sigma^* \longrightarrow \Sigma^*$ defined by $h(\$) = h(\#) = \lambda$ and $h(a) = a, \forall a \in \Sigma_1 \cup \Sigma_2$, it is easy to see that,

$$L_1 \Leftarrow^* L_2 = \begin{cases} h(L(G) - \{\$\#\}) \cup \{\lambda\}, & \text{if } \$\# \in L(G), \\ h(L(G)), & \text{if } \$\# \notin L(G). \end{cases}$$

It is obvious that h is a 3-linear erasing with respect to $L(G) - \{\$\#\}$. As the family of context-sensitive languages is closed under linear erasing and union, it follows that $L_1 \Leftarrow^* L_2$ is context-sensitive.

We have assumed until now that the language L_2 is λ-free. If the empty word belongs to L_2, it will be proved in the following that:

$$L_1 \Leftarrow^* L_2 = L_1 \leftarrow^* L_2.$$

" $\subseteq$ " This inclusion is obvious as any parallel insertion can be simulated by a sequence of sequential insertions and therefore:

$$L_1 \Leftarrow\!\!= L_2 \subseteq L_1 \Leftarrow^* L_2,$$

which implies $L_1 \Leftarrow\!\!=^* L_2 \subseteq L_1 \Leftarrow^* L_2$.

" $\supseteq$ " It can be showed, by induction on n, that

$$L_1 \Leftarrow^n L_2 \subseteq L_1 \Leftarrow\!\!=^* L_2.$$

$n = 0$. Obvious because $L_1 \subseteq (L_1 \Leftarrow\!\!=^* L_2)$.

$n \mapsto (n+1)$. Let α be a word in $(L_1 \Leftarrow^n L_2) \Leftarrow L_2$. Using the induction hypothesis we deduce that

$$\alpha \in (L_1 \Leftarrow\!\!=^* L_2) \Leftarrow L_2, \alpha = u_1 v u_2, u_1 u_2 \in (L_1 \Leftarrow\!\!=^* L_2), v \in L_2.$$

As $\lambda \in L_2$, it follows that α belongs also to $(u_1 u_2 \Leftarrow\!\!= L_2)$. Indeed, one can choose, with the exception of v, all the inserted words to be equal to λ. This further implies that:

$$\alpha \in (L_1 \Leftarrow\!\!=^* L_2) \Leftarrow\!\!= L_2 = \ L_1 \Leftarrow\!\!=^* L_2,$$

and the proof of the equality $L_1 \Leftarrow\!\!=^* L_2 = L_1 \Leftarrow^* L_2$ when $\lambda \in L_2$, is completed.

This completes the proof of the theorem as the family of context- sensitive languages is closed under iterated sequential insertion. □

2.3 Permuted insertion

Two words over Σ^* are called *letter-equivalent* iff, for every letter $a \in \Sigma$, both words contain the same number of occurrences of a. A language L is called *commutative* iff, whenever $w \in L$, all the words which are letter-equivalent to w belong to L. The *commutative closure* of a language L is the smallest commutative language containing L. The commutative closure can be viewed as a unary operation associating to every language L its commutative closure $com(L)$.

The *permuted sequential* (respectively *parallel*) *insertion*, which will be introduced in the sequel, is a combination between ordinary sequential (respectively parallel) insertion and the commutative closure (called also commutative variant in [7], p.363) operation. More precisely, given two words u and v, we will insert in u (instead of only v) all the words which are letter-equivalent to v.

Definition 2.4 *Let L_1, L_2 be two languages over the alphabet Σ. The permuted sequential insertion (permuted SIN) of L_2 into L_1 is defined as:*

$$L_1 \overset{\sim}{\leftarrow} L_2 = \bigcup_{u \in L_1, v \in L_2} (u \overset{\sim}{\leftarrow} com(v)).$$

Thus, the permuted SIN of L_2 into L_1 consists of the words obtained by inserting into words of L_1 all the words which are letter-equivalent to the words of L_2. Obviously, the permuted SIN can be expressed as $L_1 \overset{\sim}{\leftarrow} L_2 = L_1 \overset{\sim}{\leftarrow} com(L_2)$.

Example 2.9 Let $L_1 = \{\lambda\}$ and $L_2 = (ab)^*$. The permuted sequential insertion of L_2 into L_1 is:

$$L_1 \overset{\sim}{\leftarrow} L_2 = L_1 \overset{\sim}{\leftarrow} (com(L_2)) = com(L_2) =$$

$$\{w \in \{a,b\}^* |\ N_a(w) = N_b(w)\}.$$

□

The permuted SIN is not a commutative operation. For example, $a \overset{\sim}{\leftarrow} bc = a \overset{\sim}{\leftarrow} \{bc, cb\} = \{abc, bca, acb, cba\}$, whereas $bc \overset{\sim}{\leftarrow} a = \{abc, bac, bca\}$.

The permuted SIN is not associative. In general, for languages L_1, L_2, L_3, the sets $(L_1 \overset{\sim}{\leftarrow} L_2) \overset{\sim}{\leftarrow} L_3$ and $L_1 \overset{\sim}{\leftarrow} (L_2 \overset{\sim}{\leftarrow} L_3)$ are not comparable. For example, the word $adbe$ belongs to $a \overset{\sim}{\leftarrow} (b \overset{\sim}{\leftarrow} de)$ but not to the set $(a \overset{\sim}{\leftarrow} b) \overset{\sim}{\leftarrow} de$. We can use the languages from Example 2.2 to show that also the other inclusion does not hold.

Analogously with the permuted sequential insertion one can define the *permuted parallel insertion*, and a similar remark concerning its representation in terms of PIN and the commutative closure holds:

$$L_1 \overset{\sim}{\Leftarrow} L_2 = L_1 \overset{\sim}{\Leftarrow} (com(L_2)).$$

Example 2.10 Let $L_1 = \{\lambda\}$ and $L_2 = (abc)^*$. Then the permuted PIN of L_2 into L_1 is the same as the permuted SIN between them:

$$L_1 \overset{\sim}{\Leftarrow} L_2 = L_1 \overset{\sim}{\leftarrow} L_2 = L_1 \overset{\sim}{\leftarrow} (com(L_2)) = com(L_2) =$$

$$\{w \in \{a,b,c\}^* |\ N_a(w) = N_b(w) = N_c(w)\}.$$

□

The permuted PIN is not commutative. For example, $a \tilde{\Leftarrow} bc = a \Leftarrow \{bc, cb\} = \{bcabc, bcacb, cbabc, cbacb\}$, whereas $bc \tilde{\Leftarrow} a = bc \Leftarrow a = abaca$.

As concerning associativity, the following result holds:

Theorem 2.10 *If L_1, L_2, L_3 are languages over an alphabet Σ, then*

$$(L_1 \tilde{\Leftarrow} L_2) \tilde{\Leftarrow} L_3 \subseteq \quad L_1 \tilde{\Leftarrow} (L_2 \tilde{\Leftarrow} L_3).$$

Proof. The statement to be proved can be rewritten as:

$$(L_1 \Leftarrow com(L_2)) \Leftarrow com(L_3) \subseteq \quad L_1 \Leftarrow com(L_2 \Leftarrow com(L_3)).$$

Applying the associativity of the parallel insertion, all that remains to be shown is that:

$$com(L_2) \Leftarrow com(L_3) \subseteq com(L_2 \Leftarrow com(L_3)).$$

Let α be a word in the left member of the inclusion. Then α is of the form:

$$\alpha = w_0 a_1 w_1 a_2 \dots a_k w_k,$$

where $k \geq 0$, $w_i \in com(L_3)$, $0 \leq i \leq k$, and $\beta = a_1 \dots a_k \in com(b_1 \dots b_k)$, $b_1 \dots b_k \in L_2$, $a_j, b_j \in \Sigma$, $1 \leq j \leq k$. The case $k = 0$ corresponds to the situation where β is empty. The word α', obtained from α by permuting the letters a_i, $1 \leq i \leq k$, in order to obtain $b_1 \dots b_k$,

$$\alpha' = w_0 b_1 w_1 b_2 \dots b_k w_k,$$

belongs to $L_2 \Leftarrow com(L_3)$. This further implies that α is a word in the set $com(L_2 \Leftarrow com(L_3))$. □

However, the reverse inclusion does not always hold. For example, consider the languages $L_1 = \{a\}$, $L_2 = \{bc\}$, $L_3 = \{d\}$. The word *dddbcadddbc* belongs to $L_1 \tilde{\Leftarrow} (L_2 \tilde{\Leftarrow} L_3)$ but not to $(L_1 \tilde{\Leftarrow} L_2) \tilde{\Leftarrow} L_3$. Consequently, the permuted parallel insertion is not associative.

As expected, the fact that REG and CF are not closed under the commutative closure implies that they are not closed under permuted SIN. The next two theorems follow from Examples 2.9 and 2.10.

Theorem 2.11 *The family of regular languages is closed under neither permuted sequential nor permuted parallel insertion.*

Theorem 2.12 *The family of context-free languages is closed under neither permuted sequential nor permuted parallel insertion with regular languages.*

Corollary 2.3 *The family of context-free languages is closed under neither permuted sequential nor permuted parallel insertion.*

On the other hand, if the language to be inserted is a singleton, permuted SIN and PIN are family preserving for the families of the Chomsky hierarchy.

Theorem 2.13 *The families of regular, context-free and context-sensitive languages are closed under permuted sequential and permuted parallel insertion with singletons.*

Proof. Since the permuted insertion (sequential and parallel alike) with a singleton language amounts to the insertion of a finite language, our theorem follows by Theorems 2.3, 2.4. □

The family of context-sensitive languages is closed under SIN, PIN and the commutative closure. Consequently, it is closed also under permuted SIN and PIN.

Theorem 2.14 *The family of context-sensitive languages is closed under permuted sequential and permuted parallel insertion.*

2.4 Controlled insertion

We have dealt so far with operations where the insertion took place in arbitrary places of a word. As a consequence, catenation is not a particular case of any of these operations, because one cannot fix the position where the insertion takes place. A natural idea of controlling the position where the insertion is performed is that every letter determines what can be inserted after it. The catenation operation will be obtained then using a particular case of *controlled sequential insertion*.

Definition 2.5 *Let L be a language over the alphabet Σ. For each letter a of the alphabet, let $\Delta(a)$ be a language over an alphabet Σ_a. The Δ-controlled sequential insertion into L (shortly, controlled SIN), is defined as:*

$$L \leftarrowtail \Delta = \bigcup_{u \in L} (u \leftarrowtail \Delta),$$

where

$$u \leftarrow\!\!\!| \, \Delta = \{u_1 a v_a u_2 |\ u = u_1 a u_2, v_a \in \Delta(a)\}.$$

The function $\Delta : \Sigma \longrightarrow 2^{\Sigma'^*}$, *where* $\Sigma' = \bigcup_{a \in \Sigma} \Sigma_a$ *is called a control function.*

In the sequel, the control functions will be denoted by capital Greek letters.

Example 2.11 Let $L = \{a^3b, bc, \lambda, d^2\}$ and let Δ be the control function defined by $\Delta(a) = \{e, \lambda\}$, $\Delta(b) = \{f\}$, $\Delta(c) = \emptyset$, $\Delta(d) = \{d\}$. Then,

$$L \leftarrow\!\!\!| \, \Delta = \{a^3b, aea^2b, a^2eab, a^3eb, a^3bf, bfc, d^3\}.$$

□

All the insertion operations defined in the previous sections have been binary operations between languages. The controlled insertion is an $(n+1)$-ary operation, where n is the cardinality of Σ, the domain of the control function.

If we impose the restriction that for a distinguished letter $b \in \Sigma$, $\Delta(b) = L_2$, and $\Delta(a) = \emptyset$ for any letter $a \neq b$, we obtain a special case of controlled SIN, the *sequential insertion next to the letter* b, denoted $L \overset{b}{\leftarrow\!\!\!|} \ L_2$. The SIN next to a letter is a binary operation. The words from L which do not contain the letter b do not contribute to the result. A word in $L \overset{b}{\leftarrow\!\!\!|} \ L_2$ is obtained from $u \in L$ which contains b, by inserting a word of L_2 after *one* of the occurrences of b in it.

Example 2.12 If we consider the language and the control function defined in Example 2.11, then

$$\begin{aligned} L \overset{a}{\leftarrow\!\!\!|} \ \{e, \lambda\} &= \{a^3b, aea^2b, a^2eab, a^3eb\}, \\ L \overset{b}{\leftarrow\!\!\!|} \ \{f\} &= \{a^3bf, bfc\}, \\ L \overset{c}{\leftarrow\!\!\!|} \ \emptyset &= \emptyset, \\ L \overset{d}{\leftarrow\!\!\!|} \ \{d\} &= \{d^3\}. \end{aligned}$$

□

In general, the following relation holds: $L \leftarrow\!\!\!| \, \Delta = \bigcup_{a \in \Sigma} (L \overset{a}{\leftarrow\!\!\!|} \ \Delta(a))$.

Example 2.13 Let L_1, L_2 be the languages $L_1 = \{a^n c^n | \ n \geq 1\}$, $L_2 = \{d^m | \ m \geq 1\}$. The sequential insertion of L_2 into L_1, next to c, is:

$$L_1 \overset{c}{\longleftarrow\!\shortmid} L_2 = \{a^n c^p d^m c^q | \ n, m, p \geq 1, p + q = n\}$$

whereas the uncontrolled sequential insertion between L_1 and L_2 is:

$$L_1 \longleftarrow L_2 = (L_1 \overset{c}{\longleftarrow\!\shortmid} L_2) \cup (L_1 \overset{a}{\longleftarrow\!\shortmid} L_2) \cup L_2 L_1.$$

□

In general, if L_1, L_2 are languages over an alphabet Σ, then the sequential insertion of L_2 into L_1 can be expressed as:

$$L_1 \longleftarrow L_2 = \bigcup_{a \in \Sigma} (L_1 \overset{a}{\longleftarrow\!\shortmid} L_2) \cup L_2 L_1.$$

Note that the sequential insertion $L_1 \longleftarrow L_2$ can be obtained from the controlled SIN by using a marker $\#$ and a control function Δ which has the value L_2 for every letter of $\Sigma \cup \{\#\}$. Indeed,

$$L_1 \longleftarrow L_2 = h(\# L_1 \longleftarrow\!\shortmid \Delta),$$

where $\Delta(\#) = \Delta(a) = L_2, \forall a \in \Sigma$ and h is the morphism defined by $h(\#) = \lambda, h(a) = a, \forall a \in \Sigma$.

The catenation of the languages L_1 and L_2 can be obtained using a marker and the sequential insertion next to the marker. Indeed,

$$L_1 L_2 = h(L_1 \# \overset{\#}{\longleftarrow\!\shortmid} L_2),$$

where h is the morphism that erases the marker.

Note that in both the controlled SIN and the SIN next to a letter, the empty word does not occur in the result of the operation. Indeed, the notion of control implies the presence of at least one letter in the word in which the insertion is performed. Therefore, even if the word to be inserted is λ, the result contains at least the "control" letters. As it does not contain any control letter, the presence of λ in L_1 does not affect the result of the controlled SIN,

$$(L_1 \longleftarrow\!\shortmid \Delta) = (L_1 - \{\lambda\}) \longleftarrow\!\shortmid \Delta.$$

In the case of controlled sequential insertion we cannot talk about commutativity or associativity. Hovever, these notions can be studied in the case of SIN next to a letter.

Theorem 2.15 *If L_1, L_2, L_3 are languages over Σ and a, b are letters in Σ, then*

$$L_1 \overset{a}{\leftarrow\!\shortmid} (L_2 \overset{b}{\leftarrow\!\shortmid} L_3) \subseteq (L_1 \overset{a}{\leftarrow\!\shortmid} L_2) \overset{b}{\leftarrow\!\shortmid} L_3.$$

Proof. We can assume, without loss of generality, that a occurs in some words of L_1 and b occurs in some words of L_2. Indeed, if this is not the case, the left member of the inclusion is empty and the theorem holds.

Let α be a word in $L_1 \overset{a}{\leftarrow\!\shortmid} (L_2 \overset{b}{\leftarrow\!\shortmid} L_3)$. There exist words $u \in L_1$, $\beta \in L_2 \overset{b}{\leftarrow\!\shortmid} L_3$ such that $\alpha = u_1 a \beta u_2$, $u = u_1 a u_2$. As β belongs to $L_2 \overset{b}{\leftarrow\!\shortmid} L_3$, there exist words $v \in L_2$, $w \in L_3$ such that $\beta = v_1 b w v_2$, $v = v_1 b v_2$. According to the definition of the SIN next to a letter, α is a word of the set $u_1 a v u_2 \overset{b}{\leftarrow\!\shortmid} w$. As $u_1 a v u_2$ belongs to $u \overset{a}{\leftarrow\!\shortmid} v \subseteq L_1 \overset{a}{\leftarrow\!\shortmid} L_2$, we conclude that $\alpha \in (L_1 \overset{a}{\leftarrow\!\shortmid} L_2) \overset{b}{\leftarrow\!\shortmid} L_3$. □

The reverse inclusion does not hold. Indeed, for example, $a \overset{a}{\leftarrow\!\shortmid} (ba \overset{a}{\leftarrow\!\shortmid} c) = abac$, while $(a \overset{a}{\leftarrow\!\shortmid} ba) \overset{a}{\leftarrow\!\shortmid} c = \{acba, abac\}$. Consequently, the SIN next to a letter is not an associative operation.

The SIN next to a letter is also not commutative. For example, $a \overset{a}{\leftarrow\!\shortmid} b = ab$ whereas $b \overset{a}{\leftarrow\!\shortmid} a = \emptyset$.

The control that we have defined is actually a "right " control: a letter defines what can be inserted at its right side. However, a similar "left" controlled SIN can be defined, replacing in Definition 2.5 $u_1 a v_a u_2$ by $u_1 v_a a u_2$. The *left Δ-controlled SIN* into a language L is denoted by $L \leftarrow\!\prec \Delta$. The *left-SIN next to a letter* can be similarily defined.

Example 2.14 Let us consider the language and the control function defined in Example 2.11. Then we have:

$$\begin{aligned} L \leftarrow\!\prec \Delta &= \{a^3b, ea^3b, aea^2b, a^2eab, a^3fb, fbc, d^3\}, \\ L \overset{a}{\leftarrow\!\prec} \{e, \lambda\} &= \{a^3b, ea^3b, aea^2b, a^2eab\}, \\ L \overset{b}{\leftarrow\!\prec} \{f\} &= \{a^3fb, fbc\}, \\ L \overset{c}{\leftarrow\!\prec} \emptyset &= \emptyset, \\ L \overset{d}{\leftarrow\!\prec} \{d\} &= \{d^3\}. \end{aligned}$$

□

All the closure properties which will be proved in this section hold also for the left-controlled SIN because we have:

Theorem 2.16 *Let L be a language over Σ and $\Delta : \Sigma \longrightarrow 2^{\Sigma'^*}$ be a control function. Then,*

$$L' \leftarrow\!\!< \Delta' \; = Mi(L \leftarrow\!\!\dashv \Delta),$$

where $L' = Mi(L)$ and, for all $a \in \Sigma$, $\Delta'(a) = Mi(\Delta(a))$.

Proof. In order to prove the requested equality, the following facts will be used:

(i) $\mathrm{Mi}(\mathrm{Mi}(x)) = x$, $\forall x \in \Sigma^*$,

(ii) $\mathrm{Mi}(xy) = \mathrm{Mi}(y)\mathrm{Mi}(x)$, $\forall x, y \in \Sigma^*$.

Using (i), the equality to be proved can be rewritten as:

$$\mathrm{Mi}(\mathrm{Mi}(L) \leftarrow\!\!< \Delta') = \; L \leftarrow\!\!\dashv \Delta.$$

" $\subseteq$ " Let α be a word in $\mathrm{Mi}(\mathrm{Mi}(L) \leftarrow\!\!< \Delta')$. This implies that there exist $u \in \mathrm{Mi}(L)$, $u = a_1 a_2 \dots a_k$, $k \geq 1$, $a_j \in \Sigma$, $1 \leq j \leq k$, an index i, $1 \leq i \leq k$ and $v \in \Delta'(a_i) = \mathrm{Mi}(\Delta(a_i))$, such that $\alpha = \mathrm{Mi}(a_1 a_2 \dots a_{i-1} v a_i \dots a_k)$. As u is a word in $\mathrm{Mi}(L)$ and v a word in $\mathrm{Mi}(\Delta(a_i))$, from (i) it follows that $a_k a_{k-1} \dots a_1 \in L$ and that $\mathrm{Mi}(v) \in \Delta(a_i)$. Using (ii) we deduce that

$$\alpha = a_k a_{k-1} \dots a_i \mathrm{Mi}(v) a_{i-1} \dots a_2 a_1,$$

$$a_k a_{k-1} \dots a_2 a_1 \in L, \mathrm{Mi}(v) \in \Delta(a_i),$$

that is, $\alpha \in L \leftarrow\!\!\dashv \Delta$.

" $\supseteq$ " Let α be a word in $L \leftarrow\!\!\dashv \Delta$. There exist $u \in L$, $u = a_1 a_2 \dots a_k$, $k \geq 1$, $a_j \in \Sigma$, $1 \leq j \leq k$, an index i, $1 \leq i \leq k$ and $v \in \Delta(a_i)$, $1 \leq i \leq k$, such that $\alpha = a_1 a_2 \dots a_i v a_{i+1} \dots a_k$. As $a_k a_{k-1} \dots a_1$ is a word in $\mathrm{Mi}(L)$ and $\mathrm{Mi}(v) \in \Delta'(a_i)$, using (ii) and (i) we deduce that:

$$\mathrm{Mi}(\alpha) = a_k a_{k-1} \dots a_{i+1} \mathrm{Mi}(v) a_i \dots a_2 a_1$$

is a word in $\mathrm{Mi}(L) \leftarrow\!\!< \Delta'$, therefore α belongs to $\mathrm{Mi}(\mathrm{Mi}(L) \leftarrow\!\!< \Delta')$. □

In the following we shall define the parallel variants of controlled insertion. The definitions originate from the parallel insertion, but in the controlled case every letter defines the language that may be inserted after it.

Definition 2.6 *Let L be a language over Σ and $\Delta : \Sigma \longrightarrow 2^{\Sigma'^*}$ a control function satisfying $\Delta(a) \neq \emptyset$, $\forall a \in \Sigma$. The Δ-controlled parallel insertion into L (shortly, controlled PIN) is defined as:*

$$L \Leftarrow\!\!\dashv \Delta = \bigcup_{u \in L} (u \Leftarrow\!\!\dashv \Delta),$$

where

$$u \Leftarrow\!\!\dashv \Delta = \{a_1 v_1 a_2 v_2 \dots a_k v_k |\ u = a_1 \dots a_k, k \geq 1, a_i \in \Sigma,$$

$$\textit{and } v_i \in \Delta(a_i), 1 \leq i \leq k\}.$$

Example 2.15 Let $L = \{cd, \lambda, bdc, f^2\}$ and Δ be the control function defined by $\Delta(c) = \{a\}$, $\Delta(d) = \{\lambda, e\}$, $\Delta(b) = \{\lambda\}$, $\Delta(f) = \{f\}$. Then,

$$L \Leftarrow\!\!\dashv \Delta = \{cad, cade, bdca, bdeca, f^4\}.$$

□

The parallel insertion of a nonempty language L_2 into a language L_1 can be obtained using a marker # and a control function $\Delta(a) = L_2$, $\forall a \in \Sigma \cup \{\#\}$. Indeed,

$$L_1 \Leftarrow\!= L_2 = h(\# L_1 \Leftarrow\!\!\dashv \Delta),$$

where h is the morphism that erases the marker.

Note that in the previous definition the control function cannot have the empty set as its value. This condition has been introduced because of the following reasons. If there would exist a letter $a \in \Sigma$ such that $\Delta(a) = \emptyset$, then all the words of $u \in L$ which contain a would give $u \Leftarrow\!\!\dashv \Delta = \emptyset$. This means that these words would not contribute to the result of the controlled PIN. Consequently, we can introduce, without loss of generality, the condition $\Delta(a) \neq \emptyset$, $\forall a \in \Sigma$.

If we impose the restriction that for a distinguished letter $b \in \Sigma$ we have $\Delta(b) = L_2$, $L_2 \subseteq \Sigma'^*$, and $\Delta(a) = \lambda$ for every letter $a \neq b$, we obtain a particular case of controlled PIN: *parallel insertion next to the letter b*. It is a binary language operation, whereas the arity of the Δ-controlled PIN equals $\text{card}(\Sigma) + 1$. The parallel insertion of L_2 into L_1, next to b, is denoted by $L_1 \overset{b}{\Leftarrow\!\!\dashv} L_2$. It retains all the nonempty words from L_1 which do not contain b (we inserted λ after each letter). The other words from $L_1 \overset{b}{\Leftarrow\!\!\dashv} L_2$ are obtained from words $u \in L_1$ containing the letter b. After each occurrence of b in u, a word from L_2 is inserted.

Example 2.16 If one considers the language and the control function defined in Example 2.15, we have:

$$\begin{aligned} L \overset{c}{\Leftarrow\!\!\shortmid} \{a\} &= \{cad, bdca, f^2\}, \\ L \overset{d}{\Leftarrow\!\!\shortmid} \{\lambda, e\} &= \{cd, cde, bdc, bdec, f^2\}, \\ L \overset{b}{\Leftarrow\!\!\shortmid} \{\lambda\} &= \{cd, bdc, f^2\}, \\ L \overset{f}{\Leftarrow\!\!\shortmid} \{f\} &= \{cd, bdc, f^4\}. \end{aligned}$$

□

The catenation between two languages L_1 and $L_2 \neq \emptyset$ can be obtained by using a marker and the parallel insertion next to it. Indeed,

$$L_1 L_2 = h(L_1 \# \overset{\#}{\Leftarrow\!\!\shortmid} L_2),$$

where h is the morphism that erases the marker.

Note that also in the parallel variant, the presence of λ in L_1 does not change the result of the operation:

$$(L_1 \Leftarrow\!\!\shortmid \Delta) = (L_1 - \{\lambda\}) \Leftarrow\!\!\shortmid \Delta,$$

$$L_1 \overset{a}{\Leftarrow\!\!\shortmid} L_2 = (L_1 - \{\lambda\}) \overset{a}{\Leftarrow\!\!\shortmid} L_2.$$

In the case of controlled PIN, we cannot speak about associativity or commutativity. However, these notions can be studied when we restrict to the parallel insertion next to a letter. For arbitrary languages L_1, L_2, L_3, the sets $L_1 \overset{a}{\Leftarrow\!\!\shortmid} (L_2 \overset{b}{\Leftarrow\!\!\shortmid} L_3)$ and $(L_1 \overset{a}{\Leftarrow\!\!\shortmid} L_2) \overset{b}{\Leftarrow\!\!\shortmid} L_3$ are not comparable. For example, $bb \overset{b}{\Leftarrow\!\!\shortmid} (ab \overset{b}{\Leftarrow\!\!\shortmid} c) = babcbabc$, while $(bb \overset{b}{\Leftarrow\!\!\shortmid} ab) \overset{b}{\Leftarrow\!\!\shortmid} c = bcabcbcabc$. Consequently, the parallel insertion next to a letter is not an associative operation. It is also not a commutative operation. For example, $a \overset{a}{\Leftarrow\!\!\shortmid} b = ab$, whereas $b \overset{a}{\Leftarrow\!\!\shortmid} a = b$.

Analogously to the sequential case, a "left" control can be defined for parallel insertion, by replacing in Definition 2.6 $a_1 v_1 a_2 v_2 \dots a_k v_k$ by $v_1 a_1 v_2 a_2 \dots v_k a_k$. The *left* Δ*-controlled PIN* into the language L is denoted by $L \Leftarrow\!\!\shortmid \Delta$. The *left-PIN next to a letter* is defined in a similar way.

Example 2.17 If we consider the language and the control function defined in Example 2.15, we have:

$$\begin{aligned} L \Leftarrow \Delta = &\quad \{acd, aced, bdac, bedac, f^4\}, \\ L \overset{c}{\Leftarrow} \{a\} = &\quad \{acd, bdac, f^2\}, \\ L \overset{d}{\Leftarrow} \{\lambda, e\} = &\quad \{cd, ced, bdc, bedc, f^2\}, \\ L \overset{b}{\Leftarrow} \{\lambda\} = &\quad L - \{\lambda\}, \\ L \overset{f}{\Leftarrow} \{f\} = &\quad \{cd, bdc, f^4\}. \end{aligned}$$

□

The following theorem shows the similarity of the left and right controlled PIN.

Theorem 2.17 *Let L be a language over the alphabet Σ and $\Delta : \Sigma \longrightarrow 2^{\Sigma'^*}$ be a control function such that $\Delta(a) \neq \emptyset$, $\forall a \in \Sigma$. Then,*

$$(L' \Leftarrow \Delta') = Mi(L \Leftarrow \Delta),$$

where $L' = Mi(L)$ and, for all $a \in \Sigma$, $\Delta'(a) = Mi(\Delta(a))$.

Proof. Similar to the proof of the Theorem 2.16. □

In the sequel we shall prove the closure properties of the families of the Chomsky hierarchy under controlled sequential and parallel insertion.

Theorem 2.18 *The families of regular, context-free and context-sensitive languages are closed under controlled sequential insertion.*

Proof. Consider a family of languages $\mathcal{L} \in \{$ REG, CF, CS$\}$. Let Σ be an alphabet and $L \subseteq \Sigma^*$ be a language in $\mathcal{L}$. According to an earlier remark, we can assume that L is λ-free. Let $\Delta : \Sigma \longrightarrow 2^{\Sigma'^*}$ be a control function such that $\Delta(a) \in \mathcal{L}$ for all $a \in \Sigma$.

For every $a \in \Sigma$ for which $\Delta(a) \neq \emptyset$ consider the marker $\#_a$ not in Σ. Let g be the λ-free gsm:

$$\begin{aligned} g = &\quad (\Sigma, \Sigma \cup \{\#_a |\ a \in \Sigma, \Delta(a) \neq \emptyset\}, \{s_0, s\}, s_0, \{s\}, P), \\ P = &\quad \{s_0 a \longrightarrow a s_0 |\ a \in \Sigma\} \cup \{s_0 a \longrightarrow a \#_a s |\ a \in \Sigma, \Delta(a) \neq \emptyset\} \cup \\ &\quad \{s_0 a \longrightarrow a s |\ a \in \Sigma \text{ and } \lambda \in \Delta(a)\} \cup \{sa \longrightarrow as |\ a \in \Sigma\}. \end{aligned}$$

Intuitively, the gsm g works as follows. Given a word $u \in \Sigma^+$ as an input, g puts after an arbitrary letter a occurring in u the marker $\#_a$, provided $\Delta(a)$ is nonempty. All the letters from u are left unchanged. The entire word u may remain unchanged in case it contains a letter a with $\lambda \in \Delta(a)$. The construction is similar to that of Theorem 2.3 with the following differences:

- No marker is put at the beginning of the word;
- There exists a different marker for every letter a with $\Delta(a) \neq \emptyset$;
- The situation $\lambda \in \Delta(a)$, $a \in \Sigma$, is taken care of in the construction of P.

In a similar way as in the Claim of Theorem 2.3 it can be showed that:

$$g(L) = \{u_1 a \#_a u_2 |\ u_1 a u_2 \in L, \Delta(a) \neq \emptyset\} \cup \{u_1 a u_2 |\ u_1 a u_2 \in L, \lambda \in \Delta(a)\}.$$

Consider now the λ-free substitution:

$$s' : (\Sigma \cup \{\#_a |\ a \in \Sigma, \Delta(a) \neq \emptyset\})^* \longrightarrow 2^{(\Sigma \cup \Sigma')^*},$$

defined by $s'(\#_a) = \Delta(a) - \{\lambda\}$ if $a \in \Sigma$, $\Delta(a) \neq \emptyset$ and $s'(a) = a$, for all a in Σ.

It is easy to show that $s'(g(L)) = L \leftarrowtail \Delta$. The family $\mathcal{L}$ is closed under λ-free gsm mappings and under λ-free substitutions. Consequently, it will be closed also under controlled sequential insertion. □

Theorem 2.19 *The families of regular, context-free and context-sensitive languages are closed under controlled parallel insertion.*

Proof. Let L be a language over the alphabet Σ and $\Delta : \Sigma \longrightarrow 2^{\Sigma'^*}$ a control function such that $\Delta(a) \neq \emptyset$, $\forall a \in \Sigma$. Assume further that L and $\Delta(a), a \in \Sigma$ are languages belonging to the same family $\mathcal{L} \in \{$ REG, CF, CS$\}$. Then,

$$L \leftarrowtail\!\!\!\!\!\!= \Delta = \sigma(L - \{\lambda\}),$$

where σ is the λ-free substitution defined by:

$$\sigma : \Sigma^* \longrightarrow 2^{(\Sigma \cup \Sigma')^*}, \sigma(a) = a\Delta(a), \forall a \in \Sigma.$$

The theorem now follows from the closure of the families REG, CF, CS under λ-free substitution.

□

2.5 Scattered sequential insertion

In the operations defined in the previous sections, the insertion was performed in a *compact* way. Always when performing an insertion operation of a language L_2 into L_1 one (in the sequential case) or more (in the parallel case) words from L_2 were compactly inserted into words of L_1. Some of the insertion operations can be defined in a *scattered* sense as well. More precisely, instead of inserting a compact word, the letters which form it will be sparsely inserted. In the case of sequential insertion, this modification from the compact sense to the scattered sense yields the well known *shuffle* operation (see, for example [7], p.156, [3], pp.175-180).

Definition 2.7 *Let L_1, L_2 be languages over the alphabet Σ. The shuffle of L_2 into L_1 is defined as:*

$$L_1 \amalg L_2 = \bigcup_{u \in L_1, v \in L_2} (u \amalg v),$$

where

$$\begin{aligned} u \amalg v = \quad & \{u_1 v_1 u_2 v_2 \dots u_k v_k |\ u = u_1 u_2 \dots u_k, v = v_1 v_2 \dots v_k, \\ & k \geq 1, u_i, v_i \in \Sigma^*, 1 \leq i \leq k\}. \end{aligned}$$

Example 2.18 Let $L_1 = \{abb\}$, $L_2 = \{cd\}$. The *shuffle* of L_2 into L_2 is:

$$\begin{aligned} L_1 \amalg L_2 = \quad & \{cdabb, cadbb, cabdb, cabbd, acdbb, \\ & acbdb, acbbd, abcdb, abcbd, abbcd\}, \end{aligned}$$

and the *shuffle* of L_1 into L_2 is $L_2 \amalg L_1 = L_1 \amalg L_2$. □

The *shuffle* operation is commutative and associative. $(\mathcal{P}(\Sigma^*), \amalg)$ is a commutative monoid with $\{\lambda\}$ the neutral element. The families of regular and context-free languages are closed under *shuffle* with regular languages. The family of context-free languages is not closed under *shuffle*. A constructive argument of the fact that the family of context-sensitive languages is closed under *shuffle* is given below.

Theorem 2.20 *The family of context-sensitive languages is closed under shuffle.*

Proof. Let L_1, L_2 be two languages over Σ, generated by the context-sensitive grammars $G_i = (N_i, \Sigma_i, S_i, P_i)$, $i = 1, 2$, which satisfy the requirements of Theorem 2.7. Assume that the empty word belongs to neither L_1 nor L_2.

Let us construct the context-sensitive grammar $G = (N, \Sigma, S, P)$ whose components are respectively,

$$\begin{aligned} N = &\quad N_1 \cup N_2 \cup \{S\} \cup \{a' |\ a \in \Sigma_2\}, \\ \Sigma = &\quad \Sigma_1 \cup \Sigma_2, \\ P = &\quad P_1 \cup P_2' \cup \{S \longrightarrow S_1 S_2\} \cup \\ &\quad \{ba' \longrightarrow a'b |\ a \in \Sigma_2, b \in \Sigma_1\} \cup \\ &\quad \{a' \longrightarrow a |\ a \in \Sigma_2\}, \end{aligned}$$

where P_2' contains the rules of P_2 with all the terminals a replaced by their nonterminal versions a', and S is a symbol which does not occur in any of the given vocabularies.

For a word $v = a_1 \dots a_k$, $k \geq 0$, $a_i \in \Sigma_2$, $1 \leq i \leq k$, we denote by v' the corresponding word where all letters have been primed, $v' = a_1' \dots a_k'$.

Claim. $L_1 \amalg L_2 = L(G)$.

" $\subseteq$ " Let α be a word in $L_1 \amalg L_2$. There exist non-empty words $u \in L_1$, $v \in L_2$ such that $\alpha = u_1 v_1 u_2 v_2 \dots u_k v_k$, $u = u_1 u_2 \dots u_k$, $v = v_1 v_2 \dots v_k$, $k \geq 1$, $u_i, v_i \in \Sigma^*, 1 \leq i \leq k$. That further implies that the following derivations exist :

$$S_1 \Longrightarrow^* u, \text{ in } G_1,$$

$$S_2 \Longrightarrow^* v, \text{ in } G_2.$$

We can now construct the derivation according to G:

$$\begin{aligned} S \Longrightarrow &\quad S_1 S_2 \Longrightarrow^* u S_2 \Longrightarrow^* uv' = u_1 u_2 \dots u_k v_1' v_2' \dots v_k' \Longrightarrow^* \\ &\quad u_1 v_1' u_2 v_2' \dots u_k v_k' \Longrightarrow^* u_1 v_1 u_2 v_2 \dots u_k v_k. \end{aligned}$$

After applying the rule $S \longrightarrow S_1 S_2$, the derivation $S_1 \Longrightarrow^* u$ was used. Afterwards, the derivation $S_2 \Longrightarrow^* v$, where all the terminal rules $X \longrightarrow a$ have been replaced by $X \longrightarrow a'$, has been applied. Rules of the type $ba' \longrightarrow a'b$ have been used until $v_1', \dots v_k'$ have reached the desired places. Finally, the productions $a' \longrightarrow a, a \in \Sigma_2$, have transformed the nonterminals into terminals.

The existence of this derivation shows that $\alpha \in L(G)$.

" $\supseteq$ " Let α be a word in $L(G)$.

There exists a derivation $S \Longrightarrow S_1S_2 \Longrightarrow^* \alpha$ according to G. As the terminal rules are context-free, we can rearrange the derivation in the form:

$$S \Longrightarrow S_1S_2 \Longrightarrow^* uv' \Longrightarrow^* \alpha,\ u \in \Sigma_1^*,\ v \in \Sigma_2^*.$$

Moreover, as the rules from P_1 and P_2 do not mix, the nonterminal vocabularies being disjoint, we have:

$$S_1 \Longrightarrow^* u, \text{ in } G_1,$$

$$S_2 \Longrightarrow^* v, \text{ in } G_2.$$

The only way the derivation can develop after reaching the word uv' is that some rules $ba' \longrightarrow a'b$ are applied, and rules $a' \longrightarrow a$ transform the remaining nonterminals into terminals. We can rewrite the derivation of α as:

$$\begin{array}{rl} S \Longrightarrow & S_1S_2 \Longrightarrow^* uv' = u_1u_2 \dots u_kv_1'v_2' \dots v_k' \Longrightarrow^* \\ & u_1v_1u_2v_2 \dots u_kv_k = \alpha, \\ S_1 \Longrightarrow^* & u = u_1u_2 \dots u_k, \text{ in } G_1, \\ S_2 \Longrightarrow^* & v = v_1v_2 \dots v_k, \text{ in } G_2. \end{array}$$

That implies that $\alpha \in (u \amalg v) \subseteq L_1 \amalg L_2$, and the proof of the claim is complete.

The theorem holds also if λ is a word in L_1 or L_2 because:

$$\begin{array}{lll} L_1 \amalg L_2 = & [(L_1 - \{\lambda\}) \amalg (L_2 - \{\lambda\})] \cup L_1, & \text{if } \lambda \in L_2 - L_1, \\ L_1 \amalg L_2 = & [(L_1 - \{\lambda\}) \amalg (L_2 - \{\lambda\})] \cup L_2, & \text{if } \lambda \in L_1 - L_2, \\ L_1 \amalg L_2 = & [(L_1 - \{\lambda\}) \amalg (L_2 - \{\lambda\})] \cup L_1 \cup L_2, & \text{if } \lambda \in L_1 \cap L_2, \end{array}$$

and because the family CS is closed under union.

□

The parallel as well as the controlled variants of insertion do not have their natural scattered counterparts. However, for the permuted sequential insertion, a scattered version can be defined.

Definition 2.8 *Let L_1, L_2 be languages over Σ. The permuted scattered sequential insertion (permuted scattered SIN) of L_2 into L_1 is defined as:*

$$L_1 \leftsquigarrow L_2 = \bigcup_{u \in L_1, v \in L_2} (u \leftsquigarrow v),$$

where $u \leftsquigarrow v = u \amalg com(v)$.

As in the case of permuted sequential insertion, when performing the operation $u \leftsquigarrow v$, only the amounts of letters contained in v matter; their order, that is, the structure of v is irrelevant. The difference is that while in the compact variant the whole word v was inserted, now the letters of v are sparsely inserted into u.

Example 2.19 Let $L_1 = \{ab^2\}$, $L_2 = \{cd\}$.

$$\begin{aligned} ab^2 \leftsquigarrow cd = \quad & \{cdab^2, cadb^2, cabdb, cab^2d, acdb^2, \\ & acbdb, acb^2d, abcdb, abcbd, ab^2cd, \\ & dcab^2, dacb^2, dabcb, dab^2c, adcb^2, \\ & adbcb, adb^2c, abdcb, abdbc, ab^2dc\}. \end{aligned}$$

□

By combining the *shuffle* with the *com* operation, the commutativity property is lost. For example, $ab \leftsquigarrow c = \{cab, acb, abc\}$, whereas $c \leftsquigarrow ab = \{cab, abc, acb, cba, bac, bca\}$. However, the associativity property is preserved and therefore we have:

Theorem 2.21 *The permuted scattered SIN induces a monoid structure on* $\mathcal{P}(\Sigma^*)$.

Proof. The neutral element of the monoid is λ.

For showing that the permuted scattered SIN is associative we have to prove that

$$L_1 \leftsquigarrow (L_2 \leftsquigarrow L_3) = (L_1 \leftsquigarrow L_2) \leftsquigarrow L_3,$$

for languages L_1, L_2, L_3 over Σ.

If one of the languages is empty, the equality holds, both members being equal to the empty set.

Else, the two members of the equality can be rewritten as follows:

$$\begin{aligned} L_1 \leftsquigarrow (L_2 \leftsquigarrow L_3) = \quad & L_1 \amalg com(L_2 \amalg com(L_3)), \\ (L_1 \leftsquigarrow L_2) \leftsquigarrow L_3 = \quad & (L_1 \amalg com(L_2)) \amalg com(L_3) = \\ & L_1 \amalg (com(L_2) \amalg com(L_3)). \end{aligned}$$

(we have used the associativity of the *shuffle* operation). The theorem now follows as we obviously have:

$$com(L_2 \amalg com(L_3)) = com(L_2) \amalg com(L_3).$$

□

As in the case of permuted sequential insertion, the mixture with the commutative closure implies the nonclosure of REG and CF under the new operation.

Theorem 2.22 *The family of regular and the family of context-free languages are not closed under permuted scattered SIN with regular languages.*

Proof. Let L_1, L_2 be two languages over an alphabet Σ. If $L_1 = \{\lambda\}$, then the permuted scattered SIN amounts to permuted SIN, therefore we can use Examples 2.9, 2.10 to prove the theorem. □

Corollary 2.4 *The family of regular and the family of context-free languages are not closed under permuted scattered SIN.*

On the other hand, if the inserted language is a singleton, its commutative closure is a finite set and therefore we have:

Theorem 2.23 *The family of regular and the family of context-free languages are closed under permuted scattered SIN with singletons.*

Proof. Let L_1 and w be a language, respectively a word over the same alphabet Σ. The theorem follows because

$$L_1 \leftsquigarrow w = L_1 \amalg com(\{w\}),$$

and REG, CF are closed under *shuffle* with regular languages. □

The closure of the family CS under both commutative closure and *shuffle* implies the following:

Theorem 2.24 *The family of context-sensitive languages is closed under permuted scattered SIN.*

□

Bibliography

[1] M.Andraşiu, A.Atanasiu, Gh.Păun, A.Salomaa. A new cryptosystem based on formal language theory. *Bull.Math.Soc.Sci.Math.Roumanie*, to appear.

[2] M.Andraşiu, Gh.Păun, A.Salomaa. Language-theoretical problems arising from Richelieu cryptosystems. Submitted for publication.

[3] S.Ginsburg. *Algebraic and Automata-Theoretic Properties of Formal Languages*. North-Holland, Amsterdam, 1975.

[4] M.A.Harrison. *Introduction to Formal Language Theory*. Addison Wesley, Reading, Massachusetts, 1978.

[5] J.Hopcroft, J.Ulmann. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, Reading, Massachusetts, 1979.

[6] H.C.M.Kleijn, G.Rozenberg. Context-free like restrictions on selective rewriting. *Theoretical Computer Science*, vol.16, no.3(1981), pp.237-269.

[7] W.Kuich, A.Salomaa. *Semirings, Automata, Languages*. Springer Verlag, Berlin, 1986.

[8] R.C.Lyndon, M.P.Schutzenberger. The equation $a^M = b^N c^P$ in a free group, *Michigan Math.J.*, no.9(1962) pp.289-298.

[9] Gh.Păun, A.Salomaa. Semi-commutativity sets – a cryptographically grounded topic. *Bull.Math.Soc.Sci.Math.Roumanie*, to appear.

[10] G.Rozenberg, A.Salomaa. *The Mathematical Theory of L Systems*. Academic Press, London, 1980.

[11] A.Salomaa. *Theory of Automata*. Pergamon Press, Oxford, 1969.

[12] A.Salomaa. *Formal Languages*. Academic Press, London, 1973.

[13] L.Sântean[1]. Six arithmetic-like operations on languages. *Revue Roumaine de Linguistique*, Tome XXXIII, 1988, Cahiers de linguistique theorique et applique, Tome XXV, 1988, No.1, Janvier-Juin, pp.65-73.

[14] H.J.Shyr. *Free Monoids and Languages*. Lecture Notes, Institute of applied mathematics, National Chung-Hsing University, Taichung, Taiwan, 1991.

[15] H.J.Shyr, G.Thierrin, S.S.Yu. Monogenic e-closed languages and dipolar words. To appear.

[1]The maiden name of the author of this Thesis