Context-Free Languages and Pushdown Automata

Jean-Michel Autebert[1], Jean Berstel[2], and Luc Boasson[3]

[1] UFR d'Informatique, Université Denis Diderot, Paris
[2] LITP, IBP, Université Pierre et Marie Curie, Paris
[3] LITP, IBP, Université Denis Diderot, Paris

Contents

1. Introduction

This chapter is devoted to context-free languages. Context-free languages and grammars were designed initially to formalize grammatical properties of natural languages [9]. They subsequently appeared to be well adapted to the formal description of the syntax of programming languages. This led to a considerable development of the theory.

The presentation focuses on two basic tools: context-free grammars and pushdown automata. These are indeed the standard tools to generate and to recognize context-free languages. A contrario, this means also that we do not consider complexity results at all, neither of recognition by various classes of sequential or parallel Turing machines nor of "succinctness" (see e.g. [52]), that is a measure of the size of the description of a language.

We have chosen to present material which is not available in textbooks [17, 29, 1, 47, 28, 4, 30, 32, 2] (more precisely not available in more than one textbook) because it is on the borderline between classical stuff and advanced topics. However, we feel that a succinct exposition of these results may give some insight in the theory of context-free languages for advanced beginners, and also provide some examples or counter-examples for researchers.

This section ends with notation and examples. In Section 2, we present the relationship between grammars and systems of equations. As an example of the interest of this formalism, we give a short proof of Parikh's theorem.

In the next section, three normal forms of context-free grammars are established. The one with most applications is Greibach's normal form: several variants are given and, in Section 4, we present four such applications. The first three are closely related to each other.

Section 5 is devoted to pushdown automata. We consider carefully the consequences of various restrictions of the general model. The section ends with two results: one concerning the pushdown store language of a pda, the other the output language of a pushdown down transducer.

In the last section, we consider eight important subfamilies of context-free languages. We study in detail linear and quasi-rational languages, and present more briefly the other families.

In the bibliography, we have generally tried to retrieve the references to the original papers, in order to give some flavour of the chronological development of the theory.

1.1 Grammars

As general notation, we use ε to denote the empty word, and $|w|$ for the length of the word w.

A *context-free grammar* $G = (V, P)$ over an alphabet A is composed of a finite alphabet V of *variables* or *nonterminals* disjoint from A, and a finite set $P \subset V \times (V \cup A)^*$ of *productions* or *derivation rules*. Letters in A are called *terminal letters*.

Given words u, $v \in (V \cup A)^*$, we write $u \longrightarrow v$ (sometimes subscripted by G or by P) whenever there exist factorizations $u = xXy$, $v = x\alpha y$, with (X, α) a production. A *derivation* of length $k \geq 0$ from u to v is a sequence $(u_0, u_1, \ldots, u_k)$ of words in $(V \cup A)^*$ such that $u_{i-1} \longrightarrow u_i$ for $i = 1, \ldots, k$, and $u = u_0$, $v = u_k$. If this holds, we write $u \xrightarrow{k} v$. The existence of some derivation from u to v is denoted by $u \xrightarrow{*} v$. If there is a proper derivation (i.e. of length ≥ 1), we use the notation $u \xrightarrow{+} v$. The *language generated* by a variable X in grammar G is the set

$$L_G(X) = \{w \in A^* \mid X \xrightarrow{*} w\}$$

Frequently, grammars are presented with a distinguished nonterminal called the *axiom* and usually denoted by S. The language generated by this variable S in a grammar is then called the language *generated by the grammar*, for short, and is denoted $L(G)$. Any word in $(V \cup A)^*$ that derives from S is a *sentential form*.

A language L is called *context-free* if it is the language generated by some variable in a context-free grammar. Two grammars G and G' are *equivalent* if they generate the same language, i. e. if the distinguished variables S and S' are such that $L_G(S) = L'_{G'}(S')$.

More generally, if $x \in (V \cup A)^*$, we set

$$L_G(x) = \{w \in A^* \mid x \xrightarrow{*} w\} \ .$$

Context-freeness easily implies that

$$L_G(xy) = L_G(x)L_G(y) \ .$$

Consider a derivation $u = u_0 \longrightarrow u_1 \longrightarrow \cdots \longrightarrow u_k = v$, with $u, v \in (V \cup A)^*$. Then there exist productions $p_i = X_i \rightarrow \alpha_i$ and words x_i, y_i such that

$$u_i = x_i X_i y_i, \quad u_{i+1} = x_i \alpha_i y_i \qquad (i = 0, \ldots, k-1)$$

The derivation is *leftmost* if $|x_i| \leq |x_{i+1}|$ for $i = 0, \ldots, k-2$, and *rightmost* if, symmetrically, $|y_i| \leq |y_{i+1}|$ for $i = 0, \ldots, k-2$. A leftmost (rightmost) derivation is denoted by

$$u \xrightarrow[\ell]{*} v, \qquad u \xrightarrow[r]{*} v$$

It is an interesting fact that any word in a context-free language $L_G(X)$ has the same number of leftmost and of rightmost derivations. A grammar $G = (V, P)$ is *unambiguous* for a variable X if every word in $L_G(X)$ has exactly one leftmost (rightmost) derivation. A language is *unambiguous* if there is an unambiguous grammar to generate it, otherwise it is called *inherently ambiguous*.

A grammar $G = (V, P)$ over A is *trim* in the variable S if the following two conditions are fulfilled :

(i) for every nonterminal X, the language $L_G(X)$ is nonempty;

(ii) for every $X \in V$, there exist $u, v \in A^*$ such that $S \xrightarrow{*} uXv$.

The second condition means that every variable is "accessible", and the first that any variable is "co-accessible". It is not difficult to see that a grammar can always be trimmed effectively. A variation of condition (i) which is sometimes useful is to require that $L_G(X)$ is infinite for every variable X (provided the language generated by the grammar is itself infinite).

A production is *terminal* if its right side contains no variable. A production is called an *ε-rule* if its right side is the empty word. At least one ε-production is necessary if the language generated by the grammar contains the empty word. It is not too difficult to construct, for every context-free grammar G, an equivalent grammar with no ε-production excepted a production $S \longrightarrow \varepsilon$ if $\varepsilon \in L(G)$. The final special kind of grammars we want to mention is the class of proper grammars. A grammar G is *proper* if it has neither ε-productions nor any production of the form $X \longrightarrow Y$, with Y a variable. Again, an equivalent proper grammar can effectively be constructed for any grammar G if $L(G) \not\ni \varepsilon$. These constructions are presented in most textbooks. Normal forms are the topic of the next section.

1.2 Examples

There are several convenient shorthands to describe context-free grammars. Usually, a production (X, α) is written $X \longrightarrow \alpha$, and productions with same left side are grouped together, the corresponding right sides being separated by a '+'. Usually, the variables and terminal letters are clear from the context.

Subsequently, we make use several times of the following notation. Let A be an alphabet. A *copy* of A is an alphabet that is disjoint from A and in bijection with A. A copy is frequently denoted $\bar{A}$ or A'. This implicitly means that the bijection is denoted similarly, namely as the mapping $a \mapsto \bar{a}$ or $a \mapsto a'$. The inverse bijection is denoted the same, that is $\bar{\bar{a}} = a$ (resp. $(a')' = a$), and is extended to a bijection from $(A \cup \bar{A})^*$ into itself (the same for 'bar' replaced by 'prime') by $\overline{xy} = \bar{y}\,\bar{x}$.

The *Dyck languages*. Let A be an alphabet and let $\bar{A}$ be a copy. The *Dyck language* over A is the language D_A^* generated by S in the grammar

$$S \longrightarrow TS + \varepsilon\ ; \qquad T \longrightarrow aS\bar{a} \quad (a \in A)$$

The notation is justified by the fact that D_A^* is indeed a submonoid of $(A \cup \bar{A})^*$. It is even a free submonoid, generated by the language D_A of *Dyck primes* which is the language generated by the variable T in the grammar above. If A has n letters, then the notation D_n^* is frequently used instead of D_A^*. If $n = 2$, we omit the index.

There is an alternative way to define these languages as follows. Consider the congruence δ over $A \cup \bar{A}$ generated by

$$a\bar{a} \equiv \varepsilon \qquad (a \in A)$$

Then

$$D_A^* = \{w \in (A \cup \bar{A})^* \mid w \equiv \varepsilon \bmod \delta\}$$

The *class* of a word w, that is the set of all words x that are congruent to w, is denoted by $[w]_\delta$. Of course, $D_A^* = [\varepsilon]_\delta$. We often omit the subscript δ in this notation.

The *Lukasiewicz language.* Let $A = \{a, b\}$. The Lukasiewicz language is the language generated by the grammar

$$S \longrightarrow aSS + b$$

It is sometimes denoted by Ł. As we shall see below, $Ł = D_1^* b$.

2. Systems of equations

This section is devoted to an elementary presentation of systems of equations and their relation to context-free languages. Context-free languages may indeed be defined as the components of the least solution of systems of polynomial equations, whence the term "algebraic" languages introduced by Chomsky and Schützenberger [10]. The same construction was used by Ginsburg and Rice [20]. They preferred to call them ALGOL-like languages because they are "a model for the syntactic classes of the programming language ALGOL". Indeed, one says "an instruction *is*..." rather than "the symbol for instructions *derives*...".

From the methodological point of view, considering equations rather than grammars shifts the induction argument used to prove properties of languages from the number of derivations steps to the length of words. This may frequently simplify exposition, too.

The proofs of the results presented in this section are intentionally from scratch. In fact, most results can be treated differently, in at least two ways: first, they hold in a much more general framework, namely for formal power series over suitable semirings (see the chapter of Kuich[37]); next, there are general results, such as fixed-point theorems in conveniently ordered sets, that imply easily the present results. The present style of exposition was chosen to show what the minimal requirements are to make the arguments work.

The reader should notice that we never assume, in systems of equations, that the right hand sides are finite, and indeed this appears nowhere to be required. Even finiteness of the number of equations is not necessary. Next, the reader should check that all results also hold for partially commutative free monoids (this was observed already by Fliess [15]). Indeed, the argument used in most proofs is just an induction on length, and thus carries over to such monoids.

2.1 Systems

For the definition of equations, we need variables. It will be convenient to number variables. Let $V = \{X_1, \dots X_n\}$ and A be disjoint alphabets. A *system of equations* over (V, A) is a vector $P = (P_1, \dots, P_n)$ of subsets of $(V \cup A)^*$, usually written as

$$X_i = P_i \qquad i = 1, \dots, n \tag{2.1}$$

Introducing $X = (X_1, \dots, X_n)$, this can be shortened to

$$X = P$$

We frequently emphasize the dependence of the set V by writing $P_i(X)$ or $P(X)$ instead of P_i and P. An advantage of this is to yield a simple notation for substitution.

Let $L = (L_1, \dots, L_n)$ be a vector of languages over $V \cup A$. This defines a substitution as follows.

$$\begin{array}{lll}
(1) & \varepsilon(L) = \{\varepsilon\} & \\
(2) & a(L) = a & a \in A \\
(3) & X_i(L) = L_i & i = 1, \dots, n \\
(4) & uv(L) = u(L)v(L) & u, v \in (V \cup A)^* \\
(5) & Q(L) = \bigcup_{w \in Q} w(L) & Q \subset (V \cup A)^*
\end{array}$$

Observe that the last equation implies that $Q(L \cup M) = Q(L) \cup Q(M)$, where $L \cup M$ is componentwise union. A vector $L = (L_1, \dots, L_n)$ of languages over A is a *solution* if

$$L_i = P_i(L) \qquad i = 1, \dots, n$$

that is if $P_i(L)$ is obtained from $P_i(X)$ by substituting L_j to X_j in any of its occurrences. It is sometimes convenient to write $L = P(L)$ instead $L_i = P_i(L)$ for all i.

Example 2.1. 1) Consider the following system of two equations

$$\begin{aligned} X &= YX + \varepsilon \\ Y &= aXb \end{aligned}$$

Here, the variables are X, Y and the terminal alphabet is $\{a, b\}$. The vector (D_1^*, D_1) is a solution of this system, since indeed

$$\begin{aligned} D_1^* &= D_1 D_1^* + \varepsilon \\ D_1 &= a D_1^* b \end{aligned}$$

2) The system

$$\begin{aligned} X &= (aXb)^* \\ Y &= aY^*b \end{aligned}$$

has right sides that are rational sets. The vector (D_1^*, D_1) is also a solution of the second system, as it follows from elementary properties of the Dyck set. A simple formal proof will be given below.

Solutions are compared componentwise: given two vectors $L = (L_1, \ldots, L_n)$ and $M = (M_1, \ldots, M_n)$, then $L = M$ iff $L_i = M_i$ for all i, and $L \subset M$ iff $L_i \subset M_i$ for all i.

To every context-free grammar over an alphabet A, is canonically associated a polynomial system of equations (i. e. a system where the right sides are finite sets). Assume indeed that the grammar is $G = (V, P)$, with $V = \{X_1, \ldots, X_n\}$. The associated system is

$$X_i = P_i \tag{2.2}$$

with

$$P_i = \{\alpha \in (V \cup A)^* \mid (X_i, \alpha) \in P\}$$

Theorem 2.1. *Let $G = (V, P)$ be a context-free grammar over A with $V = \{X_1, \ldots, X_n\}$. Then the vector*

$$L_G = (L_G(X_1), \ldots, L_G(X_n))$$

is the least solution of the associated system.

We start with a lemma.

Lemma 2.1. *Let $M = (M_1, \ldots, M_n)$ be a solution of* (2.2), *and let $u, v \in (V \cup A)^*$ be words. If $u \xrightarrow[G]{} v$ then $v(M) \subset u(M)$.*

Proof. Indeed, if $u \longrightarrow v$, then there exists a production (X_i, α) in G, and two words x, y such that

$$u = xX_iy, \quad v = x\alpha y$$

Thus

$$u(M) = x(M)M_iy(M), \quad v(M) = x(M)\alpha(M)y(M)$$

Since $\alpha \in P_i$ and M is a solution, one has

$$\alpha(M) \subset P_i(M) = M_i$$

and consequently $v(M) \subset u(M)$. ■

Proof of the theorem. Clearly, for each $i = 1, \ldots, n$,

$$L_G(X_i) = \bigcup_{\alpha \in P_i} L_G(\alpha)$$

Now, for any word u in $(V \cup A)^*$,

$$L_G(u) = u(L_G)$$

so that the equation can be written as

$$L_G(X_i) = P_i(L_G)$$

showing that L_G is indeed a solution.

Consider next a solution $M = (M_1, \ldots, M_n)$. To show the inclusion $L_G \subset M$, let $w \in L_G(X_i)$, for some i. Then $X_i \xrightarrow{*} w$, and by the lemma (extended to derivations)

$$w(M) \subset X_i(M)$$

Since $w \in A^*$, one has $w(M) = \{w\}$, and since M is a solution, $X_i(M) = M_i$. Consequently $w \in M_i$, showing that $L_G(X_i) \subset M_i$. ∎

This theorem gives one method for computing the minimal solution of a system of equations, namely by derivations. There is another method, based on iteration. This is the fixed point approach.

Theorem 2.2. *Given a system of equations*

$$X_i = P_i \qquad i = 1, \ldots, n$$

over $V = \{X_1, \ldots, X_n\}$ and A, define a sequence $L^{(h)} = (L_1^{(h)}, \ldots, L_n^{(h)})$ of vectors of subsets of A^* by

$$L^{(0)} = (\emptyset, \ldots, \emptyset)$$
$$L^{(h+1)} = (P_1(L^{(h)}), \ldots, P_n(L^{(h)})) = P(L^{(h)})$$

and set

$$L_i = \bigcup_{h \geq 0} L_i^{(h)} \qquad i = 1, \ldots, n$$

Then the vector

$$L = (L_1, \ldots, L_n)$$

is the least solution of the system.

Proof. First,

$$L_i = \bigcup_{h \geq 0} P_i(L^{(h)}) = P_i(\bigcup_{h \geq 0} L^{(h)}) = P_i(L)$$

showing that L is indeed a solution.

Next, if $M = (M_1, \ldots, M_n)$ is any solution, then $L^{(h)} \subset M$ for all $h \geq 0$. This is clear for $h = 0$, and by induction

$$L_i^{(h+1)} = P_i(L^{(h)}) \subset P_i(M) = M_i$$ ∎

Let us remark that the basic ingredient of the proof is that P_i is "continuous" and "monotone" in the lattice $\mathfrak{P}((V \cup A)^*)^n$, for the order of componentwise inclusion (see also the chapter by Kuich [37]).

A system of equations

$$X_i = P_i \qquad i = 1, \ldots, n$$

over $V = \{X_1, \ldots, X_n\}$ and A is called

- *proper* if, for all i, one has $P_i \cap (\{\varepsilon\} \cup V) = \emptyset$,

- *strict* if, for all i, one has $P_i \subset \{\varepsilon\} \cup (V \cup A)^* A (V \cup A)^*$.

Thus, in a proper system, every word in the right side of an equation is either a terminal letter (in A) or has length at least 2. If a context-free grammar is proper, the associated system of equations is proper. In a strict system, every nonempty word in a right side contains at least one terminal letter.

A solution $L = (L_1, \ldots, L_n)$ is *proper* if $\varepsilon \notin L_i$ for all i.

Theorem 2.3. *A proper system has a unique proper solution. A strict system has a unique solution.*

Before starting the proof, let us give some examples.

Example 2.2. The equation $X = XX$ is proper. Its unique proper solution is the empty set. However, every submonoid is a solution. Thus a proper system may have more than one solution.

Example 2.3. The system

$$\begin{aligned} X &= YX + \varepsilon \\ Y &= aXb \end{aligned}$$

is neither proper nor strict. However, replacing Y by aXb in $YX + \varepsilon$, one sees that the first component of a solution is also a solution of

$$X = aXbX + \varepsilon$$

which is strict. This shows that the system has only one solution.

The system

$$\begin{aligned} X &= (aXb)^* \\ Y &= aY^*b \end{aligned}$$

is strict, so it has a unique solution. It is easily checked that

$$X = (aXb)^* = aXbX + \varepsilon$$

and

$$aXb = a(aXb)^*b$$

Thus the unique solution of this system is equal to the unique solution of the first.

Example 2.4. We claimed earlier that $Ł = D_1^*b$. Here is the proof. The Lukasiewicz language $Ł$ is the unique solution of the strict equation $X = aXX+b$, and D_1^* is (the unique) solution of the strict equation $X = aXbX+\varepsilon$. Thus $D_1^* = aD_1^*bD_1^* + \varepsilon$, and multiplying both sides by b, one gets $D_1^*b = aD_1^*bD_1^*b + b$, showing that D_1^*b is a solution of $X = aXX + b$. Since this equation has only one solution, the equality follows.

It is convenient to introduce a notation. For any $k \geq 0$, define an equivalence relation $\sim_k$ for languages $H, H' \subset A^*$ by

$$H \sim_k H' \quad \Longleftrightarrow \quad \{w \in H \mid |w| \leq k\} = \{w \in H' \mid |w| \leq k\}$$

Extend these equivalences to vectors componentwise. Then one has the following general lemma:

Lemma 2.2. *Let L and M be two solutions of a system of equations $X = P$. If*

$$L \sim_0 M \tag{2.3}$$

$$L \sim_k M \ \Rightarrow\ P(L) \sim_{k+1} P(M) \tag{2.4}$$

then $L = M$.

Proof. Since $L = P(L)$ and $M = P(M)$, the hypotheses imply that $L \sim_k M$ for all $k \geq 0$, and thus $L = M$. ■

Proof of the theorem 2.3. It suffices to show that the conditions of the lemma are fulfilled in both cases.

Consider first the case where L and M are proper solutions of the proper system $X = P$. Then by assumption $L \sim_0 M$. Assume now $L \sim_k M$, and consider any $\alpha \in P_i$ for some i. If $\alpha \in A^+$, then $\alpha(L) = \alpha(M) = \alpha$. Otherwise, there exist non empty words β, γ, such that $\alpha = \beta\gamma$. Clearly $\beta(L) \sim_k \beta(M)$ and $\gamma(L) \sim_k \gamma(M)$, and since the empty word is not in these languages, one has

$$\beta(L)\gamma(L) \sim_{k+1} \beta(M)\gamma(M)$$

Thus $P_i(L) \sim_{k+1} P_i(M)$. This proves (2.4).

Consider now the case where L and M are solutions of the strict system $X_i = P_i$ for $i = 1, \ldots, n$. Since $\varepsilon \in L_i$ for some i iff $\varepsilon \in P_i$, one has $L \sim_0 M$. Next, as before assume $L \sim_k M$, and consider any $\alpha \in P_i$ for some i. If $\alpha \neq \varepsilon$, then $\alpha = \beta a \gamma$ for words β, γ and a letter $a \in A$. Again, $\beta(L) \sim_k \beta(M)$ and $\gamma(L) \sim_k \gamma(M)$, and since a is a terminal letter, this implies that $\alpha(L) \sim_{k+1} \alpha(M)$. This proves (2.4). ■

As we have already seen, a system may have a unique solution even if it is neither proper nor strict. Stronger versions of the above theorem exist. For instance, call a system of equations

$$X = P(X)$$

weakly proper (resp. *weakly strict*) if there is an integer k such that the system

$$X = P^k(X)$$

is proper (resp. strict).

Corollary 2.1. *A weakly strict (weakly proper) system has a unique (a unique proper) solution.*

Proof. Let indeed L be a solution of $X = P(X)$. Then $L = P^k(L)$, showing that L is solution of $X = P^k(X)$. Hence the solution of $X = P(X)$ is unique. ∎

Observe that, if L is the solution of $X = P^k(X)$, then it is also the solution of the system $X = P(X)$. This may provide an easy way to compute the solution.

Example 2.5. Consider the system $X = P(X)$ given by

$$\begin{aligned} X &= YX + \varepsilon \\ Y &= aXb \end{aligned}$$

Replacing P by P^2, one gets

$$\begin{aligned} X &= aXbYX + aXb + \varepsilon \\ Y &= aYXb + ab \end{aligned}$$

which is not proper but strict. Hence the system is weakly strict.

2.2 Resolution

One popular method for resolution of systems of equations is Gaussian elimination. Consider sets $X = \{X_1, \ldots, X_n\}$ and $Y = \{Y_1, \ldots, Y_m\}$ of variables.

Theorem 2.4. *For any system of equations*

$$\begin{aligned} X &= P(X, Y) \\ Y &= Q(X, Y) \end{aligned} \tag{2.5}$$

over $(X \cup Y, A)$, *let* L'_Y *be a solution of the system of equations*

$$Y = Q(X, Y)$$

over $(Y, A \cup X)$, *and let* L_X *be a solution of the system of equations*

$$X = P(X, L'_Y)$$

over (X, A). *Then* $(L_X, L'_Y(L_X))$ *is a solution of* (2.5).

Proof. Let indeed $L'_Y = L'_Y(X)$ be a solution of the system $Y = Q(X, Y)$. For any vector $L = (L_1, \ldots, L_n)$ of languages over A, one has

$$L'_Y(L) = Q(L, L'_Y(L)) \tag{2.6}$$

by substitution. Next, let L_X be a solution of

$$X = P(X, L'_Y(X)) \tag{2.7}$$

and set $L_Y = L'_Y(L_X)$. Then (L_X, L_Y) is a solution of (2.5) since $L_X = P(L_X, L_Y)$ by (2.7) and $L_Y = Q(L_X, L_Y)$ by (2.6). ∎

A special case is "lazy" resolution. This means that some variables, or even some occurrences of variables or factors in the right sides are considered as "fixed", the obtained system is solved, and the solution is substituted in the "fixed" part. More precisely,

Proposition 2.1. *The two systems*

$$\begin{array}{l} X = P(X,Y) \\ Y = Q(X) \end{array} \qquad \textit{and} \qquad \begin{array}{l} X = P(X,Q(X)) \\ Y = Q(X) \end{array}$$

have same sets of solutions. ■

As an example, consider the equation $X = aXX + b$, that we write as

$$\begin{aligned} X &= YX + b \\ Y &= aX \end{aligned}$$

The first equation is equivalent to $X = Y^*b$, thus the equations $X = aXX+b$ and $X = (aX)^*b$ have the same solution.

2.3 Linear systems

Left or right linear systems of equations are canonically associated to finite automata. The general methods take here a special form. A system of equations

$$X_i = P_i(X), \qquad i = 1, \ldots, n$$

over (V, A) is *right linear* (resp. *left linear*) if $P_i \subset A^*V \cup A^*$ (resp. $P_i \subset VA^* \cup A^*$). If furthermore it is strict (resp. proper), then $P_i \subset A^+V \cup A^*$ (resp. $P_i \subset A^+V \cup A^+$). A (right) linear system may be written as

$$X_i = \sum_{j=1}^{n} R_{i,j}X_j + S_i \qquad i = 1, \ldots, n \tag{2.8}$$

where $R_{i,j} \subset A^*$, $S_i \subset A^*$. These sets are the *coefficients* of the system. One may also write

$$X = RX + S$$

by introducing a matrix $R = (R_{i,j})$ and a vector $S = (S_i)$.

Given a finite automaton with state set $Q = \{1, \ldots, n\}$, denote by $R_{i,j}$ the set of labels of edges from state i to state j, and set

$$S_i = \begin{cases} \{\varepsilon\} & \text{if } i \text{ is a final state} \\ \emptyset & \text{otherwise} \end{cases}$$

Then it is easily verified that the least solution of the system (2.8) is the vector $(L_1, \ldots, L_n)$, where L_i is the set of words recognized with initial state i.

Theorem 2.5. *The components of the solution of a strict linear system are in the rational closure of the set of coefficients of the system.*

Proof. There are several proofs of this result. The maybe simplest proof is to consider an alphabet $B = \{r_{i,j} \mid 1 \leq i,j \leq n\} \cup \{s_i \mid 1 \leq i \leq n\}$ and to consider the system obtained in replacing each $R_{i,j}$ by $r_{i,j}$ and similarly for the S_i. Build an automaton with state set $\{0, 1, \ldots, n\}$, having an edge labeled $r_{i,j}$ from state i to state j for $1 \leq i,j \leq n$ and an edge labeled s_i from state i to the unique final state 0. The component L_i of the solution of the system is the set of label of paths from state i to state 0, and therefore is a rational set over B. To get the solution of the original system, it suffices to substitute the sets $R_{i,j}$ and S_i to the corresponding variables.

An equivalent formulation is to say that the vector

$$L = R^* S$$

is the solution, where

$$R^* = \bigcup_{m \geq 0} R^m .$$

One way to solve the original set of equations is to use Gaussian elimination (also called Arden's lemma in the linear case). One rewrites the last equation of the system as

$$X_n = R^*_{n,n} \left(\sum_{j=1}^{n-1} R_{n,j} X_j + S_j \right)$$

and substitutes this expression in the remaining equations.

Another way is to proceed inductively, and to compute the transitive closure R^* from smaller matrices, using the formula

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^* = \begin{pmatrix} (A + BD^*C)^* & A^*B(D + CA^*B)^* \\ D^*C(A + BD^*C)^* & (D + CA^*B)^* \end{pmatrix}$$

provided A and D are square matrices. ∎

A system (2.8) is called *cycle-free* if none of the diagonal coefficients of the matrix

$$R + R^2 + \cdots + R^n$$

contains the empty word. The terminology is from graph theory : consider the graph over $\{1, \ldots, n\}$ with an edge from i to j iff $\varepsilon \in R_{i,j}$. Then this graph is cycle-free iff the system is. In fact, cycle-free systems are precisely weakly strict right linear systems. Indeed, the graph is cycle-free iff there is no path of length k for $k > n$. This is equivalent to say that in the matrix R^k, none of the coefficients contains the empty word. Thus one has

Proposition 2.2. *A cycle-free (right or left) linear system has a unique solution.* ∎

2.4 Parikh's theorem

In this section, we prove Parikh's theorem [43]. Our presentation follow [44]. A more general result is given in Kuich's chapter [37]. As already mentioned, all results concerning systems of equations, provided they make sense (e.g. Greibach normal form makes no sense in free commutative monoids) hold also for free partially commutative monoids, since the only argument used is induction on length. Two special cases of partially commutative free monoids are the free monoid and the free commutative monoid. Context-free sets in the latter case are described by Parikh's theorem:

Theorem 2.6. *Any context-free set in the free commutative monoid is rational.*

An equivalent formulation is that the set of Parikh vectors of a context-free language is semi-linear. Indeed, let A be an alphabet, and denote by $A^{\oplus}$ the free commutative monoid over A. There is a canonical mapping α from A^* onto $A^{\oplus}$ that associates, to a word w, the element $\Pi_{a\in A} a^{|w|_a}$ in $A^{\oplus}$, where $|w|_a$ is the number of occurrences of the letter a in A.

Rational sets are defined in $A^{\oplus}$ as they are in any monoid: they constitute the smallest family of languages containing the empty set, singletons, and closed under union, product and star. Here, product is the product in $A^{\oplus}$ of course. Because of commutativity, there are special relations, namely

$$(X \cup Y)^* = X^*Y^*, \quad (X^*Y)^* = \{\varepsilon\} \cup X^*Y^*Y$$

Using these, on gets easily the following

Proposition 2.3. *In the free commutative monoid $A^{\oplus}$, every rational set has star-height at most* 1. ∎

Proof of the theorem. Consider first the case of a single (strict) equation

$$X = P(X)$$

where $P(X)$ is any rational subset of $(A \cup X)^{\oplus}$. This equation may be rewritten as

$$X = R(X)X + S$$

where $S = P(X) \cap A^{\oplus}$, and $R(X)$ is a rational subset of $(A \cup X)^{\oplus}$. The set $G = R(S)$ is rational, and we show that G^*S is the (rational) solution of the equation.

Consider indeed two subsets K, M of A^*, and set $P = K^*M$. For every $w \in (A \cup X)^{\oplus}$ containing at least one occurrence of X, one has the equality

$$w(P) = w(M)K^*$$

because the set K^* can be "moved" to the end of the expression by commutativity, and $K^*K^* = K^*$. As a consequence, for every word $w \in (A \cup X)^{\oplus}$, one gets $w(P)P = w(M)P$. Thus in particular for $P = G^*S$,

$$S + R(P)P = S + R(S)P = S + GG^*S = G^*S$$

If the system has more than one equation, then it is solved by Gaussian elimination. ∎

Example 2.6. Consider the equation

$$X = aXX + b$$

The set $R(X)$ of the proof reduces to aX, and the solution is $(ab)^*b = \{a^n b^{n+1} \mid n \geq 0\}$.

3. Normal forms

In this section, we present three normal forms of context-free grammars. The two first ones are the Chomsky normal form and the Greibach normal form. They are often used to get easier proofs of results about context-free languages. The third normal form is the operator normal form. It is an example of a normal form that has been used in the syntactical analysis.

3.1 Chomsky normal form

A context-free grammar $G = (V, P)$ over the terminal alphabet A is in *weak Chomsky normal form* if each nonterminal rule has a right member in $V^\star$ and each terminal rule has a right member in $A \cup \{\varepsilon\}$. It is in *Chomsky normal form* if it is in Chomsky normal form and each right member of a nonterminal rule has length 2.

Theorem 3.1. [28, 9] *Given a context-free grammar, an equivalent context-free grammar in Chomsky normal form can effectively be constructed.*

Proof. The construction is divided into three steps. In the first step, the original grammar is transformed into a new equivalent grammar in weak Chomsky normal form. In the second step, we transform the grammar just obtained so that the length of a right member of a rule is at most 2. In the last step, we get rid of the nonterminal rules with a right member of length 1 (that is to say in V).

Step 1 : To each terminal letter $a \in A$, we associate a new variable X_a. In all the right members of the rules of the original grammar, we replace each occurrence of the terminal letters a by the new variable X_a. Finally, we add to the grammar so obtained the set of rules $X_a \longrightarrow a$. Clearly, the resulting grammar so constructed is in weak Chomsky normal form and is equivalent to the original one.

Step 2 : We now introduce a new set of variables designed to represent the product of two old variables. More precisely, to each pair $(X, Y) \in V \times V$, we associate a new variable $\langle XY \rangle$. We then construct a new grammar by replacing any product of three or more old variables $XYZ \cdots T$ by $\langle XY \rangle Z \cdots T$. Then we add all the rules $\langle XY \rangle \longrightarrow XY$. This reduces the maximal length of nonterminal rules by 1. This process is repeated until the maximum length of any right member is 2.

Step 3 : We finally get rid of nonterminal rules with a right member in V. This is achieved in the same usual way than the one used to get a proper grammar from a general one. ∎

3.2 Greibach normal forms

A context-free grammar $G = (V, P)$ over the terminal alphabet A is in *Greibach normal form* iff each rule of the grammar rewrites a variable into a word in $AV^\star$. In particular, the grammar is proper and each terminal rule rewrites a variable in a terminal letter.

It is in *quadratic Greibach normal form* iff it is in Greibach normal form and each right member of a rule of G contains at most 2 variables.

It is in *double Greibach normal form* iff each right member of the rules of G are in $AV^\star A \cup A$. In particular, a terminal rule rewrites a variable in a letter or in a word of length 2.

It is in *cubic double Greibach normal form* (resp. in *quadratic double Greibach normal form* iff it is in double Greibach normal form and each right member of a rule contains at most 3 variables (resp. at most 2 variables).

The fact that any proper context-free grammar G can be transformed in an equivalent grammar G' in Greibach normal form is a classical result [28]. However, the fact that the same result holds with G' in quadratic Greibach normal form is more rarely presented. Nearly never proved is the same result with G' in quadratic double normal form. Hence, we show how such equivalent grammars can effectively be constructed.

Theorem 3.2. *Given a proper context-free grammar G, an equivalent context free grammar in quadratic Greibach normal form can effectively be constructed from G.*

A weaker similar result has originally been proved by Greibach [24]: she showed that, given a proper context-free grammar, an equivalent context-free grammar in Greibach normal form can effectively be constructed. The additional statement stating that this grammar can be in *quadratic* Greibach normal form was proved later by Rosenkrantz [45]. We sketch here the proof he gave; we will see below an alternative proof.

Sketch of the construction:

We may assume that the grammar is proper and in Chomsky normal form, that is that each right-hand side is in $A \cup V^2$. Consider the associated system of equations

$$X_i = P_i \qquad i = 1, \ldots, n$$

This may be written as

$$X = XR + S$$

where

$$S_i = P_i \cap A$$

and

$$R_{j,i} = X_j^{-1} P_i$$

Using lazy evaluation, this system is equivalent to

$$X = SR^*$$

and since

$$R^* = RR^* + I$$

one has

$$\begin{aligned} X &= SY \\ Y &= RY + I \end{aligned}$$

where $Y = (Y_{j,i})$ is a new set of n^2 variables. Observe that each $R_{j,i}$ is a subset of V. Thus, using the system $X = SY$, each $R_{j,i}$ can be replaced by the set

$$\hat{R}_{j,i} = \sum_{X_\ell \in R_{j,i}} (SY)_\ell$$

and the whole system is equivalent to

$$\begin{aligned} X &= SY \\ Y &= \hat{R}Y + I \end{aligned}$$

where $\hat{R} = (\hat{R}_{j,i})$. In order to get the quadratic Greibach normal form, it suffices to eliminate the ε-rules. This is done in the usual way. ■

Theorem 3.3. *Given a proper context-free grammar G, an equivalent context free grammar in quadratic double Greibach normal form can effectively be constructed from G.*

This result has been proved by Hotz [31]. We follow his proof. It should be noted that the same technique allows to give an alternative proof of the previous theorem 3.2.

The proof of theorem 3.3 turns out to be a complement to the proof of

Theorem 3.4. *Given a proper context-free grammar G, an equivalent context free grammar in cubic double Greibach normal form can effectively be constructed from G.*

The construction of the desired grammar is decomposed into four steps. The two first ones will lead to an equivalent grammar in quadratic Greibach normal form. The last two ones complete the construction of an equivalent grammar in quadratic double Greibach normal form.

Let $G = (V, P)$ be a proper context-free grammar in weak Chomsky normal form over the terminal alphabet A.

Step 1 (construction of the set of new variables needed):

To each variable $X \in V$, we associate the sets

$$L(X) = \{am \in AV^\star \mid X \xrightarrow[\ell]{\star} \alpha \longrightarrow am \quad \alpha \in V^*\}$$
$$R(X) = \{ma \in V^\star A \mid X \xrightarrow[r]{\star} \alpha \longrightarrow ma \quad \alpha \in V^*\}$$

The idea is to construct a new grammar including rules the $X \longrightarrow am$ for each $am \in L(X)$. The difficulty comes from the fact that the sets $L(X)$ are infinite. This difficulty can be overcome using the fact these sets are rational. The sets $R(X)$ will only be used to get the double Greibach normal form. Formally, to each variable $X \in V$ and to each terminal letter $a \in A$, we associate the sets

$$L(a, X) = a^{-1}L(X) = \{m \in V^\star \mid X \xrightarrow[\ell]{\star} \alpha \longrightarrow am \quad \alpha \in V^*\}$$
$$R(X, a) = R(X)a^{-1} = \{m \in V^\star \mid X \xrightarrow[r]{\star} \alpha \longrightarrow ma \quad \alpha \in V^*\}$$

Clearly, each $L(a, X)$ and each $R(X, a)$ is a rational language over V since $L(X)$ and $R(X)$ are rational: to get a word in $L(X)$, we look at leftmost derivations in the original grammar. Then, such a derivation can be decomposed in a first part where the obtained words all lie in $V^\star$. The second part consists in the last step where the leftmost variable is derived in a terminal letter a. In this process, we then always derive the leftmost variable of the sentential forms. So, this derivation is obtained by using the grammar as if it were left linear. Hence, the set of words so obtained forms a rational set. It then follows immediately that each $L(a, X)$ is rational too.

A similar proof using right linear grammars shows that each $R(X, a)$ is rational.

Next, define two families $\mathcal{L}$ and $\mathcal{R}$ of languages by

$$\mathcal{L} = \{L(a, X) \mid a \in A, X \in V\}, \quad \mathcal{R} = \{R(X, a) \mid a \in A, X \in V\}.$$

We then define $\mathcal{H}$ as the closure of $\mathcal{L} \bigcup \mathcal{R}$ under the right and left quotients by a letter of V. Since each language in $\mathcal{L} \bigcup \mathcal{R}$ is rational, this gives raise to a finite number of new regular languages over V. Thus, the family $\mathcal{H}$ is finite.

The idea is now to use the languages in $\mathcal{H}$ as variables in the grammar to be constructed. The set of new variables will be denoted like this family of languages, that is, an element $L \in \mathcal{H}$ will denote both the language and the new variable.

Example 3.1. Let $G = (V, P)$ be the following grammar in weak Chomsky normal form :

$$\begin{aligned} S &\longrightarrow SXSS \ + \ b \\ X &\longrightarrow a \end{aligned}$$

We can now look for the family $\mathcal{L}$. Since

$$L(a,S) = \emptyset \quad L(b,S) = (XSS)^\star = L_0 \quad L(a,X) = \{\varepsilon\} = E \quad L(b,X) = \emptyset$$

$\mathcal{L}$ is formed of the three languages $\{\emptyset, L_0, E\}$.

Similarly, $\mathcal{R} = \{\emptyset, L_1, E\}$ because

$$R(S,a) = \emptyset \quad R(S,b) = (SXS)^\star = L_1 \quad R(X,a) = \{\varepsilon\} = E.$$

Thus, $\mathcal{L} \bigcup \mathcal{R} = \{\emptyset, L_0, L_1, E\}$. From this family, we derive the family $\mathcal{H}$ by closing $\mathcal{L} \bigcup \mathcal{R}$ under left and right quotient by a letter in V. Here are the new languages that appear :

$$\begin{array}{llllll}
X^{-1}L_0 = SS(XSS)^\star & = L_2 & \qquad & L_0S^{-1} = (XSS)^\star XS & = L_3 \\
S^{-1}L_1 = XS(SXS)^\star & = L_3 & & L_1S^{-1} = (SXS)^\star SX & = L_4 \\
S^{-1}L_2 = S(XSS)^\star & = L_5 & & L_2S^{-1} = S(SXS)^\star & = L_6 \\
X^{-1}L_3 = (SSX)^\star S & = L_6 & & L_3S^{-1} = (XSS)^\star X & = L_7 \\
S^{-1}L_4 = (XSS)^\star X & = L_7 & & L_4X^{-1} = S(XSS)^\star & = L_5 \\
S^{-1}L_5 = (XSS)^\star & = L_0 & & L_5S^{-1} = (SXS)^\star & = L_1 \\
S^{-1}L_6 = (SXS)^\star & = L_1 & & L_6S^{-1} = (SSX)^\star & = L_8 \\
X^{-1}L_7 = (SSX)^\star & = L_8 & & L_7X^{-1} = (XSS)^\star & = L_0 \\
S^{-1}L_8 = SX(SSX)^\star & = L_4 & & L_8X^{-1} = (SSX)^\star SS & = L_2
\end{array}$$

Hence, the family $\mathcal{H}$ contains 11 languages: the languages $L_0, \ldots, L_8$, the language $E = \{\varepsilon\}$ and the empty set $\emptyset$. (In the above computations, we have omitted all empty quotients.) Among these languages, E, L_0, L_1 and L_8 contain the empty word.

Step 2 (Construction of an equivalent grammar in quadratic Greibach normal form)

The new grammar has the set of variables $V \cup \mathcal{H}$, and the following rules:

(i) Each terminal rule of the original grammar is a terminal rule of the new grammar.

(ii) To each variable $X \in V$ of the original grammar is associated the (finite) set of rules $X \longrightarrow aL$ for each $a \in A$, with $L = L(a,X) \in \mathcal{H}$. The rules so created have all their right members in $A\mathcal{H}$.

(iii) Each new variable $L \in \mathcal{H}$ gives raise to the finite set of rules $L \longrightarrow XL'$ for $X \in V$ with $L' = X^{-1}L \in \mathcal{H}$ and to the rule $L \longrightarrow \varepsilon$ if $\varepsilon \in L$. Each such rule has its right member in $V\mathcal{H} \cup \{\varepsilon\}$.

(iv) In each new non ε-rule added just above, the leftmost variable in V is replaced by the right members generated in step (ii); since these right members are in $A\mathcal{H}$, the rules so obtained have all their right members in $A\mathcal{H}\mathcal{H}$.

Hence, the grammar so obtained is almost in quadratic Greibach normal form: each right member is in $A\mathcal{H}\mathcal{H} \cup A \cup \{\varepsilon\}$.

To obtain such a normal form, it suffices to complete a final operation eliminating the ε-rules. We do this in the usual way, that is we replace any occurrence of a variable L giving the empty word by $(L \cup \{\varepsilon\})$ in the right members and erase the ε-rules.

The fact that the new grammar is equivalent to the original one is immediate: it suffices to look at the two grammars as systems of equations. So far, we have proved Theorem 3.2.

Example 3.2. (continued) The rules $S \longrightarrow SXSS + b$ give raise to the new rules $S \longrightarrow bL_0 + bE$. The last rule $X \longrightarrow a$ gives raise to $X \longrightarrow aE$.

The new set of variables gives raise to

$$\begin{array}{ll}
L_0 \longrightarrow XL_2 + \varepsilon & L_1 \longrightarrow SL_3 + \varepsilon \\
L_2 \longrightarrow SL_5 & L_3 \longrightarrow XL_6 \\
L_4 \longrightarrow SL_7 & L_5 \longrightarrow SL_0 \\
L_6 \longrightarrow SL_1 & L_7 \longrightarrow XL_8 \\
L_8 \longrightarrow SL_4 + \varepsilon & E \longrightarrow \varepsilon
\end{array}$$

Replacing X by aE and S by $bE + bL_0$, the new grammar becomes

$$\begin{array}{ll}
S \longrightarrow bL_0 + bE & X \longrightarrow aE \\
L_0 \longrightarrow aEL_2 + \varepsilon & L_1 \longrightarrow bEL_3 + bL_0L_3 + \varepsilon \\
L_2 \longrightarrow bEL_5 + bL_0L_5 & L_3 \longrightarrow aEL_6 \\
L_4 \longrightarrow bEL_7 + bL_0L_7 & L_5 \longrightarrow bEL_0 + bL_0L_0 \\
L_6 \longrightarrow bEL_1 + bL_0L_1 & L_7 \longrightarrow aEL_8 \\
L_8 \longrightarrow bEL_4 + bL_0L_4 + \varepsilon & E \longrightarrow \varepsilon
\end{array}$$

This is the desired intermediate grammar obtained after step (iv). To obtain the quadratic Greibach normal form, we replace everywhere E, L_0, L_1 and L_8 by themselves plus the empty word in the right members and suppress the ε-rules. Then we get the following grammar (to be compared to the one obtained with Rosenkrantz's method):

$$\begin{array}{ll}
S \longrightarrow bL_0 + b & X \longrightarrow a \\
L_0 \longrightarrow aL_2 & L_1 \longrightarrow bL_3 + bL_0L_3 \\
L_2 \longrightarrow bL_5 + bL_0L_5 & L_3 \longrightarrow aL_6 \\
L_4 \longrightarrow bL_7 + bL_0L_7 + bL_7 & L_5 \longrightarrow bL_0 + bL_0L_0 + b \\
L_6 \longrightarrow bL_1 + bL_0L_1 + bL_0 + b & L_7 \longrightarrow aL_8 + a \\
L_8 \longrightarrow bL_4 + bL_0L_4 &
\end{array}$$

Note that this grammar is not reduced. The only useful variables are S, L_0, L_2 and L_5. The seemingly useless variables and rules will appear to be useful later. Note too that E disappeared because, when ε is removed from the language, E becomes empty.

The next two steps will be devoted to the proof of theorem 3.3.

Step 3 (Construction of an equivalent grammar in cubic double Greibach normal form)

We work on the grammar just obtained above. Each nonterminal rule of this grammar ends with a variable in $\mathcal{H}$. A variable now generates a language that is proper. Thus, the language is not necessarily in $\mathcal{H}$ (considered as a family of languages) because the empty word may be missing. However, the set $\mathcal{H}$ (considered as a set of variables) remains the same. Each variable now generates the associated language of $\mathcal{H}$ up to the empty word.

We first proceed to the same operations as in step 2, using right quotients instead of left ones. This operation is presented below in a slightly different way than we did in step 2. Precisely,

(i) For each language $L \in \mathcal{H}$, the set LX^{-1} is now a language of the family $\mathcal{H}$ up to the empty word. So, each L in $\mathcal{H}$ can be described as the union of $L'X$ for each $X \in V$ with $L' = LX^{-1}$, completed by X as soon as L' contains the empty word. We do this for each L.
Each language generated by a variable $X \in V$ is proper. Hence, in the expression above, X can be replaced by the union of all the Ra for each $a \in A$, with R the variable associated to the language $R(X,a)$. Again, this union has to be completed by a as soon as the language $R(X,a)$ contains the empty word.
This gives a system of equations where each $L \in \mathcal{H}$ is a sum of terms in $\mathcal{H}\mathcal{H}A \cup \mathcal{H}A \cup A$.

(ii) We now go back to the grammar in quadratic normal form resulting from step 2, and replace each rightmost variable of the nonterminal rules in the grammar by the expression in the system obtained in step (i). We thus obtain an equivalent grammar where the nonterminal rules have a terminal letter as rightmost symbol. It should be noted that, in the resulting grammar, the number of variables is increased by at most one in each rule, so that the grammar is cubic. Hence, the so obtained grammar is in cubic double Greibach normal form.

Example 3.3. (continued) The first identities are directly derived from the right quotients computed before. They are

$$\begin{array}{ll}
L_0 = L_3S & L_1 = L_4S \\
L_2 = L_6S & L_3 = L_7S \\
L_4 = L_5X & L_5 = L_1S + S \\
L_6 = L_8S + S & L_7 = L_0X + X \\
L_8 = L_2X + X &
\end{array}$$

Replacing now each S by $L_1b + b$ and each X by a, we obtain

$$\begin{array}{ll}
L_0 \longrightarrow L_3L_1b + L_3b & L_1 \longrightarrow L_4L_1b + L_4b \\
L_2 \longrightarrow L_6L_1b + L_6b & L_3 \longrightarrow L_7L_1b + L_7b \\
L_4 \longrightarrow L_5a & L_5 \longrightarrow L_1L_1b + L_1b \\
L_6 \longrightarrow L_8L_1b + L_8b + b & L_7 \longrightarrow L_0a + a \\
L_8 \longrightarrow L_2a + a &
\end{array}$$

Going back to the grammar obtained at the end of step 2, we replace in it each rightmost variable by the finite sum so obtained, giving raise to :

$$\begin{array}{l}
S \longrightarrow bL_3L_1b + bL_3b + b \\
X \longrightarrow a \\
L_0 \longrightarrow aL_6L_1b + aL_6b \\
L_1 \longrightarrow bL_7L_1b + bL_7b + bL_0L_7L_1b + bL_0L_7b \\
L_2 \longrightarrow bL_1L_1b + bL_1b + bb + bL_0L_1L_1b + bL_0L_1b + bL_0b \\
L_3 \longrightarrow aL_8L_1b + aL_8b + aL_1b + ab \\
L_4 \longrightarrow bL_0a + ba + bL_0L_0a \\
L_5 \longrightarrow bL_3L_1b + bL_3b + bL_0L_3L_1b + bL_0L_3b + b \\
L_6 \longrightarrow bL_4L_1b + bL_4b + bL_0L_4L_1b + bL_0L_4b \\
L_7 \longrightarrow aL_2a + aa + a \\
L_8 \longrightarrow bL_5a + bL_0L_5a
\end{array}$$

The steps 1, 2 and 3 allow thus to transform any context-free grammar in weak Chomsky normal form in an equivalent grammar in cubic double Greibach normal form, which proves Theorem 3.4. ∎

Step 4 (Construction of an equivalent grammar in quadratic double Greibach normal form)

We use here essentially the same technique of grouping variables that was previously used to derive Chomsky normal form from weak Chomsky normal form. It should be also noted that this technique can be used to transform a grammar in Greibach normal form into quadratic Greibach normal form.

In the grammar obtained in the previous step, no variable of V appears in the right member of a rule. Moreover, any variable of $\mathcal{H}$ represents, up to the empty word, the corresponding language. In particular, a language $L \in \mathcal{H}$ can be described by

- a left quotient description given by the rules of the grammar in quadratic Greibach normal form obtained in step 2.
- a right quotient description obtained in the same way. It is the intermediate description used in step 3 just before transforming the grammar in an equivalent one in double Greibach normal form.

We now enlarge the family $\mathcal{H}$ by adding the family $\mathcal{HH}$. To this new family of languages is associated a new set of variables W. It should be noted that, each $Y \in W$ represents a product $L \cdot L' \in \mathcal{HH}$. Hence, replacing L by its left quotient description, and L' by its right quotient description, we get a description of each $Y \in W$ as a finite union of terms in $A\mathcal{HHHH}A \cup A\mathcal{HHH}A \cup A\mathcal{HH}A \cup A\mathcal{H}A \cup AA$.

Each product of four elements of $\mathcal{H}$ can be replaced by a product of two elements of W ; similarly, any product of three elements of $\mathcal{H}$ can be replaced by the product of an element in W by an element of $\mathcal{H}$ (or just the contrary as well).

Then using this transformation in the right members of the rules of the grammar in cubic double Greibach normal form and adding the new rules induced by the representation of variables in W just obtained, we get an equivalent grammar which is now in quadratic double Greibach normal form.

Example 3.4. (end) The family W is formed of elements denoted $\langle LL' \rangle$ for $L, L' \in \mathcal{H}$. We first make quadratic the rules of the above obtained grammar by introducing, when necessary, some of our new variables.

$$\begin{aligned}
S &\longrightarrow bL_3L_1b + bL_3b + b \\
X &\longrightarrow aL_6L_1b + aL_6b + a \\
L_1 &\longrightarrow bL_7L_1b + bL_7b + b\langle L_0L_7\rangle L_1b + bL_0L_7b \\
L_2 &\longrightarrow bL_1L_1b + bL_1b + bb + b\langle L_0L_1\rangle L_1b + bL_0L_1b + bL_0b \\
L_3 &\longrightarrow aL_8L_1b + aL_8b + aL_1b + ab \\
L_4 &\longrightarrow bL_0a + ba + bL_0L_0a \\
L_5 &\longrightarrow bL_3L_1b + bL_3b + b\langle L_0L_3\rangle L_1b + bL_0L_3b + b \\
L_6 &\longrightarrow bL_4L_1b + bL_4b + b\langle L_0L_4\rangle L_1b + bL_0L_4b \\
L_7 &\longrightarrow aL_2a + aa + a \\
L_8 &\longrightarrow bL_5a + bL_0L_5a
\end{aligned}$$

Doing this, we have introduced the four new variables $\langle L_0L_1\rangle$, $\langle L_0L_3\rangle$, $\langle L_0L_4\rangle$ and $\langle L_0L_7\rangle$. Rather than computing the descriptions of all the elements $\langle L_iL_j\rangle$, we will compute those needed as soon as they appear.

So we begin by computing the description of $\langle L_0L_1\rangle$: for this we use

$$L_0 = aL_2 \quad L_1 = L_4L_1b + L_4b$$

which gives raise to the rules

$$\langle L_0L_1\rangle \longrightarrow a\langle L_2L_4\rangle L_1b + aL_2L_4b$$

Going on this way, we get the (huge) equivalent grammar in quadratic double Greibach normal form:

$$\begin{array}{ll}
S & \longrightarrow bL_3L_1b + bL_3b + b \\
X & \longrightarrow a \\
L_0 & \longrightarrow aL_6L_1b + aL_6b + a \\
L_1 & \longrightarrow bL_7L_1b + bL_7b + b\langle L_0L_7\rangle L_1b + bL_0L_7b \\
L_2 & \longrightarrow bL_1L_1b + bL_1b + bb + b\langle L_0L_1\rangle L_1b + bL_0L_1b + bL_0b \\
L_3 & \longrightarrow aL_8L_1b + aL_8b + aL_1b + ab \\
L_4 & \longrightarrow bL_0a + ba + bL_0L_0a \\
L_5 & \longrightarrow bL_3L_1b + bL_3b + b\langle L_0L_3\rangle L_1b + bL_0L_3b + b \\
L_6 & \longrightarrow bL_4L_1b + bL_4b + b\langle L_0L_4\rangle L_1b + bL_0L_4b \\
L_7 & \longrightarrow aL_2a + aa + a \\
\langle L_0L_1\rangle & \longrightarrow a\langle L_2L_4\rangle L_1b + aL_2L_4b \\
\langle L_0L_3\rangle & \longrightarrow a\langle L_2L_7\rangle L_1b + aL_2L_7b \\
\langle L_0L_4\rangle & \longrightarrow aL_2L_5a \\
\langle L_0L_7\rangle & \longrightarrow aL_2L_0a + a \\
\langle L_2L_4\rangle & \longrightarrow b\langle L_0L_5\rangle L_5a + bL_5L_5a \\
\langle L_2L_7\rangle & \longrightarrow b\langle L_0L_5\rangle L_0a + bL_0L_5a + bL_5L_0a + bL_5a \\
\langle L_0L_5\rangle & \longrightarrow a\langle L_2L_1\rangle L_1b + aL_2L_1b + aL_2b \\
\langle L_2L_1\rangle & \longrightarrow b\langle L_0L_5\rangle\langle L_4L_1\rangle b + b\langle L_0L_5\rangle L_4b + bL_5\langle L_4L_1\rangle b + bL_5L_4b \\
\langle L_4L_1\rangle & \longrightarrow ab\langle L_0L_7\rangle\langle L_4L_1\rangle b + b\langle L_0L_7\rangle L_4b + bL_7\langle L_4L_1\rangle b + bL_7L_3b
\end{array}$$

Remark 3.1. The only useless variable is now X.

3.3 Operator normal form

We present here another classical normal form, namely the operator normal form. A context-free grammar G over the terminal alphabet A is in *operator normal form* if no right member of a rule contains two consecutive variables. This normal form has been introduced for purposes from syntactical analysis. For these grammars, an operator precedence can be defined which is inspired of the classical precedence relations of usual arithmetic operators. From a general point of view, the following holds :

Theorem 3.5. [28, 16] *Given a context-free language, an equivalent context-free grammar in operator normal form can effectively be constructed.*

Proof. It is very easy. Given a grammar G in Chomsky normal form, to each pair of a terminal letter a and of a variable X is attached a new variable X_a designed to generate the set of words u such that X generates ua, that is to say $X_a = Xa^{-1}$. So, each language $L_G(X)$ is exactly the sum over A of all the languages $L_{X_a}a$, sum completed by $\{\varepsilon\}$ as soon as L_X contains ε. Identifying L_X and X, this can be written:

$$X = (\bigcup_{a\in A} X_a a) \cup (\{\varepsilon\} \cap X) \tag{3.1}$$

In the right members of the original grammar, we now replace each occurrence of the variables X by the right hand side of equation (3.1). This gives raise to a set of rules say P_1. Finally, we add the rules $X_a \longrightarrow \alpha$ for $X \longrightarrow \alpha a \in P_1$. This gives raise to a new grammar which is equivalent to the original one and is in operator normal form. Note that this new grammar may be neither proper nor reduced. ∎

Example 3.5. Consider the grammar given by the two rules

$$S \longrightarrow aSS + b.$$

We introduce two new variables S_a and S_b. The set of rules in P_1 is

$$S \longrightarrow aS_aaS_aa + aS_aaS_bb + aS_bbS_aa + aS_bbS_bb + b.$$

We then add the rules

$$S_a \longrightarrow aS_aaS_a + aS_bbS_a \qquad S_b \longrightarrow aS_aaS_b + aS_bbS_b + \varepsilon.$$

and get the desired grammar.

If we reduce the grammar, we note that the variable S_a is useless. So, we get the grammar

$$S \longrightarrow aS_bbS_bb + b \quad S_b \longrightarrow aS_bbS_b + \varepsilon.$$

If we need a proper grammar in operator normal form, we just apply the usual algorithm to make it proper.

Remark 3.2. The theory of *grammar forms* [13] develops a general framework for defining various similar normal forms. These are defined through patterns like $VAV + A$ indicating that the right members have to lie in $VAV \cup A$. From this point of view, the various normal forms presented above appear as particular instances of a very general situation (see [5]).

4. Applications of the Greibach normal form

4.1 Shamir's theorem

We present a first application of Greibach normal form. The presentation given here follows [33]. Recall that, given an alphabet V containing n letters, we denote by $D_V^\star$ the Dyck set over the alphabet $(V \cup \overline{V})$. Given a word $m \in (V \cup \overline{V})^\star$, we denote $\widetilde{m}$ the reversal of m. We denote $\mathfrak{P}((V \cup \overline{V})^\star)$ the family of subsets of $(V \cup \overline{V})^\star$. We now state Shamir's theorem [51]:

Theorem 4.1 (Shamir). *For any context-free language L over A, there exists an alphabet V, a letter $X \in V$ and a monoid homomorphism $\Phi : A^\star \to \mathfrak{P}((V \cup \overline{V})^\star)$ such that*

$$u \in L \iff X\Phi(u) \cap D_V^\star \neq \emptyset.$$

Proof. Let $G = (V, P)$ be a context-free grammar in Greibach normal form over A generating L. To each terminal letter $a \in A$, associate the finite set

$$\Phi(a) = \{\overline{X}\widetilde{\alpha} \in (V \cup \overline{V})^\star \mid X \longrightarrow a\alpha \in P\}.$$

This defines *Shamir's homomorphism* $\Phi : A^\star \longrightarrow \mathfrak{P}((V \cup \overline{V})^\star)$. A simple induction allows to prove that, for any terminal word $u \in A^\star$ and for any nonterminal word $m \in V^\star$

$$X \xrightarrow[\ell]{\star} um \quad \Longleftrightarrow \quad X\Phi(u) \cap [\widetilde{m}] \neq \emptyset$$

where $[\widetilde{m}]$ represents the class of $\widetilde{m}$ in the Dyck congruence. This shows, in particular, that $X \xrightarrow[\ell]{\star} u$ iff $X\Phi(u) \cap D_V^\star \neq \emptyset$ which is precisely the theorem. ■

For later use, we state another formulation of the theorem.

Given a context-free grammar $G = (V, P)$ over A in Greibach normal form generating L, we associate to each terminal letter $a \in A$ the set $\hat{\Phi}(a) = \Phi(a)a\overline{a}$. As far as Shamir's theorem is concerned, nearly nothing is changed: we use the total alphabet $T = V \cup A$ instead of V, and the same result holds with $D_T^\star$ instead of $D_V^\star$, that is

$$X \xrightarrow[\ell]{\star} u \Longleftrightarrow \exists w \in \hat{\Phi}(u) \ : \ Xw \in D_T^\star \ . \tag{4.1}$$

4.2 Chomsky-Schützenberger's theorem

We now show how to use Shamir's theorem 4.1 to prove directly the famous Chomsky-Schützenberger theorem [28, 10], that we recall here :

Theorem 4.2 (Chomsky-Schützenberger). *A language L over the alphabet A is context-free iff there exists an alphabet T, a rational set K over $(T \cup \overline{T})^\star$ and a morphism $\psi : (T \cup \overline{T})^\star \longrightarrow A^\star$, such that*

$$L = \psi(D_T^\star \cap K).$$

Proof. We follow again [33]. The "if" part follows from the classical closure properties of the family of context-free languages. Hence, we just sketch the proof of the "only if" part. Let $G = (V, P)$ be a grammar over A and set $T = V \cup A$.

Define a homomorphism ψ from $(T \cup \overline{T})^\star$ into $A^\star$ by

$$\begin{array}{l} \forall X \in V,\ \psi(X) = \psi(\overline{X}) = 1 \\ \forall a \in A,\ \ \psi(a) = a \text{ and } \psi(\overline{a}) = 1. \end{array}$$

Using morphism $\hat{\Phi}$ of the reformulation of Shamir's theorem, we note that $w \in \hat{\Phi}(u) \Longrightarrow \psi(w) = u$. Conversely, if $\psi(w) = u$ and $w \in \hat{\Phi}(A^\star)$, then $w \in \hat{\Phi}(u)$. Thus

$$w \in \hat{\Phi}(u) \iff \psi(w) = u \quad \text{for} \quad w \in \hat{\Phi}(A^\star).$$

Then, the right hand side of equation (4.1) is equivalent to

$$\exists w \in \hat{\Phi}(A^\star) : \psi(Xw) = u, \quad Xw \in D_T^\star \quad .$$

Thus, setting $K = X\hat{\Phi}(A^\star)$, which is rational, this can be written

$$X \xrightarrow[\ell]{\star} u \iff \exists w : \ \psi(Xw) = u, \quad Xw \in D_T^\star \cap K$$

and the Chomsky-Schützenberger theorem is proved. ■

4.3 The hardest context-free language

We now show how to use Shamir's theorem 4.1 to get the hardest context-free language. We begin by some new notions and results.

Given a language L over the alphabet A, we define the *nondeterministic version of* L, denoted $ND(L)$, in the following way: first add to the alphabet A three new letters [,] and +. A word h in $([(A^\star+)^\star A^\star])^\star$ can be naturally decomposed into $h = [h_1][h_2]\cdots[h_n]$, each word h_i being decomposed itself in $h_i = h_{i,1}+h_{i,2}+\cdots+h_{i,k_i}$, $h_{i,j} \in A^\star$. A *choice* in h is a word $h_{1,j_1}h_{2,j_2}\cdots h_{n,j_n}$ obtained by choosing in each $[h_i]$ a factor h_{i,j_i}. Denote by $\chi(h)$ the set of choices in h. Then, the nondeterministic version of L is defined by:

$$ND(L) = \{h \mid \chi(h) \cap L \neq \emptyset\}.$$

Given an alphabet A, we denote by H_A the nondeterministic version of the Dyck language $D_A^\star$. In the particular case of a two letter alphabet $A = \{a, b\}$, we skip the index, so that H_A is denoted H. By definition, H is the *hardest context-free language*.

The first important observation is given by:

Fact 4.1. *If L is a context-free language, so is its nondeterministic version ND(L).*

This lemma can be easily proved either by using pushdown automata or by showing that *ND(L)* is obtained by a rational substitution applied to the language L. The terminology nondeterministic version of L comes from the following

Proposition 4.1. *The language ND(L) is deterministic context-free iff L is regular; in this case, ND(L) is regular too.*

For a proof, we refer the reader to [3]. We now end this short preparation by the

Lemma 4.1. *Given an alphabet A, there exists a morphism λ such that $H_A = \lambda^{-1}(H)$.*

Proof. Let $H = H_B$ with $B = \{a, b\}$.

If the alphabet A contains only one letter c, just define the morphism λ by $\lambda([) = [$, $\lambda(]) =]$, $\lambda(+) = +$, $\lambda(c) = a$ and $\lambda(\overline{c}) = \overline{a}$.

If A contains $n \geq 3$ letters, λ will be the usual encoding : $\lambda(a_i) = ab^i a$ and $\lambda(\overline{a_i}) = \overline{a}\overline{b}^i\overline{a}$ for $1 \leq i \leq n$. For the three other letters, we define $\lambda([) = [$, $\lambda(]) =]$ and $\lambda(+) = +$. ∎

We now state the

Theorem 4.3 (Greibach). [23] *A language L over the alphabet A is context-free iff there exists a morphism φ such that $\$L = \varphi^{-1}(H)$, where $\$$ is a new letter.*

Proof. The "if" part follows directly from the closure properties of the family of context-free languages and from the fact that H is context-free. Hence, we turn to the "only if" part, for which we follow once more [33]. Given a context-free grammar $G = (V, P)$ in Greibach normal form, we associate to the morphism Φ used in Shamir's theorem a morphism φ defined by

$$\varphi(a) = [m_1 + m_2 + \cdots m_n] \iff \Phi(a) = \{m_1, m_2, \ldots, m_n\}$$

Here, $m_1, m_2, \ldots, m_n$ is some arbitrary but fixed enumeration of the words in $\Phi(a)$. Moreover, we define $\varphi(\$) = [X]$ if X is the variable in V generating L. Set now $\theta = \chi\varphi$; hence, $\theta(u)$ will be the set of choices of the word $\varphi(u)$.

It is easy to check that

$$w \in \theta(u) \iff w \in \Phi(u) \qquad \text{i.e.} \qquad \theta = \Phi.$$

(Just interpret the word $h = \varphi(u)$ as a polynomial representing the set $\Phi(u)$ and develop this polynomial.)

Consequently, Shamir's theorem can be rephrased as

$$X\Phi(u) \cap D_V^* \neq \emptyset \iff \theta(\$u) \cap D_V^* \neq \emptyset \iff \$u \in \varphi^{-1}(H_V).$$

Hence, we have $\$L = \varphi^{-1}(H_V)$. The announced result in theorem 4.3 then follows from lemma 4.1. ∎

Observation. The *membership problem* is the following: given a language L and a word u, does u belong to L? The language H is called the hardest context-free language because, by theorem 4.3, from a complexity point of view, H is the context-free language for which the membership problem is the most difficult. Any algorithm deciding if a given word belongs to H gives raise to a general algorithm for the membership problem for context-free languages; this general algorithm will have the same complexity than the one given for H.

4.4 Wechler's theorem

We end this section by showing another consequence of the Greibach normal form. Given a language L over the alphabet A and a letter $a \in A$, recall that the *left quotient* of L by a is the language $a^{-1}L = \{u \in A^\star \mid au \in L\}$. An *algebra* is a family of languages closed under union and product and containing the family Fin of finite languages. An algebra $\mathcal{F}$ is *finitely generated* if there exists a finite family $\mathcal{F}'$ such that any language in $\mathcal{F}$ is obtained from languages in $\mathcal{F}'$ under the algebra operations. It is *stable* if it is closed under left quotient. We may now state the

Theorem 4.4 (Wechler). [54] *A language L is context-free if and only if it belongs to a finitely generated stable algebra.*

Proof. Given a context-free language L, it is generated by a grammar in Greibach normal form. To each variable X is associated the (context-free) language L_X that it generates. Clearly, the left quotient of such a language by a terminal letter a can be described as a finite union of product of languages generated in the grammar. Hence, the algebra generated by all these languages L_X contains L and is stable.

Conversely, if L belongs to a finitely generated stable algebra, the finite set of generators give raise to a finite set of variables and the description of each left quotient as a finite union of product of languages of the generators gives a grammar in Greibach normal form generating L. ∎

5. Pushdown machines

In this section, we focus on the accepting device for context-free languages, namely pushdown automata with the important subclass induced by determinism, in both classical and less classical presentations. We prove here mainly two beautiful theorems: the first states that the stack language of a pushdown automaton is a rational language; the second says that the output language of a pushdown transducer is context-free when the input is precisely the language recognized by the associated pda.

5.1 Pushdown automata

The classical mechanism of recognition associated to context-free languages is the pushdown automaton. Most of the material presented in this paragraph is already in Ginsburg[17].

A *pushdown machine* over A (a *pdm* for short) is a triple $\mathcal{M} = (Q, Z, T)$ where Q is the set of *states*, Z is the *stack alphabet* and T is a finite subset of $(A \cup \{\varepsilon\}) \times Q \times Z \times Z^* \times Q$, called the set of *transition rules*. A is the *input alphabet*. An element (y, q, z, h, q') of T is a rule, and if $y = \varepsilon$, it is an ε-rule.

The first three components are viewed as pre-conditions in the behaviour of a pdm (and therefore the last two components are viewed as post-conditions), T is often seen as a function from $(A\cup\{\varepsilon\})\times Q\times Z$ into the subsets of $Z^*\times Q$, and we note $(h,q')\in T(y,q,z)$ as an equivalent for $(y,q,z,h,q')\in T$.

A pushdown machine is *realtime* if T is a finite subset of $A\times Q\times Z\times Z^*\times Q$, i.e. if there is no ε-rule. A realtime pdm is *simple* if there is only one state. In this case, the state giving no information, it is omitted, and T is a subset of $A\times Z\times Z^*$.

An *internal configuration* of a pdm $\mathcal{M}$ is a couple $(q,h)\in Q\times Z^*$, where q is the current state, and h is the string over Z^* composed of the symbols in the stack, the first letter of h being the bottom-most symbol of the stack. A *configuration* is a triple $(x,q,h)\in A^*\times Q\times Z^*$, where x is the input word to be read, and (q,h) is an internal configuration.

The *transition relation* is a relation over configurations defined in the following way: let $c=(yg,q,wz)$ and $c'=(g,q',wh)$ be two configurations, where y is in $(A\cup\{\varepsilon\})$, g is in A^*, q and q' are in Q, z is in Z, and w and h are in Z^*. There is a *transition* between c and c', and we note $c\longmapsto c'$, if $(y,q,z,h,q')\in T$. If $y=\varepsilon$, the transition is called an ε-transition, and if $y\in A$, the transition is said to involve the reading of a letter. A *valid computation* is an element of the reflexive and transitive closure of the transition relation, and we note $c\overset{*}{\longmapsto} c'$ a valid computation starting from c and leading to c'. A convenient notation is to introduce, for any word $x\in A^*$, the relation on internal configurations, denoted $\overset{x}{\Longmapsto}$, and defined by:

$$(q,w)\overset{x}{\Longmapsto}(q',w') \iff (x,q,w)\overset{*}{\longmapsto}(\varepsilon,q',w').$$

We clearly have: $\overset{x}{\Longmapsto}\circ\overset{y}{\Longmapsto} \;=\; \overset{xy}{\Longmapsto}$.

An internal configuration (q',w') is *accessible* from an internal configuration (q,w), or equivalently, (q,w) is *co-accessible* from (q',w') if there is some $x\in A^*$ such that $(q,w)\overset{x}{\Longmapsto}(q',w')$.

A rule $(y,q,z,h,q')\in T$ is an increasing rule (respectively a stationary, respectively a decreasing rule) if $|h|>1$ (respectively $|h|=1$, respectively $|h|<1$). The use of an increasing rule (respectively a stationary, respectively a decreasing rule) in a computation increases (respectively leaves unchanged, respectively decreases) the number of symbols in the stack. A pdm is in *quadratic form* if for all rules $(y,q,z,h,q')\in T$, we have: $|h|\leq 2$.

A pdm is used as a device for recognizing words by specifying starting configurations and accepting configurations. The convention is that there is only one starting internal configuration $i=(q,z)$, where the state q is the initial state, and the letter z is the initial stack symbol. For internal accepting configurations, many kinds make sense, but the set K of internal accepting configurations usually is of the form: $K=\cup_{q\in Q}\{q\}\times K_q$ with $K_q\in Rat(Z^*)$.

A *pushdown automaton* over A (a *pda* for short) is composed of a pushdown machine (Q,Z,T) over A, together with an *initial internal configura-*

tion i, and a set K of *internal accepting configurations.* It is so a 5-tuple $\mathcal{A}=(Q,Z,i,K,T)$, and (Q,Z,T) is called the *pdm associated* to $\mathcal{A}$.

For a pda, an internal configuration is *accessible* if it is accessible from the initial internal configuration, and it is *co-accessible* if it is co-accessible from an internal accepting configuration.

The sets of internal accepting configurations usually considered are:

1. the set $F \times Z^*$ where F is a subset of Q, called the set of *accepting states.*
2. the set $Q \times \{\varepsilon\}$.
3. the set $F \times \{\varepsilon\}$ where F is a subset of Q.
4. the set $Q \times Z^* Z'$ where Z' is a subset of Z.

We call each of these cases a *mode of acceptance.*

A word $x \in A^*$ is *recognized* by a pda $\mathcal{A} = (Q,Z,i,K,T)$ over A with a specified mode of acceptance if there is $k \in K$ such that $i \overset{x}{\models\!=} k$. Considering the modes of acceptance defined above, in the first case, the word is said to be recognized by *accepting states* F, in the second case the word is said to be recognized by *empty storage*, in the third case the word is said to be recognized by *empty storage and accepting states* F, and in the last case the word is said to be recognized by *topmost stack symbols* Z'. The *language accepted* by a pda with a given mode of acceptance is the set of all words recognized by this pda with this mode. For any pda $\mathcal{A} = (Q,Z,i,K,T)$, we note $L(\mathcal{A})$ the language recognized by $\mathcal{A}$, and for any set of internal accepting configurations K', we note $L(\mathcal{A},K')$ the language recognized by the pda $\mathcal{A}' = (Q,Z,i,K',T)$.

Note that, with regards to the words recognized, the names of the states and of the stack symbols are of no importance. Up to a renaming, we can always choose $Q = \{q_1,q_2,\ldots,q_p\}$, and similarly, $Z = \{z_1,z_2,\ldots,z_n\}$. Up to a renaming too, we can always set the initial internal configuration equal to (q_1,z_1).

Example 5.1. Let $\mathcal{A} = (Q,Z,(q_0,t),K,T)$ be the pda over $A = \{a,b\}$, where $Q = \{q_0,q_1,q_2,q_3\}$, $Z = \{z,t\}$ of rules:

$$\begin{array}{ll}
(a,q_0,t,zt,q_1), & (a,q_0,t,zzt,q_2),\\
(a,q_1,t,zt,q_1), & \\
(a,q_2,t,zzt,q_2), & (a,q_2,t,zt,q_1),\\
(\varepsilon,q_1,t,\varepsilon,q_3), & (\varepsilon,q_2,t,\varepsilon,q_3),\\
(b,q_3,z,\varepsilon,q_3). &
\end{array}$$

In state q_1, each letter a read increases by one the number of symbols z under the top symbol t in the stack. In state q_2, each letter a read increases by two the number of symbols z under the top symbol t in the stack, or increases it by one and changes the state to q_1. The two ε-rules remove the top stack symbol t, changing the state to q_3, in which the only thing possible to do is removing one z in the stack for each b read.

Then we have, for example:

$$\begin{aligned}
&L(\mathcal{A}, Q \times \{\varepsilon\}) = \{a^n b^p \mid 0 < n \le p \le 2n\},\\
&L(\mathcal{A}, \{q_3\} \times Z^*) = \{a^n b^p \mid 0 < n \text{ and } 0 \le p \le 2n\},\\
&L(\mathcal{A}, \{q_2\} \times \{\varepsilon\}) = \emptyset,\\
&L(\mathcal{A}, Q \times Z^* z) = \{a^n b^p \mid 0 < n \text{ and } 0 \le p < 2n\}.
\end{aligned}$$

As seen on this example, for a given pda, changing the mode of acceptance changes the languages recognized. Nevertheless, the family of languages that are recognized by pda's, using any of these modes remains the same. This can be proved easily using a useful, though technical, transformation of a pda adding it the bottom symbol testing ability. A pda *admits bottom testing* if there is a partition of the stack alphabet Z in $B \cup B'$ such that for any accessible configuration (q, w), the word w is in BB'^*. In other words, in such an automaton, a symbol at the bottom of the stack always belongs to B and, conversely, a symbol in the stack which belongs to B is the bottom symbol. So, if the topmost symbol of the stack happens to be a symbol in B, it is the only symbol in the stack. Since the only symbol in the stack that may be tested is the topmost symbol, it is then possible to know if it is the bottom symbol of the stack. Under these conditions, a valid computation leads to a configuration with an empty store if and only if the last transition uses a rule of the form: $(y, q, z, \varepsilon, q') \in T$ where z is in B. One construction to transform a pda $\mathcal{A}$ into a pda $\mathcal{A}'$ admitting bottom testing is the following. Let $\mathcal{A} = (Q, Z, i, K, T)$, let $Z' = \{z' \mid z \in Z\}$ be a copy of Z, and define T' by:

$$(y, q, z, \varepsilon, q') \in T \Leftrightarrow (y, q, z, \varepsilon, q') \in T' \wedge (y, q, z', \varepsilon, q') \in T'$$

and

$$(y, q, z, z_1 z_2 \ldots z_r, q') \in T \Leftrightarrow \begin{cases} (y, q, z, z_1 z_2' \ldots z_r', q') \in T' \wedge \\ (y, q, z', z_1' z_2' \ldots z_r', q') \in T'. \end{cases}$$

Finally, denoting by $\pi : (Z \cup Z')^* \to Z^*$ the homomorphism that erases the primes, set

$$K' = \{(q, h') \mid (q, \pi(h')) \in K\}$$

and

$$\mathcal{A}' = (Q, Z \cup Z', i, K', T').$$

Proposition 5.1. *The pda $\mathcal{A}'$ admits bottom testing and recognizes the same language as $\mathcal{A}$, for any mode of acceptance.*

Hence there is a common family of languages recognized by pda's using any mode of acceptance which is the family of context-free languages:

Theorem 5.1. *The family of languages recognized by pda's by empty storage and accepting states is exactly the family of context-free languages.*

Proof. Let $\mathcal{A} = (Q, Z, i, K, T)$ be a pda. We denote $[p, w, q]$, for $w \in Z^+$, the language

$$[p, w, q] = \{x \in A^* \mid (p, w) \overset{x}{\models\!\!=} (q, \varepsilon)\},$$

and set

$$[p, \varepsilon, q] = \begin{cases} \emptyset & \text{if } p \neq q \\ \varepsilon & \text{if } p = q \end{cases}$$

We then have, for $w, w' \in Z^*$:

$$[p, ww', q] = \bigcup_{r \in Q} [p, w', r][r, w, q].$$

We can derive from T that the languages $[q, z, q']$, for $z \in Z$, satisfy the set of equations:

$$[p, z, q] = \bigcup_{(y,p,z,h,q') \in T} y[q', h, q]. \tag{5.1}$$

Hence the languages $[q, z, q']$ are all context-free, and so is the language:

$$\bigcup_{q \in F,\ i=(q_1,z_1)} [q_1, z_1, q]$$

which is exactly the language recognized by $\mathcal{A} = (Q, Z, i, K, T)$ with $K = \{\varepsilon\} \times F$ (i.e. by empty storage and accepting states F).

Conversely, if $G = (V, P)$ is a context-free grammar over A such that $P \subset V \times (A \cup \{\varepsilon\})V^*$, one can construct from P a pdm $\mathcal{M} = (V, T)$ over A without states, where $T \subset (A \cup \{\varepsilon\}) \times V \times V^*$ is defined by: $(y, X, m) \in T \iff X \to y\tilde{m}$. The language $L_G(X)$ is then recognized by the pda associated to $\mathcal{M}$ with initial stack symbol X by empty storage. ∎

Remark 5.1. If the system of equations (5.1) is replaced by the associated context-free grammar, there is a one to one correspondence between valid computations of the pda and leftmost derivations in the grammar. Hence the number of different valid computations leading to recognize a word x gives the number of different leftmost derivations for x.

For pushdown automata, the mode of acceptance is generally chosen to give the simplest proofs for one's purpose. Other modes of acceptance than the ones quoted above have been investigated. For instance, a result of Sakarovitch [46] shows that if $K = \cup_{q \in Q}\{q\} \times L_q$ with L_q context-free, then the language recognized remains context-free.

The characterization of context-free languages in terms of languages recognized by pda's allows much simpler proofs of certain properties of context-free languages.

Example 5.2. In order to show that the family of context-free languages is closed under intersection with rational languages, consider a context-free language L given by a pda $\mathcal{A}$, and a rational language K given by a finite automaton $\mathcal{B}$. Then a pda recognizing $L \cap K$ can effectively be constructed, using the Cartesian product of the states of $\mathcal{A}$ and of the states of $\mathcal{B}$.

A pushdown automaton is *realtime* (resp. *simple*) if the associated pdm is realtime (resp. simple).

The fact that any proper context-free language can be generated by a context-free grammar in Greibach normal form implies that realtime pda's, (and even simple pda's), recognize exactly proper context-free languages.

The realtime feature is the key to formulate Shamir's and Greibach's theorems (theorems 4.1 and 4.3), and that we rephrase here in an automata-theoretic framework.

In any pdm $\mathcal{M} = (Q, Z, T)$, the set T can be written as a function $\hat{T}$ from $(A \cup \{\varepsilon\})$ into the subsets of $Q \times Z \times Z^* \times Q$. In the case of a realtime pdm, it is a function from A into the subsets of $Q \times Z \times Z^* \times Q$. Let $\overline{Z}$ be a copy of Z and $\overline{Q}$ a copy of Q as well, we can conveniently denote the element (q, z, h, q') in $Q \times Z \times Z^* \times Q$ by the word $\overline{q}.\overline{z}.h.q'$ over the Dyck alphabet $Q \cup Z \cup \overline{Z} \cup \overline{Q}$. Recall that we denote $D^*_{Q \cup Z}$ the Dyck set over this alphabet. The Shamir function Φ from A into the subsets of $(Q \cup Z \cup \overline{Z} \cup \overline{Q})^*$ is defined by

$$\Phi(a) = \{\overline{qz}hq' \mid (q, z, h, q') \in \hat{T}(a)\}\,.$$

Then extend it in the natural way to a morphism from A^* into the subsets of $(Q \cup Z \cup \overline{Z} \cup \overline{Q})^*$. Thus, Shamir's theorem states that $\Phi(x) \cap z_1 q_1 D^*_{Q \cup Z} F \neq \emptyset$ iff x is recognized by the realtime pda $\mathcal{A} = (Q, Z, i, F \times \{\varepsilon\}, T)$ by empty storage and accepting states F.

The Shamir function Φ gives raise to a function from A into $(\{[,], +\} \cup Q \cup Z \cup \overline{Z} \cup \overline{Q})^*$, extended to an homomorphism φ, that we call the *Greibach homomorphism*, by setting:

$$\varphi(x) = [m_1 + m_2 + \ldots + m_k] \Longleftrightarrow \Phi(x) = \{m_1, m_2, \ldots, m_k\}.$$

Let $H_{Q \cup Z}$ be the Hardest context-free language over $Q \cup Z$. It follows that $[z_1 q_1]\varphi(x)F \in H_m$ iff x is recognized by the realtime pda $\mathcal{A} = (Q, Z, i, K, T)$ by empty storage and accepting states F. This is theorem 4.3.

The presence of the Dyck set in Shamir's theorem is due to the fact that this language fully describes the behaviour of the stack in a pdm: a letter that is unmarked is pushed on the top of the stack, while a marked letter erases the corresponding letter provided it is the topmost symbol in the stack. Recognition by empty storage means that the stack must be empty at the end of the computation, and D^* is precisely the class of the empty word ε for the Dyck congruence.

5.2 Deterministic pda

We now focus on determinism.

A pdm $\mathcal{M} = (Q, Z, T)$ over A is *deterministic* if the set T of transitions satisfies the following conditions for all $(y, q, z) \in (A \cup \{\varepsilon\}) \times Q \times Z$:

$$\begin{gathered} \mathrm{Card}(T(y, q, z)) \leq 1 \\ T(\varepsilon, q, z) \neq \emptyset \implies T(a, q, z) = \emptyset,\ (a \in A). \end{gathered}$$

A *deterministic pda* (dpda for short) is a pda with a deterministic associated pdm. The transformation of a pda into a pda admitting bottom testing described above, when applied to a deterministic pda, gives raise to a deterministic pda. Hence, it is possible to prove that the family of languages recognized by dpda's by empty storage is the same as the family of languages recognized by dpda's by empty storage and accepting states, and that this family is included in the family of languages recognized by dpda's by accepting states. On the other hand, it is easy to verify that a language recognized by empty storage by a dpda is prefix, *i.e.* no proper prefix of a word of this language belongs to this language. So, we are left with two families of languages: the family of languages recognized by accepting states, called the family of *deterministic languages*, and the family of languages recognized by empty storage and accepting states, called the family of *deterministic-prefix languages*. It is easy to check the following

Fact 5.1. *The family of deterministic-prefix languages is exactly the family of deterministic languages that are prefix languages.*

The two families are distinct. As an example, the language $L_1 = \{a^n b^p \mid p > n > 0\}$ is deterministic but not prefix. To avoid these problems, a usual trick is to consider languages with an end marker: indeed, $L\#$ is a prefix language which is deterministic if and only if L is deterministic.

One awkward feature about dpda's is that, due to possible ε-transitions that may occur after the input of the last letter of the word, there may be several valid computations for a fixed input word (being the beginning of one each other). This inconvenient can be avoided by a rather technical construction (see e. g. [2]) that transforms a dpda into an other dpda such that an accepting state is reached only if the computation is maximal.

Proposition 5.2. *For any dpda, it is possible to construct a dpda recognizing the same language such that an accepting state cannot be on the left side of an ε-rule.*

Consequently, in such a dpda, for any recognized word, there is only one successful computation. This proves the following

Proposition 5.3. *Deterministic languages are unambiguous.*

To see that the inclusion is strict, consider the language $L_2 = \{a^n b^n \mid n > 0\} \cup \{a^n b^{2n} \mid n > 0\}$. It is unambiguous, and it is not a deterministic language. Indeed, looking carefully at the valid computation used to recognize a word $a^n b^n$, it is not too difficult to prove that it is possible to find a word $a^{n+k} b^{n+k}$ for some $k > 0$ such that the internal configuration reached is the same than for the former word. Now, the valid computation for $a^n b^n$ should be the beginning of the valid computation for the word $a^n b^{2n}$. Hence the automaton must recognize the word $a^{n+k} b^{2n+k}$, which is not in L_2.

By the way, the technical construction invoked in Proposition 5.2 is also the key to prove the following

Theorem 5.2. *The family of deterministic languages is closed under complementation.*

This property is not true for the family of context-free languages: the language $\{a^n b^p c^q \mid n \neq p \text{ or } p \neq q\}$ is a context-free language, and its complement, intersected by the rational language $a^*b^*c^*$, is the language $\{a^n b^p c^q \mid n = p \text{ and } p = q\}$ which is not context-free.

Proposition 5.4. *For any dpda, it is possible to construct a dpda recognizing the same language such that any ε-rule is decreasing.*

This proposition is quoted as an exercise in [28] and [30]. However, no proof has appeared in the standard textbooks. A proof is given below.

The proof is in two steps of independent interest: first, we get rid of nondecreasing ε-rules for dpdas recognizing by topmost stack symbols *and* accepting states. In a second step, we show that such a dpda recognizes a deterministic language. This is achieved by constructing an equivalent ordinary dpda, but without introducing any nondecreasing ε-rule.

Proposition 5.5. *Given a dpda $\mathcal{A}$ recognizing by topmost stack symbols and accepting states, it is possible to construct a dpda $\mathcal{A}'$ recognizing the same language with the same mode of acceptance, and such that any ε-rule is decreasing.*

Proof. Let $\mathcal{A} = (Q, Z, i, K, T)$ be a dpda over A. Observe first that we may always saturate the set K of accepting configurations by adding all configurations $(q, h) \in Q \times Z^+$ such that $(q, z) \overset{\varepsilon}{\models\!=} k$ for $k \in K$.

Claim. The number of consecutive nondecreasing ε-transitions in a computation may be assumed to be uniformly bounded.

The proof of the claim is simple, and appears for instance in [17].

Claim. One may assume that there are never two consecutive nondecreasing ε-transitions in a computation.

The idea is to glue together, in a single rule, any maximal (bounded in view of the first claim!) sequence of consecutive nondecreasing ε-transitions appearing in a computation. If such a sequence contains an accepting configuration then, due to the saturation of K, its initial configuration is accepting, too.

Claim. One may assume that, in any computation, there is never a nondecreasing ε-transition followed by a decreasing ε-transition.

Again, the idea is to glue together a nondecreasing ε-rule followed by a decreasing ε-rule into one ε-rule. This decreases the total number of ε-rules. Therefore, the process stops after a finite number of steps. Accepting configurations are handled in the same way than above.

From now on, we may assume, in view of these claims, that any nondecreasing ε-transition either ends the computation or is not followed by an ε-transition.

We now finish the proof. Let $\mathcal{A}' = (Q, Z, i, K, T')$ be the automaton where T' is constructed as follows. T' contains all decreasing ε-rules of T. Next,

- If $T(\varepsilon, q, z) = \emptyset$, then $T'(a, q, z) = T(a, q, z)$ for $a \in A$.
- If $T(\varepsilon, q, z) = (r, m)$, with $m \neq \varepsilon$, then $T'(a, q, z) = (p, h)$, where $(a, r, m) \longmapsto (\varepsilon, p, h)$.

It is immediate to check that $\mathcal{A}'$ is equivalent to $\mathcal{A}$. By construction, it has only decreasing ε-rules. ∎

We now turn to the second step.

Proposition 5.6. *Given a dpda $\mathcal{A}$ recognizing by topmost stack symbols and accepting states, and having only decreasing ε-rules, it is possible to construct a dpda $\mathcal{B}$ recognizing the same language by accepting states, and such that any ε-rule is decreasing.*

Proof. Let $\mathcal{A} = (Q, Z, i, K, T)$ be a dpda over A. We construct $\mathcal{B} = (Q', Z, i, K', T')$ as follows: $Q' = Q \cup P$, where $P = \{q_z \mid (q, z) \in K\}$. Next, $K' = P$. The set of rules T' first contains $T'(\varepsilon, q, z) = (q_z, \varepsilon)$ for $(q, z) \in K$. Furthermore,

- If $T(a, q, z) \neq \emptyset$ for some letter $a \in A$, then $T(a, q, z) = (q', m)$ for some $q' \in Q$ and $m \in Z^*$. In this case,
$$T'(a, q_z, z') = (q', z'm) \text{ for all } z' \in Z\,.$$
- If $T(\varepsilon, q, z) \neq \emptyset$, then, since the rule is decreasing, $T(\varepsilon, q, z) = (q', \varepsilon)$ for some $q' \in Q$. In this case,
$$T'(y, q_z, z') = T(y, q', z') \text{ for all } y \in A \cup \{\varepsilon\} \text{ and } z' \in Z\,.$$

By construction, the dpda $\mathcal{B}$ has only decreasing ε-rules. Clearly, $\mathcal{B}$ is equivalent to $\mathcal{A}$. ∎

Proof of proposition 5.4. A successive application of propositions 5.5 and 5.6 proves the statement. ∎

Remark 5.2. The proposition 5.6, but without reference to ε-rules, is proved in a simpler way in [2]. However, his construction does not apply to the proof of proposition 5.4.

Proposition 5.4 shows that, for deterministic automata, nondecreasing ε-rules are not necessary. On the contrary, decreasing ε-rules cannot be avoided.

Example 5.3. The language

$$L_3 = \{a^n b^p c a^n \mid p, n > 0\} \cup \{a^n b^p d b^p \mid p, n > 0\}$$

is deterministic, and recognized by the dpda with rules:

$$\begin{array}{lll}
(a, q_1, z_1, z_1z_2, q_1), & (a, q_1, z_2, z_2z_2, q_1), & (b, q_1, z_2, z_3, q_2), \\
(b, q_2, z_3, z_3z_3, q_2), & (c, q_2, z_3, \varepsilon, q_3), & (d, q_2, z_3, z_3, q_5), \\
(\varepsilon, q_3, z_3, \varepsilon, q_3), & (a, q_3, z_2, \varepsilon, q_3), & (a, q_3, z_1, \varepsilon, q_4) \\
(b, q_5, z_3, \varepsilon, q_5), & (\varepsilon, q_5, z_2, z_2, q_6). &
\end{array}$$

by accepting states, with accepting states q_4 and q_6.

However, L_3 cannot be recognized by any realtime deterministic pda. Indeed, a word starts with $a^n b$, and it is necessary, while reading the factor b^p, to push on the stack an unbounded number of symbols that will be matched when the word ends with db^p, and all these symbols have to be erased when the word ends with ca^n.

This example shows that, contrarily to the general case, the realtime condition induces an effective restriction on the family of recognized languages.

Let $\mathcal{R}$ be the family of languages recognized by deterministic realtime automata by empty storage. There is a Shamir theorem for languages in $\mathcal{R}$, that we state now.

Let Φ be the Shamir function from A^* into the subsets of $(Q \cup Z \cup \overline{Z} \cup \overline{Q})^*$. Since the automaton is deterministic, for all $(q, z) \in Q \times Z$, there is at most one element beginning by $\overline{q}.\overline{z}$ in each image $\Phi(a)$ of $a \in A$. Such an homomorphism is called a "controlled" homomorphism. Shamir's theorem can be rephrased as follows. A language L is in $\mathcal{R}$ if and only if there exists a controlled homomorphism Φ from A^* into the subsets of $(Q \cup Z \cup \overline{Z} \cup \overline{Q})^*$ such that $\Phi(x) \cap z_1 q_1 {D_m}^* F \neq \emptyset \iff x \in L$.

We define a rational set R by: $w \in R$ iff $w = [w_1 + w_2 + \cdots + w_r]$, $w_k \in \overline{QZ}Z^*Q$ and for all $(q, z) \in Q \times Z$, there is at most one w_k beginning with $\overline{qz}$.

Considering the Greibach homomorphism φ from A^* into the monoid $(\{[,], +\} \cup Q \cup Z \cup \overline{Z} \cup \overline{Q})^*$, then $\varphi(a) \in R$, for all a in A. It follows that $[z_1 q_1]\varphi(x) \in H_{Q\cup Z} \cap R^*$ if and only if x is recognized by the realtime dpda $\mathcal{A} = (Q, Z, i, K, T)$ over A by empty storage. (Recall that $H_{Q\cup Z}$ is the Hardest context-free language over $Q \cup Z$.) By the way, we can remark that $H_{Q\cup Z} \cap R^*$ is itself a language in $\mathcal{R}$.

The additional condition of being a simple (realtime) dpda induces also an effective restriction:

Fact 5.2. *The language $L_4 = \{a^n b a^n \mid n > 0\}$ is realtime deterministic, but not simple.*

Proof. L_4 is recognized by empty storage and accepting states by the realtime dpda $\mathcal{A} = (Q, Z, T, i, K, T)$ where $Q = \{q_1, q_2, q_3\}$, $Z = \{B, z\}$, $i = (q_1, B)$, $K = \{(q_3, \varepsilon)\}$ and T is the set of rules:

$$(a, q_1, B, Bz, q_1),\ (a, q_1, z, zz, q_1),\ (b, q_1, z, \varepsilon, q_2),$$
$$(a, q_2, z, \varepsilon, q_2), \quad (a, q_2, B, \varepsilon, q_3).$$

Hence, L_4 is a realtime deterministic language. However, it cannot be recognized by a deterministic simple automata, since it is necessary to know whether an input letter a belongs to the first or to the second factor a^n. ∎

Given two families of languages $\mathcal{C}$ and $\mathcal{D}$, the *equivalence problem* for $\mathcal{C}$ and $\mathcal{D}$, denoted $Eq(\mathcal{C}, \mathcal{D})$, is the following decision problem:
Instance: A language L in $\mathcal{C}$, and a language M in $\mathcal{D}$.
Question: Is L equal to M?

The *equivalence problem* for $\mathcal{C}$, denoted $Eq(\mathcal{C})$, is the problem $Eq(\mathcal{C}, \mathcal{C})$. It is well known that $Eq(Alg)$ is undecidable for the family Alg of context-free languages, and up to now it is unknown whether $Eq(Det)$ is decidable or not, where Det denotes the family of deterministic languages. So there has been a huge amount of works solving $Eq(\mathcal{C}, \mathcal{D})$ for various subfamilies of Det. We only quote here a few among the results published in the literature, in the positive case. The equivalence problem is decidable for parenthesis languages (see paragraph 6.6 below)[39], for simple languages (see paragraph 6.7 below)[36], for finite-turn languages (see paragraph 6.4 below)[53], realtime languages[42] ,(pre-)NTS languages[49]. A result of Sénizergues[50] shows that if $\mathcal{C}$ is an effective cylinder (i. e. a family of languages effectively closed under inverse homomorphism and intersection with rational sets) containing the family Rat of rational sets for which the equivalence problem is decidable, then so is $Eq(\mathcal{C}, Det)$.

In order to recognize the whole family of deterministic languages by realtime automata, we have to modify the standard model of pdm. We already noticed that, in a dpda, only decreasing ε-rules are necessary. They are necessary because, as seen for the language L_3 defined above, it happens that some unbounded amount of information pushed on the stack has to be erased at the same time. So, if we want to have a realtime device, this leads to use some mechanism that erases an unbounded number of topmost stack symbols in one step. Several such mechanisms have been introduced and studied in the literature (see e.g. Cole[11], Courcelle[12], Greibach[26], Nivat[41], Schützenberger[48]). We present now one such accepting device.

A *jump pdm* over A is a 4-tuple $\mathcal{A} = (Q, Z, J, T)$, where Q and Z have the same meaning as in a pdm, and J is a new alphabet in bijection with Z, the elements of which are called jump stack symbols, or simply *jump symbols*, and T, the set of transitions, is a finite subset of $A \times Q \times Z \times (Z^* \cup J) \times Q$. We denote λ the bijection between J and Z. Observe that, by definition, a jump pdm is a realtime device. A jump pdm is deterministic if for all $(a, q, z) \in A \times Q \times Z$, there is at most one (h, q) such that $(a, q, z, h, q) \in T$.

Configurations of a jump pdm are just the same as configurations of a pdm, but the transition relation is modified: let $c = (ag, q, wz)$ and $c' = (g, q', w')$, where a is in A, g is in A^*, q and q' are in Q, z is in Z, and w and w' are in Z^*. There is a transition between c and c', and we write $c \longmapsto c'$, if either $(a, q, z, h, q') \in T$ with $h \in Z^*$ and $w' = wh$, just as for pda's, or $(a, q, z, j, q') \in T$ with $j \in J$ if $w = w'zw_2$ and $z = \lambda(j)$ has no occurrence in w_2; in such a transition, an unbounded number of symbols (namely $|zw_2|$) is erased.

A *valid computation* is an element of the reflexive and transitive closure of the transition relation, and we note $c \overset{*}{\longmapsto} c'$ a valid computation starting from c and leading to c'.

A *jump pda* is to a jump pdm what a pda is to a pdm: it is a 6-tuple $\mathcal{A} = (Q, Z, J, i, K, T)$, where (Q, Z, J, T) is a jump pdm, and i and K have the same significance than in a pda. Observe that jump pda's generalize pda's: a pda is a jump pda with no jump rules. A jump pda is a *deterministic jump pda* (jump dpda for short) if the associated jump pdm is deterministic.

Since, in a jump pda, it is possible to erase an unbounded number of stack symbols in one move, the standard accepting mode is by empty storage. This is the mode considered when we do not specify an other one.

As an example, consider again the deterministic language

$$L_3 = \{a^n b^p c a^n \mid p, n > 0\} \cup \{a^n b^p d b^p \mid p, n > 0\}$$

over $\{a, b, c, d\}$. It is recognized by empty storage and accepting states by the jump pda $\mathcal{A} = (Q, Z, J, i, K, T)$, where $Q = \{q_1, \ldots, q_5\}$, $K = \{(\varepsilon, q_4)\}$, $Z = \{z, A, B\}$, $J = \{j_z, j_A, j_B\}$ and T is the set:

$$\begin{array}{l} (a, q_1, z, zA, q_1),\ (a, q_1, A, AA, q_1),\ (b, q_1, A, AB, q_2),\ (b, q_2, B, BB, q_2),\\ (d, q_2, B, \varepsilon, q_3),\quad (b, q_3, B, \varepsilon, q_3),\quad (b, q_3, A, j_z, q_4),\\ (c, q_2, B, j_A, q_5),\ (a, q_5, A, \varepsilon, q_5),\quad (a, q_5, z, \varepsilon, q_4). \end{array}$$

Indeed a word of L_3 begins with $a^n b^p$. The first four rules just push on the stack $A^n B^p$ over the bottom symbol z. Now, if the word ends with db^p, the three next rules are used to recognize the word: the first two to pop all symbols B while reading db^{p-1}, and the third (with jump symbol j_z) to erase the remaining symbols of the stack, i.e. zA^n. Last, if the word ends with ca^n, the last three rules are used to recognize the word: the first one (with jump symbol j_A) to erase all the top factor AB^p in the stack, the second to pop all symbols A while reading a^{n-1}, and the third to erase the remaining symbol z at the reading of the last a.

It is easy to construct from a (deterministic) jump pdm, a (deterministic) pdm (which will not be in general realtime) that acts in the same way: first, a rule (a, q, z, j, p) is replaced by the rule (a, q, z, z, p_j), where p_j is a new state. This replacement does not change determinism. Then, the following set of rules is added:

$$\{(\varepsilon, p_j, z, \varepsilon, p_j) \mid \lambda(j) \neq z\} \cup \{(\varepsilon, p_j, z, \varepsilon, p \mid \lambda(j) = z\}\,.$$

Remark that these new rules do not enter in conflict with the older ones, since the states involved are new states, nor with one another. So, determinism is preserved by this construction.

Consequently, in the nondeterministic case, we have the following

Proposition 5.7. *The family of languages recognized by jump pda's is exactly the family of context-free languages.*

A similar statement holds for deterministic languages.

Proposition 5.8. *The family of languages recognized by deterministic jump pda's is exactly the family of deterministic languages.*

In view of the preceding construction, and of the remark concerning the deterministic case, it only remains to prove that a deterministic language can be recognized by a jump dpda. The proof is very technical and lengthy, so we refer the interested reader either to Greibach[25], or to Cole[11].

An other model considered allows to erase rational segments of the stack word. This is clearly a generalization of jump pdm, since in a jump pdm, the erased factors have the form zh with $h \in (Z-\{z\})^*$. Observe that this rational set is recognized by the finite automaton obtained from the rules added in the construction above, (rules of the form: $(\varepsilon, p_j, z, \varepsilon, p_j)$ or $(\varepsilon, p_j, z, \varepsilon, p)$) when skipping first and fourth components (those equal to ε). It is an easy exercise to change the sets of rules added so that the erased factors belong to any rational set. If the rational sets are chosen to be prefix, as it is the case for jump pdm, determinism is still preserved. Hence, this model is equivalent to jump pdm.

Just as the behaviour of the stack in a pdm is described by the Dyck set, the behaviour of the stack in a jump pdm is described by a new set E_Z, which is a generalization of the Dyck set, defined as follows. E_Z is the class of the empty word for the congruence generated by

$$\{z\overline{z} \cong \varepsilon \mid z \in Z\} \bigcup \{zj \cong j \mid z \in Z,\ \lambda(j) \neq z\} \bigcup \{zj \cong \varepsilon \mid z \in Z,\ \lambda(j) = z\}\,.$$

We name E_m this set if $m = \mathrm{Card}(Z) = \mathrm{Card}(J)$.

It is a result of Greibach[26] that each language E_m cannot be recognized by a deterministic jump pda with $m-1$ jump symbols. Hence, the number of jump symbols induces a hierarchy.

Again, it is possible to state a Shamir-Greibach like theorem for deterministic languages, using jump dpda: let $\mathcal{A} = (Q, Z, J, i, K, T)$ be a deterministic jump pda over A. This time, the Shamir function Φ is a function from A^* into the subsets of $(Q \cup Z \cup T \cup \overline{Z} \cup \overline{Q})^*$, and the Greibach homomorphism φ is a function from A^* into $(\{[,],+\} \cup Q \cup Z \cup T \cup \overline{Z} \cup \overline{Q})^*$. We define a rational set R' by: $w \in R'$ iff $w = [w_1 + w_2 + \cdots + w_r]$, $w_k \in \overline{QZ}(Z^* \cup T)Q$ and for all $(q, z) \in Q \times Z$, there is at most one w_k beginning with $\overline{qz}$.

We have that for all a in A, $\varphi(a)$ is in R'. If $ND(E_{Q\cup Z})$ is the nondeterministic version of $E_{Q\cup Z}$ (see section 3.3), it follows that $[z_1 q_1]\varphi(x) \in ND(E_{Q\cup Z}) \cap R'^*$ if and only if x is recognized by the deterministic jump pda $\mathcal{A} = (Q, Z, J, i, K, T)$.

Again, we can remark that $ND(E_{Q\cup Z}) \cap R'^*$ is itself recognized by a deterministic jump pda.

5.3 Pushdown store languages

In this paragraph, we show that the language composed of the strings that may occur in the pushdown store is rational.

Let $\mathcal{A} = (Q, Z, i, K, T)$ be a pda over A. We call *pushdown store language* of $\mathcal{A}$ the language $P(\mathcal{A})$ over Z of all words u such that there exists some state q for which the internal configuration (q, u) is both accessible and co-accessible. Formally, $P(\mathcal{A})$ is defined by:

$$P(\mathcal{A}) = \{u \in Z^* \mid \exists x, y \in A^*,\ \exists q \in Q,\ \exists k \in K\ :\ i \overset{x}{\models\!=} (q, u) \overset{y}{\models\!=} k\}.$$

Theorem 5.3. *Given a pda and some mode of acceptance, the pushdown store language of this pda is rational.*

For any state $q \in Q$, we define the two sets:

$$\begin{aligned} Acc(q) &= \{u \in Z^* \mid\ \exists x \in A^*\ :\ i \overset{x}{\models\!=} (q, u)\}, \\ Co\text{-}Acc(q) &= \{u \in Z^* \mid\ \exists y \in A^*, \exists k \in K\ :\ (q, u) \overset{y}{\models\!=} k\}\,. \end{aligned}$$

Clearly:

$$P(\mathcal{A}) = \bigcup_{q \in Q} (Acc(q) \cap Co\text{-}Acc(q)) \tag{5.2}$$

We now show that the languages $Acc(q)$ and $Co\text{-}Acc(q)$ are rational.

Lemma 5.1. *The set $Acc(q)$ is rational.*

Proof. We first consider the particular case of a pda $\mathcal{A} = (Q, Z, i, K, T)$ in quadratic form, i.e. such that for any rule $(a, q, z, h, q') \in T$, $|h| \leq 2$.

Let $u = t_1 \cdots t_{r+1}$, where $t_i \in Z$. A valid computation $(x, q_0, y_1) \overset{*}{\longmapsto} (\varepsilon, q, u)$ can be decomposed into several steps such that, at the last move of each of these steps, one letter of u is definitively set in the stack. Formally, the whole computation is decomposed into:

$$\begin{aligned} &(x, q_0, y_1) \overset{*}{\longmapsto} (x_1, q_1, z_1') \longmapsto (x_1', q_1', t_1 y_2) \overset{*}{\longmapsto} (x_2, q_2, t_1 z_2') \longmapsto (x_2', q_2', t_1 t_2 y_3) \\ &\quad \overset{*}{\longmapsto} \cdots \overset{*}{\longmapsto} (x_r, q_r, t_1 t_2 \cdots t_{r-1} z_r') \longmapsto (x_r', q_r', t_1 t_2 \cdots t_r y_{r+1}) \overset{*}{\longmapsto} (\varepsilon, q, u), \end{aligned}$$

where $y_1, \ldots, y_{r+1}$ and $t_1, \ldots, t_{r+1}$ are in Z. Define now the context-free grammar G_q with terminal alphabet Z, nonterminal alphabet $Q \times Z$, and rules:

$$\begin{array}{lll} (p, z) \longrightarrow (p', z') & \text{if } \exists x \in A^*\ :\ (p, z) \overset{x}{\models\!=} (p', z') \\ (p, z) \longrightarrow t(p', z') & \text{if } \exists a \in A \cup \{\varepsilon\}\ :\ (a, p, z, tz', p') \in T \\ (p, z) \longrightarrow \varepsilon & \text{if } \exists x \in A^*\ :\ (p, z) \overset{x}{\models\!=} (q, \varepsilon) \\ (q, z) \longrightarrow z & \end{array}$$

A straightforward proof by induction on the length of a derivation shows that if there is a derivation $(p, z) \overset{*}{\longrightarrow} u$ in G_q, then there is a valid computation $(p, z) \overset{x}{\models\!=} (q, u)$ in $\mathcal{A}$.

Conversely, if there is a valid computation $(x, q_0, y_1) \overset{*}{\vdash\!\!-} (\varepsilon, q, u)$ in $\mathcal{A}$, then the decomposition described above of this valid computation gives the rules to be applied to form a derivation $(q_0, y_1) \xrightarrow{\star} u$ in G_q.

Thus we have:

$$L_{G_q}((q_1, z_1)) = Acc(q),$$

and since G_q is a right linear grammar, $Acc(q)$ is rational.

Note that the grammar G_q can be effectively computed since the condition

$$\exists x \in A^* \ : \ (p, z) \overset{x}{\models\!\!=} (p', z')$$

is an instance of the emptiness problem for a context-free language.

Considering now the general case, the proof goes along the same lines. However, we have to modify the grammar G_q in order to skip the condition that for any rule $(a, p, z, h, p') \in T$, $|h| \leq 2$.

Indeed, when symbols are definitively set in the stack at a time (there may be more than one), several symbols may be pushed that will have to be erased. The whole computation is now decomposed into:

$$(x, q_0, y_1) \overset{*}{\vdash\!\!-} (x_1, q_1, z_1') \vdash\!\!- (x_1', q_1', t_1 y_2) \overset{*}{\vdash\!\!-} (x_2, q_2, t_1 z_2') \vdash\!\!- (x_2', q_2', t_1 t_2 y_3)$$

$$\overset{*}{\vdash\!\!-} \ \cdots \ \overset{*}{\vdash\!\!-} (x_r, q_r, t_1 t_2 \cdots t_{r-1} z_r') \vdash\!\!- (x_r', q_r', t_1 t_2 \cdots t_r y_{r+1}) \overset{*}{\vdash\!\!-} (\varepsilon, q, u),$$

where $y_1, \ldots, y_{r+1}$ and $t_1, \ldots, t_{r+1}$ are now nonempty words over Z.

Define now the context-free grammar G_q with terminal alphabet Z, nonterminal alphabet $Q \times Z$, and rules:

$$\begin{array}{lll}
(p, z) \longrightarrow (p', z') & \text{if } \exists x \in A^* \ : \ (p, z) \overset{x}{\models\!\!=} (p', z') \\
(p, z) \longrightarrow t(p', z') & \text{if } \exists a \in A \cup \{\varepsilon\}, t, y \in Z^+, x \in A^*, p'' \in Q \ : \\
 & \quad (a, p, z, ty, p'') \in T \text{ and } (p'', y) \overset{x}{\models\!\!=} (p', z') \\
(p, z) \longrightarrow \varepsilon & \text{if } \exists x \in A^* \ : \ (p, z) \overset{x}{\models\!\!=} (q, \varepsilon) \\
(q, z) \longrightarrow z &
\end{array}$$

The same proof than before ensures that $L_{G_q}((q_1, z_1)) = Acc(q)$, and since G_q is a right-linear grammar, we get that $Acc(q)$ is rational. ■

We now turn to the proof of

Lemma 5.2. *The set Co-Acc(q) is rational.*

Proof. We first consider the case of the mode of acceptance by empty storage and accepting states. Let F be the set of accepting states.

Consider a valid computation $(x, q, u) \overset{*}{\vdash\!\!-} (\varepsilon, q', \varepsilon)$ with $q' \in F$ and $u = t_1 \cdots t_{r+1}$, where $t_1, \ldots, t_{r+1}$ are in Z. It can be decomposed into:

$$\begin{array}{l}
(x, q, u) \overset{*}{\vdash\!\!-} \ (x_r, p_r, t_1 t_2 \cdots t_r) \overset{*}{\vdash\!\!-} \cdots \\
\quad \cdots \overset{*}{\vdash\!\!-} \ (x_2, p_2, t_1 t_2) \overset{*}{\vdash\!\!-} (x_1, p_1, t_1) \overset{*}{\vdash\!\!-} (\varepsilon, q', \varepsilon) \, .
\end{array}$$

Define now a context-free grammar H over terminal alphabet Z, with nonterminal alphabet Q, and rules:

$$\begin{aligned} p &\longrightarrow p'z \text{ if } \exists x \in A^* \ : \ (p,z) \overset{x}{\models\!\!=} (p',\varepsilon) \\ p &\longrightarrow \varepsilon \quad \text{if } p \in F. \end{aligned}$$

Again, the grammar H can be effectively computed.

A straightforward proof by induction on the length of a derivation shows that if there is a derivation $p \xrightarrow{\star} u$ in H, then there is a valid computation $(x,p,u) \overset{*}{\longmapsto} (\varepsilon,q',u)$ with $q' \in F$ in $\mathcal{A}$.

Conversely, if there is a valid computation $(x,q,u) \overset{*}{\longmapsto} (\varepsilon,q',\varepsilon)$ in $\mathcal{A}$, then the decomposition described above of this valid computation gives the rules to be applied to form a derivation $q \xrightarrow{\star} u$ in H.

Thus we have:

$$L_H(q) = \mathit{Co\text{-}Acc}(q),$$

and since H is a left linear grammar, $\mathit{Co\text{-}Acc}(q)$ is rational.

It remains to explain how to generalize the result for any mode of acceptance, i.e. how to modify the grammar H in accordance with the mode of acceptance chosen.

Suppose that u is in $\mathit{Co\text{-}Acc}(q)$, i.e. there exists $x \in A^*$, and $(q',u') \in K$ such that $(x,q,u) \overset{*}{\longmapsto} (\varepsilon,q',u')$. If u' is not empty, there is a longest left factor v of u such that the symbols of v are not involved in this valid computation. This computation can be divided into two subsets: in the first one all but one of the symbols of u above v are deleted, the second one being the rest of the computation. If at the end of the first part, the internal configuration is (p,vz) for some pushdown symbol z, setting $u = vzw$, we then have two valid computations: $(x_1,q,w) \overset{*}{\longmapsto} (\varepsilon,p,\varepsilon)$ and $(x_2,p,z) \overset{*}{\longmapsto} (\varepsilon,q',u')$ with $x = x_1x_2$.

Hence, this leads to the (left linear) grammar H over terminal alphabet Z, with nonterminal alphabet $Q \cup \{\sigma\}$, and rules:

$$\begin{array}{lll} q \longrightarrow q'z & \text{if } \exists\ x \in A^* \ : \ (q,z) \overset{x}{\models\!\!=} (q',\varepsilon) \\ p \longrightarrow \sigma z & \text{if } \exists\ x \in A^* \ : \ (p,z) \overset{x}{\models\!\!=} (q',u') \\ \sigma \longrightarrow \sigma z \ + \ \varepsilon & \text{for all } z \in Z \\ q' \longrightarrow \varepsilon & \text{if } (q',\varepsilon) \in K\,. \end{array}$$

Again we have $L_H(q) = \mathit{Co\text{-}Acc}(q)$, hence we get that $\mathit{Co\text{-}Acc}(q)$ is rational. ■

Note that, also in the general case, the grammars G_q and H can be effectively computed.

From equation (5.2) and lemmas 1 and 2, we get that $P(\mathcal{A})$ is rational, hence the proof of the theorem is complete. ■

5.4 Pushdown transducers

A pda to which is adjoint an output is a pushdown transducer. In this paragraph, we show that the output language of a pushdown transducer is a context-free language when the given input is precisely the language recognized by the associated pda.

A *pushdown machine with output* over A is a 4-tuple $\mathcal{S} = (Q, Z, B, \gamma)$ where B is an alphabet called the output alphabet, γ is a finite subset of $(A \cup \{\varepsilon\}) \times Q \times Z \times Z^* \times Q \times B^*$, and if T is the projection of γ onto $(A \cup \{\varepsilon\}) \times Q \times Z \times Z^* \times Q$, (Q, Z, T) is a pushdown machine, called the pdm associated to $\mathcal{S}$.

We note $(h, q', u) \in \gamma(y, q, z)$ as an equivalent for $(y, q, z, h, q', u) \in \gamma$.

An internal configuration of a pushdown machine with output $\mathcal{S}$ is an internal configuration of the associated pdm. A configuration is a 4-tuple $(x, q, h, g) \in A^* \times Q \times Z^* \times B^*$, where x is the input word to be read, g is the word already output, and (q, h) is an internal configuration.

The transition relation is a relation over configurations defined the following way: there is a transition between $c = (yx, q, wz, g)$ and $c' = (x, q', wh, gu)$, where y is in $(A \cup \{\varepsilon\})$, g is in A^*, q and q' are in Q, z is in Z, w and h are in Z^*, and g and u are in B^*, and we note $c \longmapsto c'$, if $(y, q, z, h, q', u) \in \gamma$. A valid computation is an element of the reflexive and transitive closure of the transition relation, and we note $c \overset{*}{\longmapsto} c'$ a valid computation starting from c and leading to c'.

Besides T, we can derive from γ an other function from $A^* \times Q \times Z^*$ into the subsets of B^*, named the *output function* of $\mathcal{S}$, denoted μ, and defined as follows:

$$\mu(x, q, h) = \{g \in B^* \mid \exists\, q' \in Q, h' \in Z^* \ : \ (x, q, h, \varepsilon) \overset{*}{\longmapsto} (\varepsilon, q', h', g)\}\,.$$

It follows that, for $x \in A^*$, $y \in A \cup \{\varepsilon\}$, $q \in Q$, $z \in Z$ and $w \in Z^*$:

$$\mu(yx, q, wz) = \bigcup_{(y,q,z,h,q',u) \in T} u\mu(x, q', wh)\,.$$

A pushdown transducer is to a pushdown machine with output what a pda is to a pdm, i.e. it is a pushdown machine with output with specified initial and accepting configurations.

A *pushdown transducer* over A (pdt for short in the rest of the text) is a 6-tuple $\mathcal{T} = (Q, Z, B, i, K, \gamma)$ where (Q, Z, B, γ) is a pushdown machine with output, i is the internal starting configuration, and $K = F \times \{\varepsilon\}$ where F is a subset of Q, the accepting states.

If T is the projection of γ onto $(A \cup \{\varepsilon\}) \times Q \times Z \times Z^* \times Q$, then $\mathcal{A} = (Q, Z, i, K, T)$ is a pushdown automaton, called the pda associated to $\mathcal{T}$. By convention, the output of $\mathcal{T}$ in the initial configuration is the empty word.

The existence of a set K of accepting configurations leads to define a function similar to the function μ, but taking accepting configurations in account:

$$M(x, q, h) = \{g \in B^* \ \mid\ \exists\, (q', h') \in K \ : \ (x, q, h, \varepsilon) \overset{*}{\longmapsto} (\varepsilon, q', h', g)\}\,.$$

Finally, the *transduction realized* by $\mathcal{T}$ is the function Θ from A^* into the subsets of B^* defined by

$$\forall x \in A^*,\ \Theta(x) = M(x, q_1, z_1)\,.$$

Proposition 5.9. *The image through Θ of a rational language is context-free.*

We don't prove this proposition.

Consider now the following example: $\mathcal{T} = (Q, Z, B, i, K, \gamma)$ with $A = B = \{a, b, c\}$, $Z = \{z_1, X\}$, $Q = \{q_1, q_2, q_3, q_4\}$, $K = \{(\varepsilon, q_4)\}$ and γ composed of

$$\begin{array}{lll}(a, q_1, z_1, z_1X, q_1, a), & (a, q_1, X, XX, q_1, a), & (b, q_1, X, X, q_2, b),\\ (b, q_2, X, X, q_2, b), & (c, q_2, X, \varepsilon, q_3, c), & (c, q_3, X, \varepsilon, q_3, c),\\ (c, q_3, z_1, \varepsilon, q_4, c)\,. & & \end{array}$$

It is easy to see that, due to the fact that the language recognized by the associated pda is $\{a^ib^jc^{i+1} \mid i, j > 0\}$,

$$\Theta(x) = \begin{cases} a^ib^jc^{i+1} & \text{if } x = a^ib^jc^{i+1} \text{ with } i, j > 0\\ \emptyset & \text{otherwise.}\end{cases}$$

So, if $L = \{a^ib^ic^j \mid i, j > 0\}$, then $\Theta(L) = \{a^ib^ic^{i+1} \mid i > 0\}$. Hence

Fact 5.3. *The image through Θ of a context-free language is not always context-free.*

Nevertheless,

Theorem 5.4 (Evey). [14] *Given a pushdown transducer $\mathcal{T}$, if L is the context-free language recognized by the associated pda, the image $\Theta(L)$ is a context-free language.*

Proof. Let $\mathcal{T} = (Q, Z, B, i, K, \gamma)$ be a pdt over A. Define a new alphabet

$$H = \{\langle y, u\rangle \mid \exists q, z, h, q' \ :\ (y, q, z, h, q', u) \in \gamma\}$$

We can define a set of transitions T in $H \times Q \times Z \times Z^* \times Q$ by:

$$(\langle y, u\rangle, q, z, h, q') \in T \iff (y, q, z, h, q', u) \in \gamma.$$

Setting $\mathcal{A} = (Q, Z, i, K, T)$, we get a (realtime) pda over H recognizing a context-free language N over H. Finally, we consider the two morphisms π and ξ from H^* into A^* and B^* respectively, defined by:

$$\forall \langle y, u\rangle \in H,\ \pi(\langle y, u\rangle) = y \textit{ and } \xi(\langle y, u\rangle) = u.$$

It is then clear that $\pi(N)$ is the language L recognized by the associated pda, and $\xi(N)$ is equal to $\Theta(L)$. ∎

6. Subfamilies

We present here some subfamilies among the very numerous ones that have been studied in the literature. We will begin with the probably most classical one, namely

1. the family *Lin* of linear languages.

We then turn to some families derived from it

2. the family *Qrt* of quasi-rational languages
3. the family *Sqrt* of strong quasi-rational languages
4. the family *Fturn* of finite-turn languages.

We then present other subfamilies, namely

5. the families *Ocl* of one-counter languages and *Icl* of iterated counter languages
6. the family of parenthetic languages
7. the family of simple languages
8. the families of *LL* and *LR* languages.

6.1 Linear languages

The simplest way to define the family of linear languages is by grammars: a context-free grammar is *linear* if each right member of the rules contain at most one variable. A context-free language is *linear* if there exists a linear grammar generating it [10, 4].

We denote by *Lin* the family of linear languages. Naturally, the first question that arises is whether *Lin* is a proper subfamily of the family of context-free languages. This is easily seen to be true. Many proofs are possible. Here is an example of a context-free language which is not linear: let Δ be the linear language $\{a^n b^n \mid n \geq 0\}$; the language $\Delta\Delta$ is context-free but not linear. The direct proof naturally leads to a specific iteration theorem:

Theorem 6.1. *Given a linear language L, there exists an integer N_0 such that any word w in L of length at least N_0 admits a factorization $w = xuyvz$ satisfying*

(1) $xu^n yv^n z \in L \quad \forall n \in \mathbb{N}$
(2) $uv \neq 1$
(3) $|xuvz| \leq N_0$

The proof of this iteration theorem is very similar to the proof of the classical iteration theorem of Bar-Hillel, Perles and Shamir; it uses derivation trees in a grammar generating L. In the usual version, the third condition states that the length of uyv is at most N_0; here, due to the fact the grammar is linear, we may select in the derivation tree the topmost repetition instead of the lowest one. (Note that in a non linear grammar, the notion of topmost repetition does not make sense.) We leave to the reader the proof of the above theorem as well as its use to prove that $\{a^n b^n \mid n \geq 0\}\{a^m b^m \mid m \geq 0\}$ is not linear.

The linear languages can be defined in many various ways. We briefly describe here the most important ones.

6.1.1 Pushdown automata characterization. We begin by some definitions. Given a computation of a pda A, a *turn* in the computation is a move that decreases the height of the pushdown store and is preceded by a move that did not decreased it.

A pda A is said to be *one-turn* if in any computation, there is at most one turn.

Fact 6.1. *A language is linear if and only if it is recognized by a one-turn pda.*

The proof of this fact is easy: the construction of a pda from a grammar presented in the previous section on pda's gives raise to one-turn pda from a linear grammar; similarly, the construction of a grammar from a pda gives raise to a nearly linear grammar from one-turn pda: in the right member of any rule, there is at most one variable generating an infinite language. Such a grammar can easily be transformed in a linear one.

This characterization may help to prove that some languages are linear; it may be easier to describe a one-turn pda than a linear grammar for a given language. This is the case, for example, for the language over $A = \{a, b\}$

$$L = \{a^{n_1} b a^{n_2} b \cdots a^{n_k} \mid k \geq 2,\ \exists i, j,\ \ 1 \leq i < j \leq k; n_i \neq n_j\}.$$

The one-turn pda recognizing L can roughly be described as follows: the machine reads an arbitrary number of blocks $a^n b$, then it counts up the number of letters a in a block; it then reads again an arbitrary number of blocks $a^n b$, then it counts down the number of letters a checking it is not equal to the previous number of a's. Clearly, this nondeterministic machine is one-turn and recognized L, hence L is linear.

This characterization also naturally leads to consider the following question: say that a language is in the family *DetLin* if it is recognized by a deterministic one-turn pda (a one-turn dpda). Clearly, $DetLin \subset Det \cap Lin$. The question raises whether this inclusion is strict or not. The answer is yes. Here is a example : let $A = \{a, b\}$ and consider the language

$$L = \{a^n b^m a^p b^q \mid n = m \text{ or } p = q \quad n, m, p, q \geq 1\}.$$

It is easy to check that L is linear and deterministic:

On one hand, the language L is generated by the linear grammar G given by

$$\begin{array}{lll} S \rightarrow T + X & T \rightarrow aT + aT' & T' \rightarrow bT' + bT'' \\ T'' \rightarrow aT''b + ab & X \rightarrow Xb + X'b & X' \rightarrow X'a + X''a \\ X'' \rightarrow aX''b + ab & & \end{array}$$

On the other hand, the language L is recognized by the following dpda: count up the letters in the first block of a's; when entering the first block of b's, check if the number of b's is equal to the number of a's; if these two numbers are equal, read the second block of a's and of b's and accept; if they are not equal, restart the counting of letters a and b in the second block. This shows that $L \in Det \cap Lin$. However, $L \notin DetLin$; there is no deterministic one-turn pda recognizing L. Intuitively, in any deterministic pda recognizing L, the machine has to count in the stack the number of a's in the first block and then to check if the number of b's following these is the same. Hence, after reading the first block in $a^\star b^\star$, we already got at least one turn. If, at this moment, the number of a's and the number of b's happen to be different, the computation will have to count up the number of a's and to count down the number of b's in the second block, giving raise to a new turn. Hence, any deterministic pda recognizing L will be at least two-turn. It follows that

Proposition 6.1. *The family of deterministic and linear languages strictly contains the family of languages recognized by deterministic one-turn pda (or equivalently, the family of languages simultaneously deterministic and linear).*

The same linear language can be used to prove other results such as:

- L cannot be generated by a linear grammar in Greibach normal form.
- L is unambiguous but cannot be generated by an unambiguous linear grammar (showing that the inclusion $UnAmbLin \subset UnAmb \cap Lin$ is strict).

6.1.2 Algebraic characterization. Given an alphabet A, the *rational subsets* of $A^\star \times A^\star$ are defined as usual: they are the elements of the least family of subsets of $A^\star \times A^\star$ containing the finite ones and closed under union, product and star (i.e. generated submonoid). This family is denoted $Rat(A^\star \times A^\star)$. To any subset R of $A^\star \times A^\star$, we associate the language L_R over A defined by $L_R = \{u\widetilde{v} \mid (u,v) \in R\}$, where $\widetilde{v}$ denotes the reversal of v. We may then characterize the family of linear languages by the following

Proposition 6.2. *A language L over A is linear if and only if there exists a rational subset R of $A^\star \times A^\star$ such that $L = L_R$.*

Proof. Given a linear grammar $G = (V,P)$ generating L, we consider the finite alphabet

$$B = \{\langle u, \widetilde{v}\rangle \mid \exists X, Y \in V : X \rightarrow uYv \in P\} \cup \{\langle u, \varepsilon\rangle \mid \exists X \in V : X \rightarrow u \in P\}.$$

We then construct a new grammar over B as follows: to each terminal rule $X \longrightarrow u$ of the original grammar is associated the rule $X \longrightarrow \langle u, \varepsilon\rangle$ in the new grammar; to each nonterminal (linear) rule $X \longrightarrow uYv$ of the original grammar is associated the rule $X \longrightarrow \langle u, \widetilde{v}\rangle Y$. This grammar is right linear and generates a rational language K over B. Using the homomorphism transforming each letter $\langle u, \widetilde{v}\rangle$ of B in the corresponding element $(u, \widetilde{v})$ of $A^\star \times A^\star$, we get an homomorphic image R of K. So, R is rational. Then, it is immediate to prove that $L = L_R$.

Conversely, using the same homomorphism, given a rational subset R of $A^\star \times A^\star$, we can construct a right linear grammar generating R; the rules will be of the form $X \longrightarrow (u,v)Y$ or $X \longrightarrow (u,v)$ for some $u, v \in A^\star$. To such rules we associate $X \longrightarrow uY\widetilde{v}$ and $X \longrightarrow u\widetilde{v}$ respectively. The new grammar obtained is linear and generates L_R. ∎

As the rational subsets of $A^\star \times A^\star$ are exactly the rational transductions from $A^\star$ into $A^\star$, this characterization strongly connects linear languages to the theory of rational transductions and of rational cones [4].

6.1.3 Operator characterization. The above characterization can be reformulated in a slightly different way. Given a rational subset R of $A^\star \times A^\star$ and a language L over A, we define the binary operation *bracket* of R by L, denoted $[R, L]$, by

$$[R, L] = \{um\widetilde{v} \mid (u,v) \in R,\ m \in L\}.$$

A family of languages $\mathcal{F}$ is said to be *closed under bracket* if, given a language L in the family $\mathcal{F}$ and any rational set R in $A^\star \times A^\star$, $[R, L]$ is in $\mathcal{F}$. We may then state

Proposition 6.3. *The family Lin of linear languages is the smallest family of languages containing the finite sets and closed under bracket.*

Proof. Denote by Fin the family of finite languages and let $\mathcal{M} = \{[R, F] \mid R \in Rat(A^\star \times A^\star), F \in Fin\}$. Since $[K, [R, F]] = [KR, F]$, $\mathcal{M}$ is closed under bracket and is the smallest family of languages containing the finite sets and closed under bracket. Next, let L be a linear language. By Proposition 6.2, there exists a rational set R of $A^\star \times A^\star$ such that $L = L_R$; this can now be reformulated $L = [R, \{1\}]$ showing that L is in $\mathcal{M}$. Hence, we have $Lin \subset \mathcal{M}$. As we know that the family Lin contains the finite languages and is closed under bracket, we have the reverse inclusion. ∎

We shall see later that this characterization leads naturally to define some new subfamilies of the family of context-free languages.

6.2 Quasi-rational languages

One of the oldest families of languages derived from the family Lin is the family Qrt of quasi-rational languages. Again, this family can be defined in various ways, that we present now.

Definition 6.1. *The family Qrt of* quasi-rational languages *is the substitution closure of the family Lin of linear languages.*

This definition can be made more precise: we define, for k in $\mathbb{N}$, the family *Qrt(k)* by $Qrt(0) = Rat$, and $Qrt(k+1) = Lin \ \square\ Qrt(k)$, where Rat is the family of rational languages and, for two families of languages $\mathcal{L}$ and $\mathcal{M}$, the family $\mathcal{L} \ \square\ \mathcal{M}$ is the family obtained by substituting languages in $\mathcal{M}$ into languages in $\mathcal{L}$. Clearly,

$$Qrt = \bigcup_{k \in \mathbb{N}} Qrt(k).$$

It follows that $Qrt(1)$ is exactly the family Lin. It should be noted that, due to closure properties of the family Lin, one has $Lin \,\square\, Rat = Lin$. On the contrary, the inclusion $Rat \,\square\, Lin \supset Lin$ is strict.

Example 6.1. Over the alphabet $A = \{a, b\}$, we consider the linear language $L = \{a^n b^n \mid n > 0\}$. We then substitute to the letter a the linear language $L_a = \{x^n y^n \mid n > 0\}$ and to the letter b the finite language $L_b = \{z\}$. This gives raise to a quasi-rational language in $Qrt(2)$, namely $M = \{x^{n_1} y^{n_1} \cdots x^{n_k} y^{n_k} z^k \mid k > 0, n_i > 0,\ i = 1, \ldots, k\}$.

One of the first question solved was: does there exist a context-free language which is not in the family Qrt? The answer is yes, and the first proofs were direct; they proved this and two related results. The first one states that $Qrt(k)$ is strictly included in $Qrt(k+1)$. The second one states that, similarly to the case of $Lin = Qrt(1)$, we have, for each integer k, $Qrt(k) \,\square\, Rat = Qrt(k)$ and $Rat \,\square\, Qrt(k) \subset Qrt(k+1)$. We will denote $QRT(k)$ the family $Rat \,\square\, Qrt(k)$. These results can be summarized in the following chain of inclusions

$$\begin{gathered} Rat = Qrt(0) \subsetneq Lin = Qrt(1) \subsetneq QRT(1) \subsetneq Qrt(2) \subsetneq \\ \cdots \\ \subsetneq Qrt(k) \subsetneq QRT(k) \subsetneq Qrt(k+1) \subsetneq \\ \cdots \end{gathered}$$

Before explaining how these results have been proved, we turn to some characterizations of languages in Qrt used in these proofs and which are of independent interest.

Given a context-free grammar $G = (V, P)$ over A and, for each $a \in A$, a context-free grammar $G_a = (W_a, Q_a)$ over B in which the axiom is a, we construct a new grammar H over B called the *direct sum* of the grammars G and G_a for $a \in A$ as follows. The set of variables of H is the disjoint union of V and of the sets W_a for $a \in A$; the set of rules of H is the union of the sets of rules of G and of all the rules of the grammars G_a.

Using the results of the section considering grammars as equations, it is easy to see that, for each variable $X \in V$, the language generated by X in the grammar H is obtained from $L_G(X)$ by substituting to each letter a the language L_a generated by the grammar G_a. We then may repeat such an operation giving raise to an *iterated sum* of grammars. It then follows immediately that

Proposition 6.4. [40] *A language L is quasi-rational iff there exists a grammar generating L that is an iterated sum of linear grammars.*

Example 6.2. (continued) Consider the language L generated by the linear grammar $S \longrightarrow aSb + ab$ and the languages L_a generated by the linear grammar $a \longrightarrow xay + xy$ and L_b generated by the linear grammar $b \longrightarrow z$. The direct sum of these grammars is:

$$S \longrightarrow aSb + ab \quad a \longrightarrow xay + xy \quad b \longrightarrow z$$

and generates the language M of the previous example.

This characterization leads to the following new approach of quasi-rational languages. A variable S in a context-free grammar G is *expansive* if there exists a derivation $S \xrightarrow{\star} uSvSw$ for some words u, v, w. A grammar which contains no expansive variable is said to be *nonexpansive*. A language is *nonexpansive* if there exists a nonexpansive grammar generating it. Then,

Proposition 6.5. *A language is quasi-rational iff it is nonexpansive.*

This proposition explains that some authors use the term nonexpansive instead of quasi-rational. Proving that any quasi-rational language of order k is generated by a nonexpansive grammar is straightforward by induction on k: for $k = 1$, we have a linear language, thus generated by a linear grammar; such a grammar is obviously nonexpansive. Given now a language L in $Qrt(k+1)$, by definition, it is obtained by substituting to each letter a a linear language L_a in a language $M \in Qrt(k)$. By induction hypothesis, M is generated by a non-expansive grammar G; each language L_a is generated by a linear grammar. The direct sum of G and of the G_a is clearly nonexpansive.

The converse goes roughly this way: first, given a grammar G, define a preorder relation $\leq$ on the variables by setting $X \leq Y$ if there exists two words u, v and a derivation such that $X \xrightarrow{\star} uYv$. As usual, this preorder induces an equivalence relation $X \equiv Y$ iff $X \leq Y$ and $Y \leq X$. Verify then that, if G is nonexpansive, in the right member of a rule $X \longrightarrow \alpha$, there is at most one occurrence of a variable Y equivalent to X. Conclude then, using the order relation attached to the preorder $\leq$, that the grammar can be described as an iterated direct sum of linear grammars, so that the generated language is quasi-rational.

Proposition 6.5 is the result that has been used to prove directly that there exists a context-free language which is not quasi-rational (see [40, 55] for example). One of the first languages considered was the Lukasiewicz language generated by the grammar G

$$S \longrightarrow aSS + b.$$

The proofs showed that any grammar generating this language had to be expansive; the proofs were refined to exhibit, for each integer k, a language in $Qrt(k+1)$ not in $Qrt(k)$. They used the following grammars clearly related to G :

$$\begin{aligned} S_k &\longrightarrow aS_kS_{k-1} \\ &\dots \\ S_i &\longrightarrow aS_iS_{i-1} \\ &\dots \\ S_1 &\longrightarrow aS_1S_0 \\ S_0 &\longrightarrow aS_0b + b. \end{aligned}$$

These results are now proved as a consequence of a very powerful lemma (the *syntactic lemma*) which will not be presented here (see [4]).

It should be noted that, contrarily to the situation for linear languages, any quasi-rational language can be generated by a nonexpansive grammar in Greibach normal form. This follows from the construction of Rosenkrantz which preserves the nonexpansivity. On the other hand, it is an open problem to know if any unambiguous quasi-rational language can always be generated by an unambiguous nonexpansive grammar (i.e. do we have $NonAmbQrt = NonAmb \cap Qrt$?). A possibly related open problem is the following: given two quasi-rational languages, is it true that their intersection either is quasi-rational or is not context-free?

Proposition 6.5 leads to consider a new notion: the *index of a derivation* is the maximum number of occurrences of variables in the sentential forms composing it. A terminal word u has index k if, among all the derivations generating u, the one of minimum index is of index k. The grammar G is of *finite index* if the index of any generated word is bounded by a fixed integer k. Otherwise, it is of infinite index. It can be proved

Proposition 6.6. *A language is quasi-rational iff it is generated by a grammar of finite index.*

This result can be made even more precise: the family $Qrt(k)$ is exactly the family of languages generated by grammars of index k. We refer the reader to [22, 27, 47, 4] for a proof of this proposition.

6.3 Strong quasi-rational languages

We present now a less usual family of languages. It is derived from the bracket operation defined above. Recall that Lin is the smallest family closed under bracket containing the finite sets. Recall also that $Rat \,\square\, Lin$ denotes the rational closure of Lin, and denote $SQRT(1)$ this family of languages. (This family was denoted $QRT(1)$ just above.) We then define $Sqrt(2)$ as the smallest family of languages containing $SQRT(1)$ and closed under bracket. More generally, for each integer k, we define the families $SQRT(k)$ as the rational closure of $Sqrt(k)$, and $Sqrt(k+1)$ as the smallest family of languages containing $SQRT(k)$ closed under bracket. Hence, we may write

$$SQRT(k) = Rat \,\square\, Sqrt(k) \ , \quad Sqrt(k+1) = [SQRT(k)] \ ,$$

where $[\mathcal{L}]$ denotes the bracket closure of the family $\mathcal{L}$. Finally, we denote by $Sqrt$, the infinite union of the families $Sqrt(k)$. This is the family *strong quasi-rational* languages [7]. Clearly, for each k, $Sqrt(k) \subsetneq Qrt(k)$. The following more precise fact holds

Fact 6.2. *There exists a language in $Qrt(2)$ which is not in $Sqrt$.*

Such a language is the language M of the example above:

$$M = \{x^{n_1}y^{n_1}\cdots x^{n_k}y^{n_k}z^k \mid k > 0, n_i > 0, \quad i = 1, \ldots, k\}.$$

It is in $Qrt(2)$. We want to show that it does not lie in $Sqrt$.

First, we show that if $\mathcal{F}$ is any family of languages such that the language M belongs to $Rat \,\square\, \mathcal{F}$, then M is a finite union of products of languages in $\mathcal{F}$. This follows immediately from the fact that M does not contain any infinite regular set. Hence, if $M \in SQRT(k)$, M is a finite union of product of languages in $Sqrt(k-1)$. Trying to split M into a finite product of languages immediately leads to note that there is exactly one factor in the product very similar to the language M itself. Thus, if $M \in SQRT(k)$, then M belongs to the family $Sqrt(k)$.

Next, we check that if $M = [R, L]$, then R is a finite subset of $\{x, y, z\}^\star \times \{x, y, z\}^\star$. This implies that, if M belongs to the family $Sqrt(k)$, it is a finite union of products of languages lying in $SQRT(k-1)$. Again, there is one factor in this union of products very similar to M leading to the conclusion that M should lie in $SQRT(k-1)$.

Hence, we may conclude that, if M belongs to $Sqrt$, it belongs to $Sqrt(1) = Lin$. As M is not linear, the fact is proved.

Similarly to the situation for quasi-rational languages, we have

Proposition 6.7. *For each k, the family $Sqrt(k)$ is a strict subfamily of $Sqrt(k+1)$.*

6.4 Finite-turn languages

The characterization of linear languages by one-turn pda naturally leads to define finite-turn pda's and languages. A pda is *k-turn* if any computation admits at most k turns. Naturally, a language will be said to belong to the family $Fturn(k)$ if it is recognized by a k-turn pda. Then a *finite-turn* pda is a pda which is k-turn for some integer k. A language is *finite-turn* if it is recognized by a finite-turn pda [21]. It is easy to prove that 0-turn languages are rational.

The family of finite-turn languages can be described using the bracket operation too. This definition is similar to the one of strong quasi-rational languages where the rational closure is replaced by the closure under union and product. More precisely, let $Fturn_1$ be the family Lin and set, for each integer k,

$$FTURN_k = Fin \,\square\, Fturn_k \;, \quad Fturn_{k+1} = [FTURN_k] \;,$$

so that $FTURN_k$ is the closure of $Fturn_k$ under union and product and that $Fturn_{k+1}$ is the closure of $FTURN_k$ under bracket. Finally, we denote by $Fturn$ the infinite union over k in $\mathbb{N}$ the families $Fturn_k$ [21]:

$$Fturn = \bigcup_k Fturn_k = \bigcup_k FTURN_k .$$

It should be noted that the two families $Fturn_k$ and $Fturn(k)$ are not equal. For instance, let $\Delta = \{a^n b^n \mid n \geq 1\}$ and consider the language $L = \Delta^k$. It is easily seen that L is in $FTURN_1$. (So it belongs to $Fturn_2$ also.) Besides, L does not belong to $Fturn(k-1)$. So, $FTURN_1$ is not contained in $Fturn(k)$. However, the infinite union of the families $Fturn_k$ and the infinite union of the families $Fturn(k)$ coincide:

Fact 6.3. *The family $Fturn$ is exactly the family of finite-turn languages.*

Proof. It consists in showing that

(1) if L is a finite-turn language, so is $[R, L]$
(2) the family of finite-turn languages is closed under union and product.

This implies that $Fturn$ is contained in the family of finite-turn languages. Conversely, given a k-turn language, we decompose the computations of the k-turn pda recognizing it to get a description of the language through union, product and the bracket operation of $(k-1)$-turn languages, showing then the reverse inclusion. ∎

Remark 6.1. The second part of the above proof shows in fact that, for each k, we have the inclusion $Fturn(k) \subsetneq Fturn_k$.

The given characterization of finite-turn languages obviously shows that they are all strong quasi-rational languages. Here again, we get a proper subfamily :

Fact 6.4. *There exists a language in $Sqrt(1)$ which is not finite-turn.*

Such a language is, for instance, $\Delta^\star = \{a^n b^n \mid n \geq 1\}^\star$. As for the above families, we have chains of strict inclusions:

Proposition 6.8. *For each k, the family $Fturn_k$ is a strict subfamily of $Fturn_{k+1}$, and the family $Fturn(k)$ is a strict subfamily of $Fturn(k+1)$.*

6.5 Counter languages

We first present in this section the family of one-counter languages. It is defined through pda's.

Definition 6.2. *A pda is* one-counter *if the stack alphabet contains only one letter. A context-free language is a* one-counter language *if it is recognized by a one-counter pda by empty storage and accepting states.*

We denote by Ocl this family of languages. The terminology used here comes from the fact that, as soon as the stack alphabet is reduced to a single letter, the stack can be viewed a counter.

Example 6.3. Over the alphabet $A = \{a, b\}$, we consider the Lukasiewicz language generated by the grammar $S \longrightarrow aSS + b$. It is a one-counter language: each letter a increases the height of the stack by 1, each letter b decreases it by 1. The word is accepted iff it empties the stack.

As in the case of linear languages, the first question is whether Ocl is a proper subfamily of the family of context-free languages. The proof that this holds is more technical than in the case of linear languages. The idea is to prove an iteration lemma for one-counter languages and to use it to get the desired strict inclusion [6]. We will give later on such counter-examples, but we will not state this lemma which is too technical and beyond the scope of this presentation.

As in the case of linear languages, the definition of one-counter languages through pda's naturally leads to define the family $DetOcl$ as the family of languages recognized by a deterministic one-counter pda. Clearly, $DetOcl \subset Det \cap Ocl$. As in the linear case, the inclusion is strict :

Proposition 6.9. *The family of deterministic and one-counter languages strictly contains the family of languages recognized by a deterministic one-counter pda (or, equivalently, the family of languages simultaneously deterministic and one-counter).*

Proof. Over the alphabet $A = \{a, b, \#\}$, consider the language $L = \{w\#w' \mid w, w' \in \{a, b\}^\star \; w' \neq \tilde{w}\}$. We will show that L is in $Det \cap Ocl$ and is not in $DetOcl$.

It is deterministic: clearly the language $\{w\#\tilde{w} \mid w \in \{a, b\}^\star\}$ is deterministic. So, its complement C is deterministic, too. The language L is exactly the intersection of the language C and of the rational language $\{a, b\}^\star\#\{a, b\}^\star$. It follows that L is deterministic.

It is one-counter: the language L can be described as the (non disjoint) union of the two languages

$$L_1 = \{w\#w' \mid |w| \neq |w'|\}$$
$$L_2 = \{ucv\#u'dv' \mid c, d \in \{a, b\}, c \neq d, |u| = |v'|\}\,.$$

Clearly, L_1 and L_2 are one-counter languages. Thus, $L = L_1 \cup L_2$ is a one-counter language.

To see that L is not in $DetOcl$, the idea is to observe that, after reading an input w, the length of the stack is polynomially bounded in the length of the input. Since there is only one stack symbol, there exist two distinct words w and x of the same length that lead to the same (unique due to determinism) configuration. Hence, since $w\#\widetilde{x}$ is accepted, so is $x\#\widetilde{x}$, which is impossible. ∎

Remark 6.2. The last argument of the above proof can be also used to show that the linear language $\{w\#\widetilde{w} \mid w \in \{a,b\}^\star\}$ is not one-counter; it needs to prove that a one-counter pda may always be supposed to be realtime (see [18]). This shows, in particular, that the family Ocl is a strict subfamily of the family of context-free languages.

As in the case of linear languages, it can be seen that $Ocl \,\square\, Rat = Ocl$ whence the inclusion $Rat \,\square\, Ocl \subset Ocl$ is strict. This new larger family, denoted here OCL, is exactly the family of languages recognized by a one-counter pda which *admits bottom testing*. Again as in the case of linear languages, we may define the family Icl of iterated counter languages as the substitution closure of the family Ocl.

Similarly to what happened for the quasi-rational languages, this definition can be made more precise: we may define, for each k in $\mathbb{N}$, the family $Ocl(k)$ by $Ocl(0) = Rat$, and $Ocl(k+1) = Ocl(k) \,\square\, Ocl$. Then, the family Icl is the infinite union over k in $\mathbb{N}$ of the families $Ocl(k)$. Using such definitions, $Ocl(1) = Ocl$.

The study of the families $Ocl(k)$ leads naturally to prove that $Ocl(k) \,\square\, Rat = Ocl(k)$, whence $Rat \,\square\, Ocl(k) \subsetneq Ocl(k)$. This last family will naturally be denoted $OCL(k)$ and we get the following chain of strict inclusions

$$\begin{gathered}
Rat = Ocl(0) \subsetneq Ocl(1) \subsetneq OCL(1) \subsetneq Ocl(2) \subsetneq \\
\cdots \\
\subsetneq Ocl(k) \subsetneq OCL(k) \subsetneq Ocl(k+1) \subsetneq \\
\cdots
\end{gathered}$$

(To be compared to the similar chain of inclusions concerning the families $Qrt(k)$ and $QRT(k)$.)

The languages in Icl can be characterized as languages recognized by pda's such that the stack language lies in the bounded set $z_1^\star \cdots z_k^\star$.

Up to now, we may remark that the situation here is very similar to the situation we had when dealing with linear and quasi-rational languages. However, it is worth noticing that, contrarily to the case of linear languages, one-counter languages do not enjoy other characterizations through grammars or operators as linear languages did. This explains that we will not get here subfamilies similar to the strong quasi-rational languages, etc...

If we compare the families Lin and Ocl with respect to inclusion, we see that these two families are incomparable. Even more,

Proposition 6.10. *There is a language in Ocl which is not in Qrt. There is a language in Lin which is not in Icl.*

The Lukasiewicz language given above as an example of one-counter language is precisely the language proved not to be quasi-rational (see previous subsection). The second inclusion can be proved using the linear language $L = \{w\#\tilde{w} \mid w \in \{a,b\}^\star\}$ (see the previous remark).

Such a result leads to consider the two following problems: is it possible to characterize the languages in $Lin \cap Ocl$ on one hand, and, to describe any context-free language by substituting linear and one-counter languages on the other hand.

The first question is still open (see [8] for results on this theme). The second question has a negative answer: defining the family of *Greibach languages* as the substitution closure of linear and one-counter languages, we get a large strict subfamily of the family of context-free languages: it does not contain the language $D^\star$, the Dyck language over $\{a, b, \overline{a}, \overline{b}\}$ (see [4]).

6.6 Parenthetic languages

We now turn to another subfamily of the family of context-free languages. Consider an alphabet A containing two particular letters a and $\overline{a}$. A context-free grammar over the terminal alphabet A is *parenthetic* if each rule of the grammar has the following form $X \longrightarrow a\alpha\overline{a}$ with α containing neither the letter a nor the letter $\overline{a}$. As usual, a language is said to be *parenthetic* if it is generated by some parenthetic grammar.

This family of languages has been introduced by McNaughton [39]. In the particular case where the alphabet A does not contain any other letters than the two special ones a and $\overline{a}$, we speak of *pure parenthetic* grammar or language.

Example 6.4. Over the alphabet $A = \{a, \overline{a}\} \cup \{x\}$, the grammar G given by

$$S \longrightarrow aSS\overline{a} + ax\overline{a}$$

is parenthetic.

Clearly, any pure parenthetic language over $A = \{a, \overline{a}\}$ is included in the Dyck language $D_1^{\star}$. The following characterization due to Knuth [35] shows, in particular, that $D_1^{\star}$ is not (purely) parenthetic. A word u over the alphabet $A = \{a, \overline{a}\} \cup B$ is *balanced* if it satisfies $|u|_a = |u|_{\overline{a}}$ and, for any prefix v of u, $|v|_a \geq |v|_{\overline{a}}$. It should be noted that a word w is in $D_1^{\star}$ iff it is balanced.

Given a word u over the alphabet $A = \{a, \overline{a}\} \cup B$ and an occurrence of the letter a in u, we factorize u in $u = vaw$. An occurrence of a letter $b \in A$ in w is an *associate* of a iff $u = vaxby$ with xb balanced.

Example 6.5. Let $u = abab\overline{a}ab\overline{a}\overline{a}$. The first a and the first b are associates. The first a and the first $\overline{a}$ are associates too.

A language L over the alphabet $A = \{a, \overline{a}\} \cup B$, is *balanced* iff any word $u \in L$ is balanced. It is said to have *bounded associates* if there exists an integer k such that any occurrence of a letter a in $u \in L$ admits at most k associates. We may then characterize parenthetic languages as follows:

Proposition 6.11. *A context-free language L is parenthetic if and only if it is balanced and has bounded associates.*

The proof of the "only if" part is immediate. The "if" part consists in a careful study of the structure of any grammar generating a language which is balanced and have bounded associates. This study shows that the rules of the grammar may be 'recentered' to fulfill the parenthetic conditions.

Example 6.6. Consider the grammar G over $A = \{a, \overline{a}\} \cup \{b, c, d, e\}$ given by the set of rules

$$S \longrightarrow XY \quad X \longrightarrow aab\overline{a}X\overline{a} + ad \quad Y \longrightarrow aYac\overline{a}\overline{a} + e\overline{a}.$$

Clearly, this grammar is not parenthetic. However, the generated language is balanced and has bounded associates. Hence, it is a parenthetic language.

This characterization can be used to see that the Dyck set D_1^* is not parenthetic: it is balanced but it has unbounded associates. This characterization allows also to prove the following

Proposition 6.12. *If a language L is nonexpansive and parenthetic, there exists a parenthetic nonexpansive grammar generating it.*

This fact contrasts with the Proposition 6.1.

Besides this characterization, parenthetic languages enjoy some other nice properties. In particular, any such language is deterministic. Moreover, the equivalence problem for parenthetic grammars is decidable [35], solving in this particular case the equivalence problem of deterministic languages.

This family of languages can be related to the whole family of context-free languages in the following way. Given a context-free grammar $G = (V, P)$ over B, we associate to it a parenthetic grammar $Par(G)$ as follows : enlarge the terminal alphabet B into the terminal alphabet $A = B \cup \{a, \overline{a}\}$ where a and $\overline{a}$ are two new letters; to each rule $X \longrightarrow \alpha$ of G, associate the rule of $Par(G)$ given by $X \longrightarrow a\alpha\overline{a}$. It is easy to check that to each leftmost derivation in $Par(G)$ generating a word w, corresponds a leftmost derivation in G generating the word u obtained from w by erasing the new letters a and $\overline{a}$. This correspondence is a bijection between derivations. Hence, the *degree of ambiguity* of a word u in the grammar G is the number of words w generated in $Par(G)$ that map to u when the letters a and $\overline{a}$ are erased.

Example 6.7. Consider the grammar H given by $S \longrightarrow SS + x$. The corresponding parenthetic grammar $Par(H)$ is $S \longrightarrow aSS\overline{a} + ax\overline{a}$. The word xxx is the image of the two words $aax\overline{a}aax\overline{a}ax\overline{a}\overline{a}\overline{a}$ and $aaax\overline{a}ax\overline{a}\overline{a}ax\overline{a}\overline{a}$ corresponding to the two (leftmost) derivations in H

$$S \longrightarrow SS \longrightarrow xS \longrightarrow xSS \longrightarrow xxS \longrightarrow xxx$$

and

$$S \longrightarrow SS \longrightarrow SSS \longrightarrow xSS \longrightarrow xxS \longrightarrow xxx.$$

A very similar family of languages has been introduced by Ginsburg [19]. Let $k \geq 1$. Given an alphabet $A = \{a_1, \ldots, a_k\}$, we associate to it the copy $\overline{A} = \{\overline{a}_1, \ldots, \overline{a}_k\}$ and the alphabet $Z = A \cup \overline{A}$. A grammar over $Z \cup B$ with k rules is *completely parenthetic* if the i^{th} rule has the form $X \longrightarrow a_i \alpha \overline{a}_i$ with α containing no letters in Z. As usual, a language is *completely parenthetic* if there exists a completely parenthetic grammar generating it.

Clearly, if we consider the morphism from Z onto $\{a, \overline{a}\}$ erasing the indices, we get from any completely parenthetic language a parenthetic language. Such languages often appear in the study of languages attached to trees.

Example 6.8. Given the completely parenthetic grammar G' given by $S \longrightarrow a_1 SS\overline{a_1} + a_2 x \overline{a_2}$. The corresponding parenthetic grammar is the grammar G of the above example.

6.7 Simple languages

A context-free grammar $G = (V, P)$ over A is *simple* if it is in Greibach normal form and if, for each pair $(X, a) \in V \times A$, there is at most one rule of the form $X \longrightarrow am$. As usual, a language is *simple* if it can be generated by a simple grammar [28, 36]. It is easy to check that any simple language is deterministic. (It is even $LL(1)$.) It is easy too to check that there does exist deterministic (even $LL(1)$) languages which are not simple. The simple languages are exactly the languages recognized by simple deterministic pda's as defined in the previous section. Moreover, this family of languages enjoys nice properties :

1. Any simple language is prefix (i.e. if the two words u and uv are in L then v is the empty word).
2. The equivalence problem for simple grammars is decidable [36], solving again in a particular case the equivalence problem of deterministic languages.
3. The family of simple languages generates a free monoid.

Similarly to parenthetic and completely parenthetic languages, simple languages give raise to a family, namely the family of *very simple* languages. A grammar is *very simple* if it is simple and such that for any terminal letter a there is at most one rule of the form $S \longrightarrow am$. (Here, a appears as first letter of one rule at most; in the case of simple grammars, it could appear as first letter of various rules, provided they have not the same left member.)

Clearly, any very simple language is simple. The converse is not true: for instance $L = \{a^n cb^n a^m c \mid n \geq 1 \ m \geq 0\}$ is simple but not very simple. It is simple because it is generated by

$$S \longrightarrow aS'XT \quad S' \longrightarrow aS'X + c \quad T \longrightarrow aT + c \quad X \longrightarrow b.$$

To prove that L is not very simple, we show that any grammar in Greibach normal form generating L admits at least two rules whose right member begins with the letter a. Using for instance Ogden's iteration lemma on the word $a^n cb^n a^n c$ where the n first letters a are marked, we get that there is a derivation

$$S \xrightarrow{\star} a^i X b^j a^n c \quad X \xrightarrow{\star} a^k X b^k \quad X \xrightarrow{\star} a^{k'} cb^{k''};$$

from this we derive that there is a rule of the form $X \longrightarrow a\alpha$. Marking now the n last letters a, we get that there is a derivation

$$S \xrightarrow{\star} a^n cb^n a^{i'} Y a^{j'} c \quad Y \xrightarrow{\star} a^h Y a^{h'} \quad Y \xrightarrow{\star} a^{h''};$$

from this we derive that there is a rule of the form $Y \longrightarrow a\beta$.

Clearly the two variables X and Y have to be different: if $X = Y$, we may derive from $X = Y$ the word $a^k a^{h''} b^k$ which is not a factor of L. Thus, we have two different rules with a right member beginning by a, hence, the grammar cannot be very simple.

Any context-free language is an homomorphic image of a very simple language. Indeed, a context-free grammar in Chomsky normal form can be transformed in a very simple grammar by adding new terminal letters. The homomorphism erasing these new letters will reconstruct the original one. Let us mention along the same lines that, to any completely parenthetic grammar is naturally associated a very simple grammar obtained by erasing all the barred letters. Hence, any very simple language is an homomorphic image of a completely parenthetic language.

6.8 LL and LR languages

We end this survey of various classical subfamilies of the family of context-free languages by briefly presenting the two most usual subfamilies appearing in syntactical analysis. Given a word w over the alphabet A, define $First_k(w)$ as the prefix of length k of w; if $|w| < k$, $First_k(w)$ is equal to w. We may now define LL-grammars

Definition 6.3. [1, 38] *A context-free grammar $G = (V, P)$ over the terminal alphabet A is a* LL(k)-grammar *if*

$$S \xrightarrow[\ell]{\star} uXm \longrightarrow u\alpha m \xrightarrow[\ell]{\star} uv$$

$$S \xrightarrow[\ell]{\star} uXm' \longrightarrow u\alpha' m' \xrightarrow[\ell]{\star} uv'$$

(with $u, v, v' \in A^\star$ and $X \in V$) and

$$First_k(v) = First_k(v')$$

imply $\alpha = \alpha'$.

A language is a *LL(k)-language* if it can be generated by a $LL(k)$-grammar. It is a *LL-language* if it is a $LL(k)$-language for some k. The idea is that given a terminal word uv and a leftmost derivation from S into um, the first k letters of v allow to determine what is the next rule to be used in the derivation. We will not develop here this syntactical analysis technique. However, it follows clearly from this remark that any LL-language is deterministic. More precisely, the families of $LL(k)$-languages form a hierarchy. Their infinite union is a strict subfamily of the family of deterministic languages.

For instance, the language $L = \{a^n cb^n \mid n \geq 1\} \cup \{a^n db^{2n} \mid n \geq 1\}$ is clearly deterministic. It is not a LL-language: an unbounded number of letters a has to be read before it can be decided which rule to apply in an early stage of the leftmost derivation, because it depends on whether the word contains a letter c or a letter d.

Using rightmost derivations instead of leftmost derivations leads to define the LR-grammars:

Definition 6.4. [28, 34] *A context-free grammar $G = (V, P)$ over the terminal alphabet A is a* LR(k)-grammar *if,*

$$S \xrightarrow[r]{\star} mXu \longrightarrow m\alpha u = pv$$
$$S \xrightarrow[r]{\star} m'X'u' \longrightarrow m'\alpha' u' = pv'$$

(with $u, u' \in A^\star, p \in (V \cup A)^\star V$) and

$$First_k(v) = First_k(v')$$

imply $X = X'$ and $\alpha = \alpha'$.

Again, a language is a *LR(k)-language* if it is generated by a $LR(k)$-grammar. It is a *LR-language* if it is a $LR(k)$-language for some k.

The idea is the following: given a sentential form pv where v is the longest terminal suffix, the first k letters of v allows to determine the rule that has been applied just before getting the sentential form pv. Here again, this remark that we will not develop here, implies that any $LR(k)$-language is deterministic. However, the situation is now very different from the LL situation.

Proposition 6.13. *The family of $LR(1)$-languages is exactly the family of deterministic languages.*

So, from the families of languages point of view, the $LR(k)$-condition does not give raise to an infinite hierarchy. It should be noted that, in terms of grammars, it is indeed an infinite hierarchy. It should be noted also that a grammar which is not LR may generate a language which is indeed LR. It may even be rational: the grammar $S \longrightarrow aSa$, $S \longrightarrow a$ is not LR and it generates the rational language a^+.

References

1. A.V. Aho and J.D. Ullman. *The Theory of Parsing, Translation and Compiling.*, volume 1. Prentice-Hall, 1973.
2. J.-M. Autebert. *Théorie des langages et des automates.* Masson, 1994.
3. J.-M. Autebert, L. Boasson, and I.H. Sudborough. Some observations on hardest context-free languages. Technical Report 81-25, Rapport LITP, April 1981.
4. J. Berstel. *Transductions and Context-Free Languages.* Teubner Verlag, 1979.
5. M. Blattner and S. Ginsburg. Canonical forms of context-free grammars and position restricted grammar forms. In Karpinski, editor, *Fundamentals of Computing Theory*, volume 56 of *Lect. Notes Comp. Sci.*, 1977.
6. L. Boasson. Two iteration theorems for some families of languages. *J. Comput. System Sci.*, 7(6):583–596, December 1973.
7. L. Boasson, J.P. Crestin, and M. Nivat. Familles de langages translatables et fermées par crochet. *Acta Inform.*, 2:383–393, 1973.
8. F.J. Brandenburg. On the intersection of stacks and queues. *Theoret. Comput. Sci.*, 23:69–82, 1983.
9. N. Chomsky. On certain formal properties of grammars. *Inform. and Control*, 2:137–167, 1959.
10. N. Chomsky and M.P. Schützenberger. The algebraic theory of context-free languages. In P. Bradford and D. Hirschberg, editors, *Computer programming and formal systems*, pages 118–161. North-Holland (Amsterdam), 1963.
11. S.V. Cole. Deterministic pushdown store machines and realtime computation. *J. Assoc. Comput. Mach.*, 18:306–328, 1971.
12. B. Courcelle. On jump deterministic pushdown automata. *Math. Systems Theory*, 11:87–109, 1977.
13. A. Cremers and S. Ginsburg. Context-free grammar forms. *J. Comput. System Sci.*, 11:86–117, 1975.
14. R.J. Evey. The theory and application of pushdown store machines. In *Mathematical Linguistics and Automatic Translation*, NSF-IO, pages 217–255. Harvard University, May 1963.
15. M. Fliess. Transductions de séries formelles. *Discrete Math.*, 10:57–74, 1974.
16. R.W. Floyd. Syntactic analysis and operator precedence. *J. Assoc. Comput. Mach.*, 10:313–333, 1963.
17. S. Ginsburg. *The Mathematical Theory of Context-Free Languages.* McGraw-Hill, 1966.
18. S. Ginsburg, J. Goldstine, and S. Greibach. Some uniformely erasable families of languages. *Theoret. Comput. Sci.*, 2:29–44, 1976.
19. S. Ginsburg and M.A. Harrison. Bracketed context-free languages. *J. Comput. System Sci.*, 1:1–23, 1967.
20. S. Ginsburg and H. G. Rice. Two families of languages related to ALGOL. *J. Assoc. Comput. Mach.*, 9:350–371, 1962.
21. S. Ginsburg and E. Spanier. Finite-turn pushdown automata. *SIAM J. Control*, 4:429–453, 1966.
22. S. Ginsburg and E. Spanier. Derivation-bounded languages. *J. Comput. System Sci.*, 2:228–250, 1968.
23. S. Greibach. The hardest context-free language. *SIAM J. Comput.*, 2:304–310, 1973.
24. S. A. Greibach. A new normal form theorem for context-free phrase structure grammars. *J. Assoc. Comput. Mach.*, 12(1):42–52, 1965.

25. S.A. Greibach. Jump pda's, deterministic context-free languages, principal afdl's and polynomial time recognition. In *Proc. 5th Annual ACM Conf. Theory of Computing*, pages 20–28, 1973.
26. S.A. Greibach. Jump pda's and hierarchies of deterministic cf languages. *SIAM J. Comput.*, 3:111–127, 1974.
27. J. Gruska. A few remarks on the index of context-free grammars and languages. *Inform. and Control*, 19:216–223, 1971.
28. M.A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, 1978.
29. J. E. Hopcroft and J. D. Ullman. *Formal Languages and Their Relation to Automata.* Addison-Wesley, 1969.
30. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.
31. G. Hotz. Normal form transformations of context-free grammars. *Acta Cybernetica*, 4(1):65–84, 1978.
32. G. Hotz and K. Estenfeld. *Formale Sprachen.* B.I.-Wissenschaftsverlag, 1981.
33. G. Hotz and T. Kretschmer. The power of the Greibach normal form. *Elektron. Informationsverarb. Kybernet.*, 25(10):507–512, 1989.
34. D.E. Knuth. On the translation of languages from left to right. *Inform. and Control*, 8:607–639, 1965.
35. D.E. Knuth. A characterization of parenthesis languages. *Inform. and Control*, 11:269–289, 1967.
36. A.J. Korenjack and J.E. Hopcroft. Simple deterministic languages. In *Conference record of seventh annual symposium on switching and automata theory*, pages 36–46, Berkeley, 1966.
37. W. Kuich. *Formal power series*, chapter 9. This volume.
38. P.M. Lewis and R.E. Stearns. Syntax-directed transduction. *J. Assoc. Comput. Mach.*, 15(3):465–488, 1968.
39. R. McNaughton. Parenthesis grammars. *J. Assoc. Comput. Mach.*, 14(3):490–500, 1967.
40. M. Nivat. *Transductions des langages de Chomsky, Ch. VI, miméographié.* PhD thesis, Université de Paris, 1967.
41. M. Nivat. Transductions des langages de Chomsky. *Annales de l'Institut Fourier*, 18:339–456, 1968.
42. M. Oyamaguchi. The equivalence problem for realtime dpda's. *J. Assoc. Comput. Mach.*, 34:731–760, 1987.
43. R. J. Parikh. On context-free languages. *J. Assoc. Comput. Mach.*, 13:570–581, 1966.
44. D. L. Pilling. Commutative regular equations and Parikh's theorem. *J. London Math. Soc.*, 6:663–666, 1973.
45. D.J. Rosenkrantz. Matrix equations and normal forms for context-free grammars. *J. Assoc. Comput. Mach.*, 14:501–507, 1967.
46. Jacques Sakarovitch. Pushdown automata with terminal languages. In *Languages and Automata Symposium*, number 421 in Publication RIMS, Kyoto University, pages 15–29, 1981.
47. A. Salomaa. *Formal Languages.* Academic Press, 1973.
48. M. P. Schützenberger. On context-free languages and pushdown automata. *Inform. and Control*, 6:217–255, 1963.
49. G. Sénizergues. The equivalence and inclusion problems for NTS languages. *J. Comput. System Sci.*, 31:303–331, 1985.
50. G. Sénizergues. Church-Rosser controlled rewriting systems and equivalence problems for deterministic context-free languages. *Inform. Comput.*, 81:265–279, 1989.

51. E. Shamir. A representation theorem for algebraic and context-free power series in noncommuting variables. *Inform. Comput.*, 11:39–254, 1967.
52. S. Sippu and E. Soisalon-Soininen. *Parsing Theory, Vol I.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1988.
53. L. Valiant. The equivalence problem for deterministic finite turn pushdown automata. *Inform. and Control*, 25:123–133, 1974.
54. H. Wechler. Characterization of rational and algebraic power series. *RAIRO Inform. Théor.*, 17:3–11, 1983.
55. M.K. Yntema. Inclusion relations among families of context-free languages. *Inform. and Control*, 10:572–597, 1967.

Index