Dottorato di Ricerca in Informatica
I ciclo
Università di Salerno

On Drawn Symbolic Picture Languages

Carmine Gravino

Novembre 2002

Coordinatore:
Prof. A. De Santis

Relatore:
Prof.ssa F. Ferrucci

Contents

Acknowledgements

I'd like to express my gratitude to the people who, in one way or another, helped me to carry out my work. First, I want to thank prof. Filomena Ferrucci and prof. Gennaro Costagliola. Working with them, I learned much about doing research. It was an enriching experience.

I'd like to thank all my colleagues and friends. I cannot mention every single name here, but you know who you are. Thanks.

Last but not least, I want to thank my family. Their constant support has been invaluable.

Chapter 1

Introduction

In the literature, several rewriting systems have been proposed for the generation and description of *pictures* [60]. Each of these proposals considers a particular data structure for representing the pictures (a string, an array, a tree, a plex, a graph). One of the most interesting is certainly the definition of picture as given in [52]. Here an elegant description of picture languages in terms of string languages is provided. Essentially, a picture (called *drawn picture*) consists of unit lines drawn on the Cartesian plane considered as a square grid. A picture is then described by a string over the alphabet Π={u, d, r, l} encoding a walk through the picture. The symbol u (d, r, and l, respectively) is interpreted as "draw one unit line in the Cartesian plane by moving the pen *up* (*down*, *right*, and *left*, respectively) from the current position". In the last decades, such a model has been extensively investigated by providing a firm theoretical background and at the same time by establishing the intractability of several important decision problems [3, 4, 41, 42, 43, 44, 45, 52, 59, 66]. As examples, the *membership problem* has been proved to be undecidable for context-sensitive languages [52] and NP-Complete for regular languages [66] and for context-free languages [42], while the *equivalence* and *containment* problems are undecidable for regular picture languages [44]. The considerations about the intractability of such problems motivated the investigation of subclasses of drawn pictures, with nice properties from the decidability and complexity point of view [42, 46, 66].

In [12, 16] an interesting extension of drawn pictures, named drawn symbolic pictures, has been proposed. Such an extension is based on the observation that a picture may embed more information than the shape, such as colors, labels, icons etc., and is obtained by associating a symbol from an alphabet Σ to each point of the picture. Thus, a drawn

symbolic picture can be seen as a set of symbols arranged on a two-dimensional plane such that pairs of symbols on the plane are connected by unit lines to form a connected graph. Although the extension is simple and intuitive, it will turn out to be useful to describe more complex pictures. However, the introduction of symbols requires new theoretical issues to be addressed. In the thesis we will investigate such issues by theoretically analyzing drawn symbolic picture languages and providing solutions for their description, generation and recognition. As a matter of fact, a particular type of context-free grammars will be used to generate *string-based representations* of drawn symbolic pictures. Such representations, named $\Sigma \cup \Pi$-words, consist of strings whose symbols are taken from the alphabets Σ and Π. Then, for any string-based representation a corresponding drawn symbolic picture will be constructed by applying a mapping $dspic_f$ which is parametric with respect to a *merging function f*. The definition of the merging function f will allow us to solve an interesting issue determined by symbol introduction. Indeed, since a $\Sigma \cup \Pi$-word can describe any position more than once possibly with different symbols, the merging function f will be used to establish the symbol which will be actually associated to each position. Many different merging functions can be considered, making the mechanism for drawn symbolic picture description and construction to be very general.
Moreover, we will provide the notion of *symbolic picture* as a generalization of drawn symbolic picture by using the concept of *invisible lines* introduced in [36, 65]. A symbolic picture can be described by a set of points labelled with symbols from an alphabet Σ, a start point and an end point. Informally, symbolic picture languages can be considered as drawn symbolic picture languages where the spatial relations are not visualized. Even though this aspect may appear trivial, it introduces another degree of freedom in picture string descriptions which determines different complexity properties.
The approach of extending the notion of drawn picture languages to deal with (drawn) symbolic pictures has the advantage of having a considerably solid background about drawn picture languages. Indeed, these languages have been intensively studied from both decidability and computational points of view [3, 42, 44, 45, 52, 66].

In the thesis, special focus will be put on the classes of regular and context-free drawn symbolic picture languages for which several decidability, complexity and theoretical properties will be established. It is worth noting that the problems, which were intractable for drawn picture languages, turn out to be obviously intractable also for drawn symbolic picture languages. On the other hand, the nice properties (from the decidability and complexity point of view) determined for restricted subclasses of drawn picture languages not always can be inherited by the analogous subclasses of drawn symbolic picture languages. So, an interesting issue is to determine the conditions ensuring the preservation of such properties. In the thesis, several conditions will be provided in terms of properties of both string descriptions and merging functions f. Among them, the *f-independency property* can be used for one of the proposed characterizations. The property of *f-independency* of string description will be formalized and the decidability of the *f-independency problem* for context-free drawn symbolic picture grammars will be addressed. As a matter of fact, it will be proved that it is always possible to decide whether or not a context-free grammar generates only *f-independent descriptions* for drawn symbolic pictures.

In the same way, we will identify the conditions that allow to preserve other characterizations in the field of (drawn) symbolic picture languages. In particular, we will establish the properties of merging functions for which the set of (drawn) symbolic pictures equipped with the intuitive concatenation operator (simply superimpose end point and start point of the pictures and use f merging function for symbol merging) is a monoid and a finitely generated monoid. These characterizations turn out to be particularly interesting since all picture semigroups used in "picture language theory", such as drawn pictures [52], non-connected pictures [36] and connected pixel pictures [47] are finitely generated monoids. It is worth noting that the same property ensuring the characterization of the set of drawn symbolic pictures as a monoid is crucial to extend the membership results of drawn pictures in the setting of drawn symbolic pictures.

Let us observe that the goal of this thesis is twofold. On one hand, we are interested in providing more theoretical results on the symbolic picture formalism. On the other hand, we are actually providing a good theoretical framework to the *Positional Grammars* formalism as defined in [8]. This is due to the close similarity between the description of symbolic pictures and the way two-dimensional sentences are generated by a linear positional grammar [8, 9].

Positional Grammars are a formalism for the description and recognition of visual languages [14]. It provides the syntactic framework for the *Visual Language Compiler-Compiler* (*VLCC*, in short) tool for the automatic generation of compilers for several visual languages such as two-dimensional arithmetic expressions, electric circuits, combinatorial networks, entity relationship diagrams, semi-structured flowcharts, Petri nets. *VLCC* is a graphical system which is able to assist a designer in the definition of a visual language and automatically generates a corresponding visual environment. The most appealing feature of *VLCC* is that it inherits and extends to the visual field concepts and techniques of traditional compiler generation tools like *YACC* [38]. This is due to the characteristics of the Positional Grammar which represents a natural extension of the context-free grammars.

While the experimentation and the enhancements of *VLCC* are constantly growing, little analysis has been made of the theoretical aspects of the Positional Grammars formalism. So, a goal of the present thesis is to reduce this gap by investigating several decidability and complexity properties of some subclasses of linear positional grammar. The main characteristics of these classes is that they are based on the four spatial relationships *up*, *down*, *left*, and *right*. This feature turns out to be particular interesting since it allows these subclasses to be understood as a natural extension of the picture grammars introduced by Maurer et al. [52] and of the context-free grammars. Such formalization allows that the theoretical background established for picture languages and context-free languages can be exploited to get insight into the features of the Positional Grammars. Furthermore, we can "apply" the techniques and methodologies developed in the field of visual languages on drawn symbolic picture languages. Following this line, in the thesis we will provide an *LR parser* for (drawn) symbolic pictures, starting from our studies on *pLR parsing technique* [11].

Let us observe that while the immediate objective of the linear positional grammar is to treat real world problems like the parsing of iconic languages, the immediate objective for the study of drawn pictures and drawn symbolic pictures is to provide theoretical insight in the theory of formal languages. Our scope is to follow both objectives to provide a complete study on the 2D symbolic languages.

Finally, we will present a discussion on several approach proposed in literature for the description of picture languages such as non-connected picture [36], pixel-picture [47], chain code picture to describe 3D curves [6, 7], two-dimensional languages [30, 31, 32, 39, 40, 67]. For some of them a comparison with the drawn symbolic picture model is provided.

1.1 The Organization of this Thesis

The thesis is structured as follows. Chapter 2 formally defines the drawn symbolic pictures as an extension of drawn pictures, its string representation and the grammar model which generates such descriptions. A characterization of drawn symbolic pictures as finitely generated monoid is provided. Moreover, the notion of *f-independency* is introduced and some decidability results are provided. Chapter 3 focuses on the Membership problem for drawn symbolic picture languages. In the past, the considerations about the intractability of such problem for the drawn pictures has motivated the investigation of subclasses with nice properties from the decidability and complexity point of view. In this chapter we provides the conditions ensuring the preservation of the complexity properties for subclasses of drawn symbolic pictures like stripe and three-way languages. In chapter 4 we propose the notion of symbolic picture as a generalization of drawn symbolic picture by using the notion of invisible lines introduced in [36, 65]. In chapter 5, a discussion on the relationship between (drawn) symbolic pictures and Positional Grammars is presented. Moreover, we focuses on the construction of a parser for (drawn) symbolic picture grammars based on the LR technique. Finally, chapter 6 contains a discussion on several approach proposed in literature for the description of picture languages. Final remarks concludes the thesis.

Chapter 2

Drawn Symbolic Picture Languages

In this chapter we formalize the class of drawn symbolic picture languages as an extension of the drawn picture languages. The extension is based on the following observation. A drawn picture consists of unit lines drawn on the Cartesian plane. If we associate a symbol from an alphabet to each point of the picture, then we have the notion of drawn symbolic picture [16, 13]. We start by recalling the basic notions on drawn pictures (section 2.1) and introducing the formal definition of drawn symbolic pictures (section 2.2). Then, we will provide a string-based representation of drawn symbolic pictures and the notion of drawn symbolic picture grammar.

From a theoretical point of view, in section 2.3 we will identify the conditions under which the set of drawn symbolic pictures is a finitely generated monoid (like other picture semigroups used in "picture language theory" [36, 47, 52]).

In the last decades, intractability of several important decision problems for drawn picture languages has been proved. In section 2.4, such undecidability results will be easily extended to the class of drawn symbolic picture languages and an investigation of subclasses of pictures languages, with nice properties from the decidability and complexity point of view, will be provided. Moreover, other decidability problems will be exploited like the *f-independency* and the *ambiguity* problems. Finally in section 2.5 we prove the decidability of the *f-independency* problem for context-free string description grammars.

2.1 Preliminary: Drawn Pictures

We assume that the reader is familiar with the basic definitions of the Formal Language Theory [35, 28, 62]. We just recall several standard notations before reporting the basic notions concerning picture languages from [36, 47, 52].

Let A be a finite set of symbols called an alphabet. For a string w over A, $|w|$ denotes its length and if b is a letter then $\#_b(w)$ denotes the number of occurrences of b in w. The set of prefixes of w is denoted by $pref(w)$ and *suff(w)* denotes the set of suffixes of w. Analogously for a language L, $pref(L) = \bigcup_{w \in L} pref(L)$ and $suff(L) = \bigcup_{w \in L} suff(L)$. Given a finite set P, 2^P denotes the set of its subsets. As usual, Z denotes the set of integers. Given a set S and a binary operation $*$, M=(S,$*$) is a monoid if (1) $*$ is associative and (2) have a neutral element u (i.e. $u{*}a{=}a{*}u{=}a$ for all $a{\in}$S). A monoid M=(S,$*$) is inverse if for each $x{\in}$S, there exists a unique element, denoted x^{-1}, such that $x * x^{-1} * x = x$ and $x^{-1} * x * x^{-1} = x^{-1}$, and we name it inverse of x [55]. Let M be a monoid, a generating system of M is a pair (Σ, h) where Σ is an alphabet and h is an homomorphism on M such that $h(\Sigma^*)$=M [61].

In the literature the term picture has been used for several different formalisms. One of the most interesting is certainly the definition of picture as given in [52]. In agreement with such definition a picture (called *drawn picture*) consists of unit lines drawn on the Cartesian plane considered as a square grid. More formally, in the following we recall the basic concepts and notations.

The universal point set, denoted by M_0, is the Cartesian product of Z with itself. For each point $v = (m, n) \in M_0$, the up-neighbor of v, denoted by u(v), is the point $(m, n+1)$, the down-neighbor of v, denoted by d(v), is the point $(m, n-1)$, the left-neighbor of v, denoted by l(v), is the point $(m-1, n)$, the right-neighbor of v, denoted by r(v), is the point $(m+1, n)$. The neighborhood of v is defined as N(v) = {u(v), d(v), l(v), r(v)}. The universal line set M_1 is defined as the set of lines of length 1 and ends in M_0. Formally, $M_1 = \{\{v, v'\} \mid v, v' \in M_0$ and $v' \in$N$(v)\}$.

A basic picture p is a finite subset of M_1 where its points are in W(p) = $\{v \in M_0 \mid \{v, v'\}$ is in p, for some $v' \in M_0\}$. If for every v,v' $\in$W(p), with $v \neq v'$, there exist points v_0,v_1,...,v_n in W(p) such that $n \geq 1$, v_0=v,v_n=v' and $\{v_i,v_{i+1}\}\in p$ for $1 \leq i \leq n-1$, then p is connected.

A drawn picture q is a triple $q =< p, s, e >$, where p is a connected basic picture denoted by $base(q)$, and the points s and e are called *start* and *end point* of q, respectively. If p is nonempty then s and e are points in W(q) = $\{v \in M_0 \mid \{v, v'\}$ is in p, for some $v' \in M_0\}$. If p is empty then $s = e =$ (0,0), and W(q) = (0,0). Thus, the empty drawn picture is denoted by <∅, (0,0), (0,0)>.

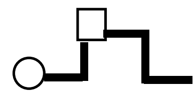

Figure 2.1: *A drawn picture.*

As an example, let us consider the drawn picture q = <{{(0,0), (1,0)}, {(1,0), (1,1)}, {(1,1), (2,1)}, {(2,1), (2,0)}, {(2,0), (3,0)}}, (0,0), (1,1)> which is depicted in Fig. 2.1. A circle is used to denote the *start point* and a square is used to denote the *end point*.

According to this definition, two drawn pictures are equivalent if one can be translated into the other by using the *translational mapping*. In particular, such mappings is used to move pictures in the plane preserving their shape, size and relative symbols. For integers m and n, the translational mapping $t_{m,n}$ is a function from M_0 to M_0, which is defined as $t_{m,n}(i,j) = (i+m, j+n)$. Several mappings are induced by $t_{m,n}$. In particular, if P is a subset of M_0, $t_{m,n}$(P) = $\{t_{m,n}(v)|v \in$P}. Similarly, if A is a subset of M_1, $t_{m,n}$(A) = $\{\{t_{m,n}(v), t_{m,n}(v')\}|\{v,v'\} \in$A}. Let q_1=$<p_1, s_1, e_1>$ and q_2 $=<p_2, s_2, e_2>$ be two drawn pictures. q_1 is translational equivalent to q_2 if there exists integers m and n such that $q_1 =< t_{m,n}(p_2), t_{m,n}(s_2), t_{m,n}(e_2) >$. Moreover, let $t_{m,n}(s_2) = e_1$ for some m,$n \in Z$, the concatenation of q_1 and q_2, denoted by $q_1 \bullet q_2$, is defined as $q_1 \bullet q_2 =< p_1 \cup t_{m,n}(p_2), s_1, t_{m,n}(e_2) >$.

A *basic picture language* is a set of basic picture and a *drawn picture language* is a set of drawn pictures.

In [52] the authors provide an elegant description of picture languages in terms of string languages. Since any point v in the plane has four neighbors (namely, u(v), d(v), l(v), and r(v)), a natural way to describe a drawn picture $q =< p, s, e >$ is to describe a walk through the picture. Such a walk starts from the start point s, touches at least once each line in b, and ends at the end point e. Each move in the walk from a point v to its neighbor v' is represented in a string by a single symbol: "u" if v' = u(v), "d" if v' = d(v), "l" if v' = l(v), "r" if v' = r(v). Thus, a walk is described by a string on the alphabet Π= {u, d, l, r}. The symbol u (d, r, and l, respectively) is interpreted as "draw one unit line in the Cartesian plane by moving the pen up (down, right, and left, respectively) from the current position". For example, the string w = rurdrlul describes the drawn picture of Fig. 2.1. The empty drawn picture <$\emptyset$, (0,0), (0,0)> is described by the empty string ϵ.

The drawn picture described by a string w over Π , denoted by $dpic(w)$, is defined inductively as follows:

- if $w = \epsilon$, then $dpic(w)$ = <$\emptyset$, (0,0), (0,0)>;
- if $w = z\pi$ for some $z \in \Pi^*$ and $\pi \in \Pi$, with $dpic(z) = < p, s, e >$ then $dpic(w) = < p \cup \{e, \pi(e)\}, s, \pi(e) >$.

Let w be a Π-word and $dpic(w)$=$< p, s, e >$ be the corresponding drawn picture. The shift of w is *shift*(w)=(x(e)-x(s), y(e)-y(s)), where for a point v=(m, n), x(v)=m and y(v)=n. As an example, given the Π-word w=rrl, the drawn picture resulting from $dpic(w)$ has start point (0,0) and end point (1,0), so *shift*(w)=(1,0).

Every string from Π^* is called a *picture description*, and every language over Π is called a *picture description language*. If G is a context-free string grammar which generates a picture description language, then we say that G is a *context-free picture description grammar*. Thus, the notion of picture grammar and language can be defined as follows. A picture grammar **G** is a pair <G, *dpic*()>, where G is a picture description grammar and *dpic*() is as above. Given a picture grammar **G** = <G, *dpic*()>, the picture language L generated by **G**, denoted by L(**G**), is L(**G**) = *dpic*(L(G)) = *dpic*(x)| x$\in$L(G).

2.2 Drawn Symbolic Pictures

In order to define a drawn symbolic picture we introduce the notion of invisible symbol ϕ which is a special symbol associated to a point that has no visual representation in a picture. In the sequel, we indicate with Σ an arbitrary alphabet which does not contain ϕ, with $\Sigma_\phi = \Sigma \cup \{\phi\}$ the set which contains ϕ, and with δ a function which may contain ϕ in its range. Thus, the definition of drawn symbolic picture can be formalized as follows.

Definition 2.1 *A drawn symbolic picture is a triple q = <sc, Σ, δ >, where sc is a drawn picture, Σ is an alphabet of symbols, and δ is a function δ: W(sc)$\rightarrow \Sigma_\phi$. The* start *and* end points *of q are the start and end points of* sc, *respectively. The empty drawn symbolic picture is denoted by $\underline{\epsilon}$=<< $\emptyset$, (0,0), (0,0)>, Σ, δ >, where δ(0,0)= ϕ.*

Let us consider the following example.

Example 2.1 *Let* q = <*sc*, Σ, δ > *be a drawn symbolic picture, where sc* = <{{*(0,0), (1,0)*}, {*(1,0), (1,1)*}, {*(1,1), (2,1)*}, {*(2,1), (2,0)*}, {*(2,0), (3,0)*}}, *(0,0), (3,0)*>, Σ = {*a, b, c*}, *and* δ *is the function such that* $\delta((0,0))$ = a, $(\delta(1,0))$ = ϕ , $\delta((1,1))$ = b, $\delta((2,1))$ = b, $\delta((2,0))$ = ϕ, $\delta((3,0))$ = a. *The visual representation of this picture is depicted in Fig. 2.2, where the start and end points are marked with a circle and a square, respectively.*

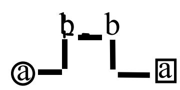

Figure 2.2: *A drawn symbolic picture.*

Example 2.2 *The electric circuit shown in Fig. 2.3(a) can be described by the drawn symbolic picture depicted in Fig. 2.3(b).*

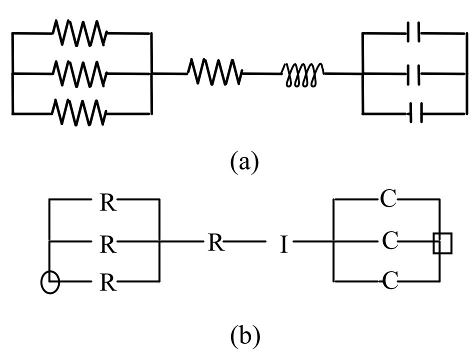

Figure 2.3: *(a) An electric circuit. (b) The corresponding drawn symbolic picture.*

Example 2.3 *The drawn symbolic picture, depicted in Fig. 2.4, describes a section of a natural rubber molecule [22]. The symbols "C" and "H", represent the carbon atom, and the hydrogen atom, resp., while symbol "2" is used to denote a double link between two carbon atoms.*

A set of drawn symbolic pictures is a *drawn symbolic picture language.*

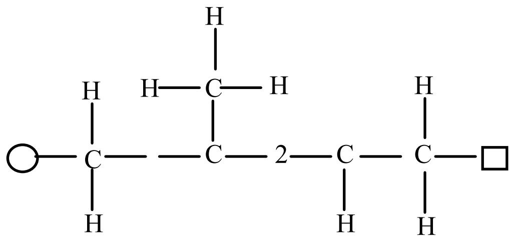

Figure 2.4: *A drawn symbolic picture describing a section of a natural rubber molecule.*

2.2.1 String Representations of Drawn Symbolic Pictures

In order to introduce a *string-based representation* of drawn symbolic pictures we need the following notations.

Given two alphabets Γ and Λ such that $\Gamma \cap \Lambda = \emptyset$, a string in the set $\Gamma \cup \Lambda$ is called a $\Gamma \cup \Lambda$-word. For example, the strings "a", "$a\mathbf{r}b\mathbf{d}c\mathbf{l}a\mathbf{r}$", "$a\mathbf{r}b\mathbf{dr}c\mathbf{l}a\mathbf{rr}b$" and "$a\mathbf{r}bbc\mathbf{dl}a$" are words in the set $(\Gamma \cup \Lambda)^*$= ({a, b, c} $\cup$ {**u**, **d**, **l**, **r**})*, where symbols from the two sets are shown using different styles for sake of clarity. A set of $\Gamma \cup \Lambda$-words is called a $\Gamma \cup \Lambda$-language.

The description of a drawn symbolic picture can be given in analogy with the description of a drawn picture by describing a walk through the picture. Such a walk starts from the start point, touches at least once each line and symbol in the drawn symbolic picture, and ends at the end point. Thus, such a walk is described by a $\Sigma \cup \Pi$-word, where Σ is the alphabet of symbols and Π contains the moves {**u**, **d**, **l**, **r**}. As an example, given the drawn symbolic picture depicted in Fig. 2.2, a possible walk is described by the $\Sigma \cup \Pi$-word "$a\mathbf{ru}b\mathbf{r}b\mathbf{dr}a$". Note that the symbol ϕ is not present in the word.

Although the extension is simple and intuitive, the introduction of symbols requires new theoretical issues to be addressed. As a matter of fact, let us observe that any Π-word describes one and only one picture. This is not the case for $\Sigma \cup \Pi$-words. As an example, the $\Sigma \cup \Pi$-word w=$a\mathbf{r}b\mathbf{u}c\mathbf{d}g$ can specify both the two pictures depicted in Fig. 2.5(i) and Fig. 2.5(ii). Indeed, w specifies that two symbols b and g might be associated to the point (1,0). From the definition of drawn symbolic picture only one symbol can be associated to each point, thus it is necessary to establish which symbol must be associated to each point. For the point (1,0) this symbol could be b or g, or a symbol that summarizes the information of b and g. In general, this symbol could be the image of b and g by a function *f*. So, if the function f is such that $f(b,g)=b$ then w describes the drawn symbolic picture depicted in Fig. 2.5(i), if $f(b,g)=g$ then w describes the drawn symbolic picture of Fig. 2.5(ii), otherwise if $f(b,g)=e$ then w describes the drawn symbolic picture of Fig. 2.5(iii).

In the sequel, we name such a function f the *merging function* which is formally defined as follows.

Definition 2.2 *Let Σ_ϕ be an alphabet of the symbols. A merging function f on Σ_ϕ is a mapping $\Sigma_\phi \times \Sigma_\phi \rightarrow \Sigma_\phi$ such that:*

- *f is a total function, and*
- *$f(\phi\ ,a)= f(a,\ \phi)=$a for each $a\in \Sigma_\phi$.*

Now we are ready to provide the definition of drawn symbolic picture described by a $\Sigma\cup\Pi$-word w with respect to a merging function *f*, denoted by $dspic_f(w)$.

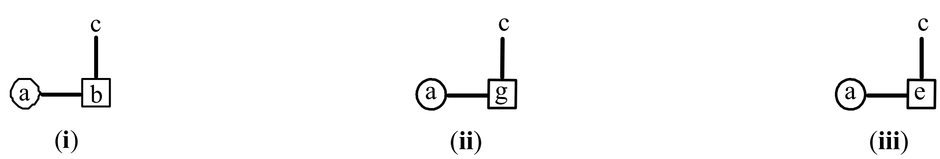

Figure 2.5: *Three drawn symbolic pictures described by* w=a$\boldsymbol{r}$b$\boldsymbol{u}$c$\boldsymbol{d}$g.

Definition 2.3 *Let w be a $\Sigma\cup\Pi$-word and f a merging function on Σ_ϕ. The drawn symbolic picture described by w w.r.t. the merging function f, denoted by $dspic_f(w)$, is defined inductively as follows:*

- *if $w=\epsilon$, then $dspic_f(w)$ is the empty drawn symbolic picture $\underline{\epsilon}$=<< $\emptyset$, (0,0), (0,0)>, Σ, δ >, where δ(0,0)= ϕ;*
- *if $w=w'\tau$ for some $w'\in(\Sigma\cup\Pi)$*, with $dspic_f(w')=<sc,\Sigma,\delta>$, $sc=<p,s,e>$,*

$$dspic_f(w)=\begin{cases} << p,s,e>,\Sigma,\underline{\delta}_1> & \text{if } \tau\in\Sigma \\ << p\cup\{e,\tau(e)\},s,\tau(e)>,\Sigma,\underline{\delta}_2> & \text{if } \tau\in\Pi \end{cases}$$

where $\underline{\delta}_1$: $W(sc)\rightarrow\Sigma_\phi$ is the labeling function such that:

$$\underline{\delta}_1(v)=\begin{cases} f(\delta(v),\tau) & \text{if } v=e \\ \delta(v) & \text{otherwise} \end{cases}$$

where $\underline{\delta}_2$*:* $W(sc) \cup \{\pi(e)\} \rightarrow \Sigma_\phi$ *is the labeling function such that:*

$$\underline{\delta}_2(v) = \begin{cases} \phi & \textit{if } v \notin W(sc) \\ \delta(v) & \textit{otherwise} \end{cases}$$

Informally, $dspic_f(w)$ constructs a drawn symbolic picture by scanning the string w from left to right. The initial position in the plane is set to (0,0) and a pointer p points to the first symbol in w. Whenever p points to a symbol $\sigma \in \Sigma$, and ρ is the symbol associated to the current position in the plane then $f(\sigma, \rho)$ is written on that position, and p is moved to the next symbol in w. Whenever p points to a symbol $\pi \in \Pi$, then the current position on the plane is updated in agreement with π and ϕ is associated to that position if it is new. Then, as usual p points to the next symbol in w.

Let us consider the following example.

Example 2.4 *Given the* $\Sigma \cup \Pi$*-word* w = $a\boldsymbol{u}be\boldsymbol{r}a\boldsymbol{d}\boldsymbol{l}c\boldsymbol{r}b\boldsymbol{l}e$*, with* $\Sigma = \{a, b, c, e\}$*, and* $f{:}\Sigma_\phi \times \Sigma_\phi \rightarrow \Sigma_\phi$ *a total function, then* $dspic_f(w)$ = $<< \{\{(0,0),(0,1)\},\{(0,1),(1,1)\},\{(1,1),(1,0)\},\{(1,0),(0,0)\}\},(0,0),(0,0) >, \Sigma, \delta >$ *where* δ *is shown in Table 2.1. The drawn symbolic picture* $dspic_f(w)$ *is obtained by applying* $dspic_f()$ *on the prefixes of* w *as follows:*

- $dspic_f(a)$ = $<< \emptyset$, (0,0), (0,0)$>, \Sigma, \delta^1 >$, *visualized as in Fig. 2.6(1);*
- $dspic_f(a\boldsymbol{u})$ = $<<\{\{(0,0), (0,1)\}\}, (0,0), (0,1)>, \Sigma, \delta^2 >$, *visualized as in Fig. 2.6(2);*
- $dspic_f(a\boldsymbol{u}b)$= $<<\{\{(0,0), (0,1)\}\}, (0,0), (0,1)>, \Sigma, \delta^3 >$, *visualized as in Fig 2.6(3);*
- $dspic_f(a\boldsymbol{u}be)$= $<<\{\{(0,0), (0,1)\}\}, (0,0), (0,1)>, \Sigma, \delta^4 >$, *visualized as in Fig 2.6(4);*
- $dspic_f(a\boldsymbol{u}be\boldsymbol{r})$ = $<<\{\{(0,0), (0,1)\}, \{(0,1), (1,1)\}\}, (0,0), (1,1)>, \Sigma, \delta^5 >$, *visualized as in Fig 2.6(5);*
- $dspic_f(a\boldsymbol{u}be\boldsymbol{r}a)$ = $<<\{\{(0,0), (0,1)\}, \{(0,1), (1,1)\}\}, (0,0), (1,1)>, \Sigma, \delta^6 >$, *visualized as in Fig 2.6(6);*

- $dspic_f(a\mathbf{u}be\mathbf{r}a\mathbf{d}) = <<\{\{(0,0), (0,1)\}, \{(0,1), (1,1)\}, \{(1,1), (1,0)\}\}, (0,0), (1,0)>, \Sigma, \delta^7 >$, *visualized as in Fig 2.6(7);*

- $dspic_f(a\mathbf{u}be\mathbf{r}a\mathbf{dl}) = <<\{\{(0,0), (0,1)\}, \{(0,1), (1,1)\}, \{(1,1), (1,0)\}, \{(1,0), (0,0)\}\}, (0,0), (0,0)>, \Sigma, \delta^8 >$, *visualized as in Fig 2.6(8);*

- $dspic_f(a\mathbf{u}be\mathbf{r}a\mathbf{dl}c) = <<\{\{(0,0), (0,1)\}, \{(0,1), (1,1)\}, \{(1,1), (1,0)\}, \{(1,0), (0,0)\}\}, (0,0), (0,0)>, \Sigma, \delta^9 >$, *visualized as in Fig 2.6(9);*

- $dspic_f(a\mathbf{u}be\mathbf{r}a\mathbf{dl}c\mathbf{r}) = <<\{\{(0,0), (0,1)\}, \{(0,1), (1,1)\}, \{(1,1), (1,0)\}, \{(1,0), (0,0)\}\}, (0,0), (1,0)>, \Sigma, \delta^{10} >$, *visualized as in Fig 2.6(10);*

- $dspic_f(a\mathbf{u}be\mathbf{r}a\mathbf{dl}c\mathbf{r}b) = <<\{\{(0,0), (0,1)\}, \{(0,1), (1,1)\}, \{(1,1), (1,0)\}, \{(1,0), (0,0)\}\}, (0,0), (1,0)>, \Sigma, \delta^{11} >$, *visualized as in Fig 2.6(11);*

- $dspic_f(a\mathbf{u}be\mathbf{r}a\mathbf{dl}c\mathbf{r}b\mathbf{l}) = <<\{\{(0,0), (0,1)\}, \{(0,1), (1,1)\}, \{(1,1), (1,0)\}, \{(1,0), (0,0)\}\}, (0,0), (0,0)>, \Sigma, \delta^{12} >$, *visualized as in Fig 2.6(12);*

- $dspic_f(a\mathbf{u}be\mathbf{r}a\mathbf{dl}c\mathbf{r}b\mathbf{l}e) = <<\{\{(0,0), (0,1)\}, \{(0,1), (1,1)\}, \{(1,1), (1,0)\}, \{(1,0), (0,0)\}\}, (0,0), (0,0)>, \Sigma, \delta^{13} >$, *visualized as in Fig 2.6(13);*

Table 2.1: *The function δ of example 2.4.*

	δ^1	δ^2	δ^3	δ^4	δ^5	δ^6	δ^7	δ^8	δ^9	δ^{10}	δ^{11}	δ^{12}	δ^{13}
(0,0)	*a*	*a*	*a*	*a*	*a*	*a*	*a*	*a*	*f(a,c)*	*f(a,c)*	*f(a,c)*	*f(a,c)*	*f(f(a,c),e)*
(0,1)	-	ϕ	*b*	*f(b,e)*	*f(b,e)*	*f(b,e)*	*f(b,e)*	*f(b,e)*	*f(b,e)*	*f(b,e)*	*f(b,e)*	*f(b,e)*	*f(b,e)*
(1,1)	-	-	-	-	ϕ	*a*	*a*	*a*	*a*	*a*	*a*	*a*	*a*
(1,0)	-	-	-	-	-	-	ϕ	ϕ	ϕ	ϕ	*b*	*b*	*b*

Different merging functions f can be considered. As an example, we can define a function f which always selects for a position the first symbol associated during scanning and ignores other possible symbols. In the following we denote such a function f with i_1. For the example 2.4 if $i_1(i_1(a,c),e) = a$ and $i_1(b,e) = b$ then the picture described by the $\Sigma \cup \Pi$-word w = $a\mathbf{u}be\mathbf{r}a\mathbf{dl}c\mathbf{r}b\mathbf{l}e$, is the one depicted in Fig. 2.7.

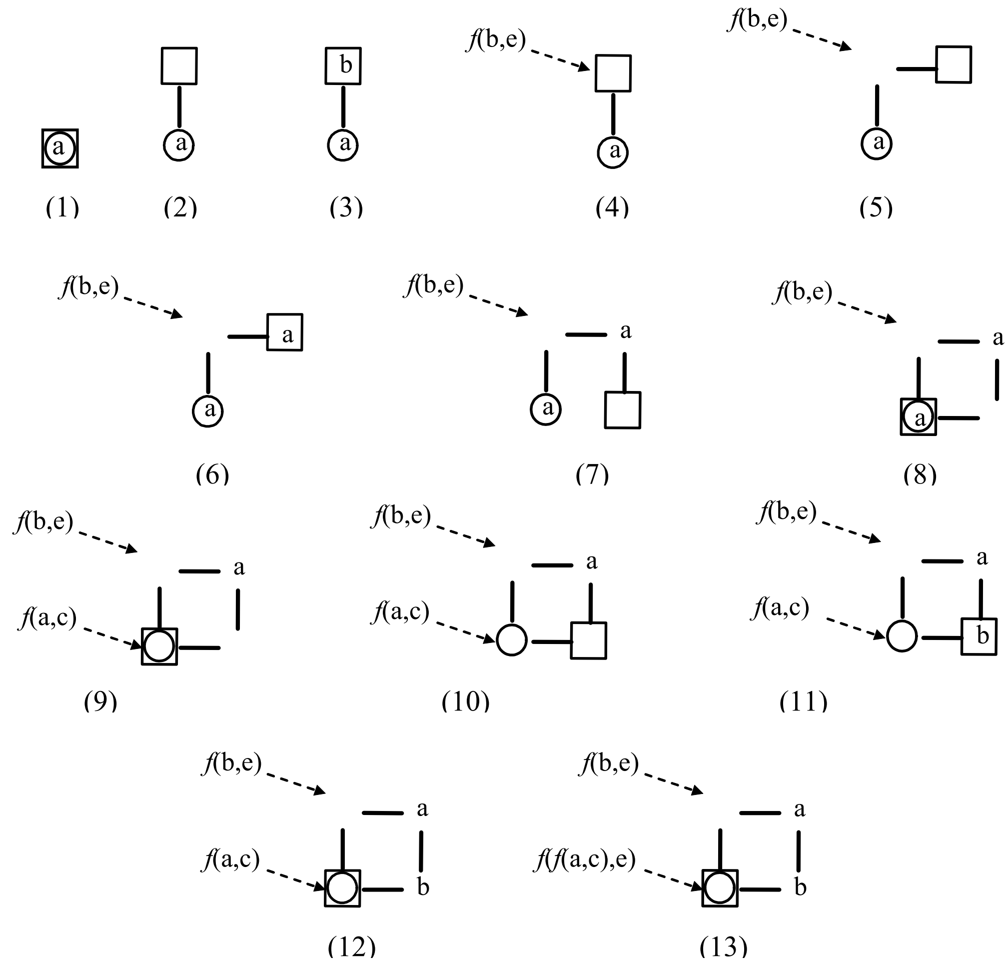

Figure 2.6: *The application of* $dspic_f()$ *on the prefixes of "a**u**be**r**a**dl**c**r**b**l**e"*

As another example we can consider a function f that uses a total order relation $<$ on Σ_ϕ, such that $f(a,b) = b$ if and only if $a < b$. Obviously, $\phi < a$ for each $a \in \Sigma$. In the following we denote such a function f with *ord*.

In the example 2.4, if we suppose that $\phi < a < e < b < c$ is the total order relation used by *ord* on Σ, then $c = ord(ord(a,c),e)$ and $b = ord(b,e)$ and the picture obtained is depicted in Fig. 2.8.

The merging functions f introduced above are selection functions since they select one of the input symbols. We could consider more general merging functions f that map two symbols $a, b \in \Sigma$ into a generic symbol $c \in \Sigma$. As an example, let us consider a drawn symbolic picture language where the symbols associated to points represent colors in *RGB*

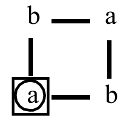

Figure 2.7: *The drawn symbolic picture obtained by* $dspic_{i1}$*(auberadlcrble).*

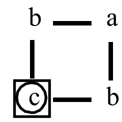

Figure 2.8: *The drawn symbolic picture obtained by* $dspic_{ord}$*(auberadlcrble).*

representation, i.e. triples (r, g, b) where each component is a value in the set {0, 1,..., 255}. Whenever a description associates two different colors to the same position then the composition of them is visualized. Hence, the function f used by $dspic_f()$ is the additive function for the *RGB* representation. In particular, given two colors c_1=(r_1,g_1,b_1) and c_2=(r_2,g_2,b_2) then f(c_1,c_2)=((r_1*r_2)/255, (g_1*g_2)/255, (b_1*b_2)/255). In the following we denote such function with *addRGB*. As an example, given the $\Sigma \cup \Pi$-word w = (165,165,165)**u**(150,150,150)**r**(188,188,188)**l**(204, 204,204) the corresponding drawn symbolic picture is depicted in Fig. 2.9, where the colors of the picture are gray levels.

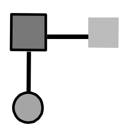

Figure 2.9: *The drawn symbolic picture obtained by* $dspic_{addRGB}(w)$*.*

As usual, the definition of $dspic_f()$ can be easily extended to the languages of $\Sigma \cup \Pi$-words in a natural manner, i.e. given a $\Sigma \cup \Pi$-language S, $dspic_f$(S)={$dspic_f(w)$ | $w \in$S }.

2.2.2 Drawn Symbolic Picture Grammars

As shown in the previous subsection, a drawn symbolic picture can be constructed by applying the function $dspic_f()$ to a $\Sigma \cup \Pi$-word. In agreement with this approach, a grammar generating drawn symbolic picture descriptions must be able to generate strings of symbols and moves. Given two alphabets Γ and Λ a $\Gamma \cup \Lambda$-grammar is a string grammar that generates a $\Gamma \cup \Lambda$-language. More formally, we have the following definitions:

Definition 2.4 *Let Γ and Λ be two disjoint finite sets of symbols, a $\Gamma \cup \Lambda$-grammar is specified by a 5-tuple $<\Gamma, \Lambda, N, P, S>$, and is defined as a $\Gamma \cup \Lambda$-grammar $G = <\Gamma \cup \Lambda, N, P, S>$ for which $L(G) \subseteq (\Gamma \cup \Lambda)^*$.*

In the sequel, a $\Sigma \cup \Pi$-grammar, denoted by $G_{\Sigma \cup \Pi}$, will be used to generate drawn symbolic picture descriptions. Thus, the drawn symbolic picture grammars and languages are defined as follows.

Definition 2.5 *A drawn symbolic picture grammar G is a pair $<G_{\Sigma \cup \Pi}, dspic_f()>$, where $G_{\Sigma \cup \Pi}$ is a $\Sigma \cup \Pi$-grammar and $dspic_f()$ is the function that translates a $\Sigma \cup \Pi$-word into a drawn symbolic picture in agreement with a total function $f{:}\Sigma_\phi \times \Sigma_\phi \rightarrow \Sigma_\phi$. Given a drawn symbolic grammar $G = <G_{\Sigma \cup \Pi}, dspic_f()>$, the drawn symbolic picture language L generated by G, denoted by L(G), is:*

$$L(G) = dspic_f(L(G_{\Sigma \cup \Pi})) = \{dspic_f(x) \mid x \in L(G_{\Sigma \cup \Pi})\}.$$

Example 2.5 *Let $G = <G_{\Sigma \cup \Pi}, dspic_f()>$ be a drawn symbolic grammar, where ord is the total order relation $\phi < a < e < b < c$ and $G_{\Sigma \cup \Pi}$ is the $\Sigma \cup \Pi$-grammar specified as follows: $G_{\Sigma \cup \Pi} = <\{a, b, c, e\}, \Pi, \{S, B, C\}, P, S>$ where S is the initial symbol and P contains the productions: $S \rightarrow a\ \boldsymbol{u}\ b\ B$, $B \rightarrow e\ \boldsymbol{r}\ C\ \boldsymbol{r}\ b\ \boldsymbol{l}\ e$, $C \rightarrow a\ \boldsymbol{d}\ \boldsymbol{l}\ c$, $C \rightarrow \boldsymbol{d}\ C$. L(G) contains the drawn symbolic picture of Fig 2.8, indeed $G_{\Sigma \cup \Pi}$ generates the $\Sigma \cup \Pi$-word "$a\boldsymbol{u}be\boldsymbol{r}a\boldsymbol{dl}c\boldsymbol{r}b\boldsymbol{l}e$".*

A drawn symbolic picture language L is *regular* (*context-free*) if there exists a *regular* (*context-free*) drawn symbolic picture grammar G that generates L. A drawn symbolic picture grammar G=$<G_{\Sigma \cup \Pi}, dspic_f()>$ is *regular* (*context-free*) if $G_{\Sigma \cup \Pi}$ is a *regular* (*context-free*) drawn symbolic picture description grammar.

2.3 The Drawn Symbolic Picture Set: a Finitely Generated Monoid

In the previuos section, we have defined the class of drawn symbolic picture languages as an extension of the drawn picture languages. It is worth noting that the set of drawn pictures as defined in [52] is a finitely generated monoid. Moreover, such a property holds for all picture semigroups used in "picture language theory", such as non-connected pictures [36] and connected pixel pictures [47].

In this section we provide an analogous characterization for the set of drawn symbolic pictures. In particular we will identify the properties of the merging function f ensuring that the set of drawn symbolic pictures is a finitely generated monoid. As a consequence of such characterization several results for the class of regular drawn symbolic picture languages can be easily obtained.

We start by introducing the *concatenation* operator on drawn symbolic pictures and showing that the set of drawn symbolic pictures together with such operator is a monoid. In analogous way to concatenation between drawn pictures, the concatenation of two drawn symbolic pictures can be obtained by overlapping the end point of the first picture with the start point of the second picture. However, the presence of symbols requires a suitable extension of such procedure to determine the symbol associated to each overlapping position. In order to clarify such need, let us consider the two drawn symbolic pictures $q_1 =<< p_1, s_1, e_1 >, \Sigma, \delta_1 >$ and $q_2 =<< p_2, s_2, e_2 >, \Sigma, \delta_2 >$ depicted in Figg. 2.10(i) and 2.10(ii). Let us observe that the drawn symbolic picture obtained by concatenating q_1 and q_2 presents two overlapping positions. As a matter of fact, (2,0) and (3,0) do not only belong to p_1 but also to $t_{3,0}(p_2)$ since there exist points (-1,0) and (0,0) in p_2. Since both δ_1 and δ_2 provide a symbol for these positions we can exploit a merging function f to determine the symbol to be associated to these points. As an example, if the merging function f is such that $f(a, e) = a$ and $f(b, g) = e$, then we obtain the drawn symbolic picture depicted in Fig. 2.10(iii).

Thus, the concatenation operator can be defined w.r.t a merging function. In the following, we provide the formal definition.

Definition 2.6 *Let* $q_1 =<< p_1, s_1, e_1 >, \Sigma, \delta_1 >$ *and* $q_2 = <<p_2, s_2, e_2 >, \Sigma, \delta_2 >$ *be two drawn symbolic pictures. Let* m, $n \in Z$ *be such that* $t_{m,n}(s_2) = e_1$, *and* f *is a merging*

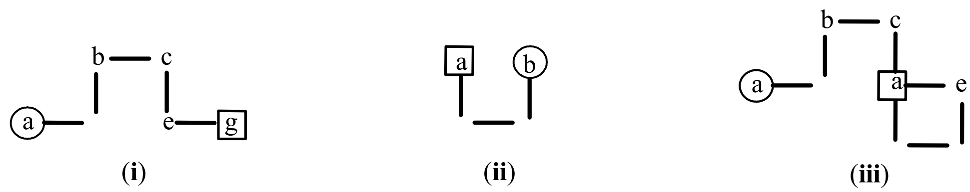

Figure 2.10: *Two drawn symbolic picture (i) and their concatenation (ii)*

function on Σ_ϕ. *The* concatenation *of* q_1 *and* q_2 *w.r.t. f, denoted by* $q_1 \bullet_f q_2$, *is defined as*

$$q_1 \bullet_f q_2 \ = << p_1 \cup t_{m,n}(p_2),\ s_1,\ t_{m,n}(e_2) >,\ \Sigma,\ \delta >$$

where $\delta : W(< p_1 \cup t_{m,n}(p_2), s_1, t_{m,n}(e_2) >) \rightarrow \Sigma_\phi$ *is such that*

$$\delta(v) = \begin{cases} \delta_1(v) & \textit{if } (v \in W(< p_1, s_1, e_1 >)) \textit{ and } (v \notin W(t_{m,n}(< p_2, s_2, e_2 >))) \\ \delta_2(t_{-m,-n}(v)) & \textit{if } (v \notin W(< p_1, s_1, e_1 >)) \textit{ and } (v \in W(t_{m,n}(< p_2, s_2, e_2 >))) \\ f(\delta_1(v), \delta_2(t_{-m,-n}(v))) & \textit{if } (v \in W(< p_1, s_1, e_1 >)) \textit{ and } (v \in W(t_{m,n}(< p_2, s_2, e_2 >))) \end{cases}$$

It is easy to verify that the concatenation of two drawn symbolic pictures is always defined and it is unique with respect to a given merging function *f*.

In the sequel we provide the properties on merging function f ensuring that the set of drawn symbolic pictures equipped with the concatenation operator $\bullet_f$ is a monoid and a finitely generated inverse monoid.

Proposition 2.1 *Let DSP be the set of drawn symbolic pictures on a set* Σ_ϕ *and* $\bullet_f$ *be the concatenation operator w.r.t. f. If f is associative then (DSP,*$\bullet_f$*) is a monoid.*

Proof. *Let us observe that the associative property of concatenation operator directly follows from the associative property of f. Moreover, it is easy to verify that the empty drawn symbolic picture* $\underline{\epsilon}$*=<<* $\emptyset$*, (0,0), (0,0)>,* Σ*,* δ *> is the neutral element of the concatenation operator.*

Let us observe that by definition of merging function, f can be considered as an operator on Σ_ϕ: thus if f is associative then (Σ_ϕ, f) is a monoid. So, we could state Proposition 2.1 as follows:

"Let DSP be the set of drawn symbolic pictures on a set Σ_ϕ and $\bullet_f$ be the concatenation operator w.r.t *f*. If (Σ_ϕ, f) is a monoid then (DSP,$\bullet_f$) is a monoid."

The proposition 2.2 states that (DSP,$\bullet_f$) is a finitely generated monoid.

Proposition 2.2 *Let DSP be the set of drawn symbolic pictures on a set Σ_ϕ and $\bullet_f$ be the concatenation operator w.r.t f such that (Σ_ϕ, f) is a monoid. (DSP, $\bullet_f$) is a finitely generated monoid.*

Proof. *Since (Σ_ϕ, f) is a monoid, from proposition 2.1 it follows that (DSP, $\bullet_f$) is a monoid. We prove that it is finitely generated by induction on the length of the drawn symbolic pictures. The basis of induction is given by the drawn symbolic pictures of length 0 and 1. In this case the thesis can be easily proved. Indeed, the drawn symbolic pictures $<< \emptyset$, (0,0), (0,0)$>$, Σ, $\delta >$ and $<<\{$(0,0), π(0,0)$\}$, (0,0), π(0,0)$>$, Σ, $\delta >$ are finitely generated since the alphabet Σ and the set of moves are finite sets. Let us suppose that the thesis holds for drawn symbolic pictures of length n and prove that it also holds for drawn symbolic pictures of length n+1. It can be easily verified from the definition of concatenation, that a drawn symbolic picture q, with $|q| = n+1$, can be obtained as the concatenation of two drawn symbolic pictures q_1 and q_2 such that q=$q_1 \bullet_f q_2$ with $|q_1| \leq n$ and $|q_2| \leq n$. By inductive hypothesis q_1 and q_1 are finitely generated so q is finitely generated.*

Theorem 2.1 *Let DSP be the set of drawn symbolic pictures on a set Σ_ϕ and $\bullet_f$ be the concatenation operator w.r.t f. If (Σ_ϕ, f) is a monoid then the mapping function $dspic_f$ is a morphism from the free monoid $(\Sigma \cup \Pi)^*$ into the monoid (DSP, $\bullet_f$).*

Proof. *It is easy to verify that: 1) $dspic_f(\epsilon)$=ϕ and for all w_1, $w_2 \in (\Sigma \cup \Pi)^*$ $dspic_f(w_1 \cdot w_2)$ = $dspic_f(w_1) \bullet_f dspic_f(w_2)$.*

As a consequence, according to finitely generated monoid representation theory [57], each $\Sigma \cup \Pi$-word represents a drawn symbolic picture and ($\Sigma \cup \Pi$, $dspic_f$) is the generating system of (DSP, $\bullet_f$).

It is worth noting that given a drawn symbolic picture q and a merging function *f*, there may exist many different $\Sigma \cup \Pi$-words w describing q (i.e. such that $dspic_f(w)$=p). As an example both the $\Sigma \cup \Pi$-words w_1=$a\mathbf{ru}b\mathbf{r}c\mathbf{d}e\mathbf{r}g\mathbf{lr}$ and w_2=$a\mathbf{ru}b\mathbf{r}c\mathbf{d}e\mathbf{r}g$ describe the drawn symbolic picture depicted in Fig. 2.10(i). As described in [47] we can define the congruence associated to the generating system ($\Sigma \cup \Pi$,$dspic_f$), denoted by $\equiv_f$, such that:

$$w \equiv_f w' \text{ if and only if } dspic_f(w) = dspic_f(w') \text{ for } w,w' \in (\Sigma \cup \Pi)^*$$

Thus, given a drawn symbolic picture q, let $w \in (\Sigma \cup \Pi)^*$ such that $dspic_f(w)=q$. The *description language* of q w.r.t f is the set

$$[w]_{\equiv_f} = \{w' \in (\Sigma \cup \Pi)^* \mid w \equiv_f w' \}$$

It can be easily proved that each equivalence class of $\equiv_f$ is a rational language of $(\Sigma \cup \Pi)^*$ [47]. This property immediately implies that the membership problem is decidable for each subset S=$dspic_f(\Gamma) \subseteq$DSP such that $\Gamma \subseteq (\Sigma \cup \Pi)^*$ is obtained by applying rational operations on singleton of $(\Sigma \cup \Pi)^*$. As another consequence of the previous observations we can state the following theorem, which provides a set-theoretical characterization for regular drawn symbolic picture languages.

Theorem 2.2 *Let DSP be the set of drawn symbolic pictures on a set Σ_ϕ and $\bullet_f$ be the concatenation operator w.r.t f such that (Σ_ϕ, f) is a monoid. A drawn symbolic picture language K$\subseteq$DSP is regular if and only if there exists a nonempty regular $\Sigma \cup \Pi$-language L such that K=$dspic_f(L)$.*

Let us observe that all picture semigroups used in picture language theory are inverse monoids. In the following we characterize function f for which the set of drawn symbolic picture is an inverse monoid.

Let us recall that a monoid M=(S,$*$) is inverse if for each $x \in$S, there exists a unique element, denoted x^{-1}, such that $x * x^{-1} * x = x$ and $x^{-1} * x * x^{-1} = x^{-1}$. The element x^{-1} is named inverse of x [55].

Proposition 2.3 provides the conditions ensuring that monoid (DSP,$\bullet_f$) is inverse.

Proposition 2.3 *Let DSP be the set of drawn symbolic pictures on a set Σ_ϕ and $\bullet_f$ be the concatenation operator w.r.t f such that (Σ_ϕ, f) is a monoid. (DSP, $\bullet_f$) is an inverse monoid if the merging function f is such that for each a$\in \Sigma$ there exists a unique b$\in \Sigma$ with f(a, b)=ϕ and f is not idempotent.*

Proof. *From proposition 2.1, (DSP, $\bullet_f$) is a monoid since (Σ_ϕ, f) is a monoid. In order to prove that (DSP, $\bullet_f$) is inverse we show that for each q$\in$(DSP, $\bullet_f$) there exists a unique element, denoted q^{-1}, such that $q \bullet_f q^{-1} \bullet_f q = q$ and $q^{-1} \bullet_f q \bullet_f q^{-1} = q^{-1}$. Let q=<<p, s, e>, Σ, δ_1 > be a nonempty drawn symbolic picture, and let q^{-1}=<<p, e, s>, Σ, δ_2 > such that for each v$\in$W(<p, s, e>) f(δ_1(v),δ_2(v))=ϕ. It is worth noting that the hypothesis on* *merging function f ensure that for each q there exists a unique drawn symbolic picture* q^{-1}*. Now it is easy to verify that* $q \bullet_f q^{-1} \bullet_f q = q$ *and* $q^{-1} \bullet_f q \bullet_f q^{-1} = q^{-1}$*.*

From propositions 2.2 and 2.3, we have the following theorem.

Theorem 2.3 *Let DSP be the set of drawn symbolic pictures on a set* Σ_ϕ *and* $\bullet_f$ *be the concatenation operator w.r.t f such that (*Σ_ϕ*, f) is a monoid. (DSP,* $\bullet_f$*) is a finitely generated inverse monoid if the merging function f is such that for each* $a \in \Sigma$ *there exists a unique* $b \in \Sigma$ *with f(a, b)=*ϕ *and f is not idempotent.*

2.4 Decidability Problems for Picture Languages

In the last decades, drawn picture languages have been deeply analyzed and the intractability of several important decision problems has been proved. In particular in [66] it has been shown that the membership problem is NP-complete and that the emptiness-of-intersection problem for regular picture languages is undecidable. Moreover, other decision problems such as the *equivalence problem*, the *containment problem* and the ambiguity problem are proved to be undecidable also for regular drawn picture languages [42, 44]. These results have motivated the investigation of subclasses of drawn pictures, with nice properties from the decidability and complexity point of view [66, 42, 46]. It is worth noting that the problems, which were intractable for drawn picture languages, turn out to be obviously intractable also for drawn symbolic picture languages. On the other hand, the nice properties (from the decidability and complexity point of view) determined for restricted subclasses of drawn picture languages not always can be inherited by the analogous subclasses of drawn symbolic picture languages. So, it is interesting to determine the conditions, in terms of properties of both string descriptions and merging functions, ensuring the preservation of such properties. Following this line, in chapter 3 we will analyze the *picture membership problem* for several subclasses of drawn symbolic picture languages.

In the rest of this chapter we analyze the picture ambiguity and the *f-independency* problems.

A picture description language L is unambiguous if for every picture p in $dpic$(L) there is a unique Π-word w in L such that $p = dpic(w)$. Otherwise, L is ambiguous. In section 2.4.1 the ambiguity problem for drawn symbolic picture languages is addressed.

In section 2.4.2 we will focus our attention on the notion of *f-independency* for $\Sigma \cup \Pi$-words and in particular we analyze an interesting subclass of $\Sigma \cup \Pi$-grammars, named *f-independent*. A $\Sigma \cup \Pi$-word w is *f-independent* if it describes for each point of the corresponding picture at most one occurrence of a symbol in Σ. Otherwise we say that w presents a conflict or w is not *f-independent*. Then in section 2.5 we will prove that it is decidable to establish whether or not a given context-free $\Sigma \cup \Pi$-grammar is *f-independent*.

2.4.1 The Ambiguity Problem

The formal language theory largely investigates the grammar ambiguity problem. In the following we will focus our attention on this aspect and provide some decidability results.

A string grammar G is *ambiguous* if there exists a sentence in $L(G)$ for which G produces two different derivation trees. In the case of the drawn symbolic grammars, this definition implies two distinct cases: a grammar $\mathbf{G}$ is ambiguous if there exists a picture p in L($\mathbf{G}$) such that, either there are two string descriptions in $\mathbf{G}$ for p, or the only string description of p in $\mathbf{G}$ may be derived in two different ways. In both cases, p would have two different syntactic interpretations. In the former we have *visual ambiguity*, in the latter we have *structural ambiguity*. In this section we will focus our attention on visual ambiguity.

The general visual ambiguity for picture languages was addressed in [41]. There, it has been proved that the problem of determining whether or not a language contains two distinct words that describe the same picture is undecidable for regular languages and for linear languages which describe three-way stripe picture languages. Picture ambiguity was also previously treated in [52]. There, the authors show that the problem of determining whether or not an arbitrary picture p can be drawn by two distinct words in a language L is undecidable for context-sensitive Π-languages and is decidable for context-free Π-languages. In this section, we will show that this problem, extended to the drawn symbolic languages languages, is decidable for context-free grammar $\mathbf{G}$.

In a drawn symbolic picture grammar we distinguish two reasons for which $dspic_f()$ produces the same drawn symbolic picture on different sentences:

1. the sentences describe different walks on the drawn symbolic picture;

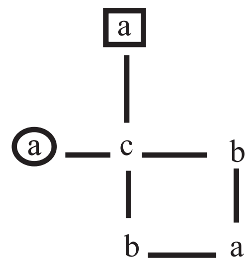

Figure 2.11: *An ambiguous picture under the drawn symbolic grammar of Example 2.6*

2. the sentences describe the same walk on the drawn symbolic picture, and the walk intersects itself in a point. Due to the definition of $dspic_f()$, the sentences may have different symbols associated to that point yet producing the same picture.

In the first case, we talk about *visual ambiguity on draw*, in the second case, we talk about *visual ambiguity on intersection*. It is intuitive that an *f-independent* drawn symbolic picture grammar cannot exhibit visual ambiguity on intersection.

Example 2.6 *Let* $\mathbf{G}=< G_{\Sigma\cup\Pi},\ dspic_f()>$ *be a the drawn symbolic picture grammar, where* $G_{\Sigma\cup\Pi} = <\ \{r,l,u,d\},\ \{a,b,c,k\},\ \{S,A,B,C\},\ P,S\ >$, *f is such that f(c,c)=c, f(c,k)=c, and P is the set of productions* $\{S \rightarrow a\mathbf{r}cAaB,\ A \rightarrow \mathbf{r}b\mathbf{d},\ A \rightarrow \mathbf{d}b\mathbf{r},\ B \rightarrow \mathbf{l}b\mathbf{u}C,\ B \rightarrow \mathbf{u}b\mathbf{l}C,\ C \rightarrow c\mathbf{u}a,\ C \rightarrow k\mathbf{u}a\}$. *This grammar is both visually ambiguous on draw, and visually ambiguous on intersection. It is visually ambiguous on draw since there exist two different sentences* $w = \mathbf{r}c\mathbf{r}b\mathbf{d}a\mathbf{l}b\mathbf{u}c\mathbf{u}a$ *and* $w_1 = \mathbf{r}c\mathbf{d}b\mathbf{r}a\mathbf{u}b\mathbf{l}c\mathbf{u}a$ *in* $L(G_{\Sigma\cup\Pi})$, *that are the description of different walks on the same drawn symbolic picture. It is visually ambiguous on intersection since there exist two different sentences* w *and* $w_2 = \mathbf{r}c\mathbf{r}b\mathbf{d}a\mathbf{l}b\mathbf{u}k\mathbf{u}a$ *in* $L(G_{\Sigma\cup\Pi})$, *that describe a walk on the drawn symbolic picture intersecting itself in a point in which they have different symbols associated. Fig. 2.11 shows the ambiguous drawn symbolic picture described by* $\mathbf{G}$ *which is the result of applying the function* $dspic_f()$ *on any of* w, w_1 *and* w_2.

The decidability of the visual ambiguity problem for a drawn symbolic picture derives from the following decidability properties which are straightforward consequences of the regularity of the set of *f-independent* description language of a drawn symbolic picture (see Theorem 2.4) and from the closure property of context-free languages with regular sets.

Corollary 2.1 *Let* G *be a* $\Sigma \cup \Pi$*-grammar, and dsp a drawn symbolic picture. The following problems are decidable:*

1. $L(\mathrm{G}) \cap dspdes(dsp) \neq \emptyset$
   i.e., $L(\mathrm{G})$ *contains a description for the drawn symbolic picture dsp;*

2. $|L(\mathrm{G}) \cap dspdes(dsp)| < \infty$
   i.e., $L(\mathrm{G})$ *contains a finite number of descriptions for the drawn symbolic picture dsp;*

3. $|L(\mathrm{G}) \cap dspdes(dsp)| = 1$
   i.e., $L(\mathrm{G})$ *contains only one description for the drawn symbolic picture dsp.*

As an immediate consequence of this Corollary, given a drawn symbolic picture *dsp* and an arbitrary $\Sigma \cup \Pi$-grammar G it is possible to decide if *dsp* is described by only one sentence of G, i.e., if the description of *dsp* in G is visually ambiguous.

2.4.2 The *f*-independency Problem

In this section we characterize an interesting subclass of drawn symbolic picture languages, named *f-independent*.

Let us observe that in general a $\Sigma \cup \Pi$-word w can describe different pictures depending on the applied merging function, as shown in Fig. 2.5. We say that a word w is *f-independent* whenever it always describes the same picture, independently from the applied merging function *f*. More formally, we give the following definition.

Definition 2.7 *Let* Φ *be the family of merging functions on* Σ_ϕ*. Given a* $\Sigma \cup \Pi$*-word w, w is f-independent whenever the following condition holds:*

$$dspic_f(w) = dspic_g(w),\ \forall f,\ g \in \Phi.$$

A $\Sigma \cup \Pi$*-language* L *is f-independent if* $\forall w \in L$*, w is f-independent.*

Taking into account the definition of merging functions, it is easy to verify the following characterization which provides a condition ensuring the *f-indenpendency* of $\Sigma \cup \Pi$-words.

Property 2.1 *A $\Sigma \cup \Pi$-word w is f-independent if and only if one of the following conditions holds:*

- $w = \epsilon$;
- $w = w'\tau$ *for some f-independent* $w' \in (\Sigma \cup \Pi)^*$, $dspic_f(w') = <sc, \Sigma, \delta>$, $sc = <p, s, e>$, *and* ($\tau \in \Sigma$ *and* $\delta(e)=\phi$).

In other words, w is *f-independent* if and only if it specifies at most one symbol in Σ for each point of the described picture. Otherwise we say that w presents a "conflict" or w is *f-dependent*. As an example, the $\Sigma \cup \Pi$-word $w = a\mathbf{r}b\mathbf{r}c\mathbf{l}$ is *f-independent*. On the contrary, the $\Sigma \cup \Pi$-words $w' = a\mathbf{r}b\mathbf{r}c\mathbf{l}e$ and $w'' = \mathbf{r}aa\mathbf{r}c\mathbf{l}e$ are *f-dependent* because two symbols (b and e in w', and two occurrences of a in w'') are specified for the same position.

The set of *f-independent* words which describe a drawn symbolic picture p, is defined as:

$Cdspdes(p) = \{w \in (\Sigma \cup \Pi)^* \mid dspic_f(w) = p, \forall f$ in the family of the merging functions $\}$

It can be easily proved the regularity of the set $Cdspdes(p)$.

Theorem 2.4 *Let p be a drawn symbolic picture. Then, the set Cdspdes(p) is a regular $\Sigma \cup \Pi$-language.*

The concept of *f-independency* can be extended to drawn symbolic picture grammars in the natural way.

Definition 2.8 *A $\Sigma \cup \Pi$-grammar $G_{\Sigma \cup \Pi}$ is f-independent if any $\Sigma \cup \Pi$-word $w \in L(G_{\Sigma \cup \Pi})$ is f-independent. A drawn symbolic picture grammar* $\mathbf{G} = <G_{\Sigma \cup \Pi}, dspic_f()>$ *is f-independent if $G_{\Sigma \cup \Pi}$ is f-independent.*

It is worth noting that the notion of *f-independent* description is strongly related to the concept of self-avoiding introduced in [59]. In particular, a picture word is self-avoiding if all its nonempty factors are not loop. A word represents a loop if its shift is (0,0) (start point=end point). In other words we can say that a picture description is self-avoiding if all its nonempty subpictures have shift different from (0,0). Let us consider the *f-dependent* $\Sigma \cup \Pi$-word $w = a\mathbf{r}b\mathbf{r}c\mathbf{l}e$ in which the two symbols b and e are described for the position (1,0). The Π-word $w' = \mathbf{rrl}$, obtained from w not considering the symbols, is not self-avoiding. So, we can say that a drawn symbolic picture is self-avoiding if it is *f-independent* with the addition that each position can be traversed at most one time.

The decidability of the *f-independency* of a drawn symbolic picture grammar is an interesting problem. The problem has been partially addressed in [16]. In particular, it can be shown that it is always possible to decide whether or not a Σ∪Π-grammar generates only *f-independent* descriptions for a drawn symbolic picture. The proof of such results exploits the following decidability properties which are straightforward consequences of the regularity of the set of *f-independent* description language of a drawn symbolic picture:

Corollary 2.2 *Let* G *be a* $\Sigma \cup \Pi$*-grammar, and dsp a drawn symbolic picture. The following problems are decidable:*

1. $L(\mathrm{G}) \cap Cdspdes(dsp) \neq \emptyset$
   i.e., L(G) *contains an f-independent description for the drawn symbolic picture dsp;*

2. $|L(\mathrm{G}) \cap Cdspdes(dsp)| < \infty$
   i.e., L(G) *contains a finite number of f-independent descriptions for the drawn symbolic picture dsp;*

3. $|L(\mathrm{G}) \cap Cdspdes(dsp)| = 1$
   i.e., L(G) *contains only one f-independent description for the drawn symbolic picture dsp.*

Proof. *From theorem 2.4 and from the closure property of context-free languages with regular sets we have that the set L*(G)*∩Cdspdes(dsp) is a context-free language. Thus, the decidability results follow from the decidability of the emptiness, the finiteness, and the singleton problems for context-free string languages [35].*

As a consequence of regularity of the set of drawn symbolic picture descriptions for a given *dsp* (see section 2.3) and of Corollary 2.2, we have the following result which states that the problem of determining whether or not a drawn symbolic picture has only *f-independent* descriptions in a language is decidable for context-free languages.

Theorem 2.5 *Let* G *be a* $\Sigma \cup \Pi$*-grammar and dsp be a drawn symbolic picture. It is decidable whether L*(G) *contains only f-independent descriptions for dsp.*

Proof. *By Corollary 2.2 we have that it is decidable whether L(G) contains an f-independent description for dsp. Moreover, we show that it is decidable whether L(G) contains an f-dependent description for dsp. To this aim, we notice that the f-dependent description language of dsp, defined as:*

$$\text{NCdspdes}(dsp) = \text{dspdes}(dsp) - \text{Cdspdes}(dsp)$$

is a regular $\Sigma \cup \Pi$-language (by the closure properties of regular sets). Thus, by the closure properties of context-free languages, the set

$$L(\text{G}) \cap \text{NCdspdes}(dsp)$$

is a context-free language. Then, the claim follows from the decidability of the emptiness problem of context-free languages.

Starting from this partially result, in the next section we will prove that it is decidable to establish whether or not a given context-free $\Sigma \cup \Pi$-grammar is *f-independent.*

2.5 The Decidability of the *f*-independency for Context-free Description Grammars

In this section we prove the decidability of the *f-independency* problem for context-free $\Sigma \cup \Pi$-grammars, and to this aim we introduce the following notions.

Let G = $<\Gamma$, Δ, N, P, S$>$ be a $\Gamma \cup \Delta$-grammar and A$\subseteq \Gamma \cup \Delta$ a subset of the terminal symbols. We define the string grammar G_A= $<$A, N, P_A, S$>$ as the projection of the grammar G on the set A, where P_A is the set of productions obtained from the productions in P by removing all the terminals not in A. Moreover L_A denotes the language generated by the grammar G_A, i.e. the set of words obtained by eliminating from the words of L(G) the terminals symbols not in A. As an example if L=$\{a\mathbf{r}b\mathbf{d}g$, $c\mathbf{luu}g\mathbf{dd}\}$ and $A = \Pi$ then L_A=$\{$**rd**, **luudd**$\}$.

Given Σ=$\{a_i \mid 1 \leq i \leq n\}$, if L is a language and if with each word w in L we associate the n-tuple whose ith coordinate, $1 \leq i \leq n$, is the number of occurrences of a_i in w, then the resulting set of n-tuples is semilinear. In particular, given the function Ψ: $\Sigma^* \rightarrow N^n$ defined by Ψ(z)=(a_1(z),...,a_n(z)) where a_i(z) denotes the number of occurrence of the symbol a_i in z, it has been proved that Ψ(M) is semilinear for each language M (see Th. 5.2.1 in [28]).

Moreover, we introduce the notion of shift of a $\Sigma \cup \Pi$-word, in analogy to the concept of shift for a drawn picture as defined in [52].

Definition 2.9 *Let w be a $\Sigma \cup \Pi$-word and $dspic_f(w)$ =<<p, s, e>, Σ, δ > be the corresponding drawn symbolic picture. The shift of w is defined by shift(w)=(x(e)-x(s),y(e)-y(s)), where for a point v=(m, n), x(v)=m and y(v)=n.*

As an example, given the $\Sigma \cup \Pi$-word w=$a\mathbf{r}b\mathbf{r}c\mathbf{l}a$, the drawn symbolic picture resulting from $dspic_f(w)$ has start point (0,0) and end point (1,0), so *shift*(w)=(1,0). On the contrary, the $\Sigma \cup \Pi$-word "$a\mathbf{r}b\mathbf{r}c\mathbf{l}a\mathbf{l}c$" has shift (0,0). In other words, a $\Sigma \cup \Pi$-word w has shift (0,0) if $\#_{\mathbf{r}}(w) = \#_{\mathbf{l}}$(w), $\#_{\mathbf{u}}(w) = \#_{\mathbf{d}}(w)$.

Theorem 2.6 *Given a context-free $\Sigma \cup \Pi$-grammar G, it is decidable whether or not G is f-independent.*

Proof. *G = < $G_{\Sigma\cup\Pi}$, $dspic_f$()> is f-independent if any $\Sigma \cup \Pi$-word $w \in L(G_{\Sigma\cup\Pi})$ is f-independent. Let L^1=suff(pref($L(G_{\Sigma\cup\Pi})$)). Let us consider the language: L^2=\{$w \in L^1$ | $w=ax'b$ with $a,b \in \Sigma$ and $x' \in (\Sigma \cup \Pi)^*$\}. If L^2 is context-free then M=L^2_Π is context-free. Let P_1 be the vectors of Parikh of the words in L^2_Π, i.e. P_1=$\Psi(L^2_\Pi)$. Let K be the set of all $\Sigma \cup \Pi$-words w such that shift(w)=(0,0) (i.e., $\#_{\boldsymbol{r}}(w)$=$\#_{\boldsymbol{l}}(w)$, $\#_{\boldsymbol{u}}(w)$= $\#_{\boldsymbol{d}}(w)$) and P_2 be the set of Parikh vectors of K, i.e. P_2= Ψ(K). Then $L(G_{\Sigma\cup\Pi})$ is f-independent if and only if $P_1 \cap P_2$=$\emptyset$. Since Ψ(X) is semilinear for each language X and it is well known that the intersection problem is decidable (see Th.5.2.1 and Th.5.6.1 in [28]) then it is decidable whether or not $P_1 \cap P_2$=$\emptyset$. Hence the theorem holds.*

As described at the beginning of this chapter the notion of *f-independency* for $\Sigma \cup \Pi$-words is strongly related to the concept of self-avoiding for Π-words. Indeed, a picture description is self-avoiding if all its nonempty subpictures have shift different from (0,0). Thus, given a context-free $\Sigma \cup \Pi$-grammar G that generates a string description language L(G), it is decidable whether or not L(G) contains a non self-avoiding string description. Moreover, as an immediate consequence of the Theorem 2.6, we can state the following Corollary.

Corollary 2.3 *Given a context-free Π-grammar G that generates a string description language L(G), it is decidable whether or not L(G) contains a non self-avoiding string description.*

The problem to decide whether or not a Π-language L contains a self-avoiding Π-word is more difficult. As a matter of fact Robilliard and Simplot have proved that, given a rational language L over Π, it is undecidable to know whether or not L contains a self-avoiding word.

In [10] an interesting approach to prove the decidability of *f-independency* problem for regular Σ∪Π-grammar has been developed. In particular the proposed algorithm works on the automaton recognizing the language generated by the Σ∪Π-grammar. Such approach is described in the next subsection.

2.5.1 An Algorithm for Regular Description Grammars

In this section, we will show that it is decidable whether or not a given regular $\Sigma \cup \Pi$-grammar G generates only *f-independent* drawn symbolic picture descriptions by using an algorithm that works on the automaton recognizing L(G).

Let us observe that w=$a\mathbf{r}b\mathbf{r}c\mathbf{l}a$ is a *f-dependent* description. Indeed the application of $dspic_f(w)$ to w produces a conflict between symbols b and a. Such a conflict can also be detected by considering the $\Sigma \cup \Pi$-subsentence w_1=$b\mathbf{r}c\mathbf{l}a$ (which starts from symbol b and ends to symbol a) and by observing that shift(w_1)=(0,0). So, the *f-independency* problem for a $\Sigma \cup \Pi$-grammar G can be reduced to verify whether or not G generates a (sub)sentence w=$a_i \alpha a_j$ with $\alpha \in (\Pi \cup \Sigma)^*, a_i, a_j \in \Sigma$ such that $shift(w)$=(0,0).
Let us observe that given a regular Σ∪Π-grammar G=(Σ∪Π, N, P, S) it is always possible to find a deterministic finite automaton, named $\Sigma \cup \Pi$-automaton, that recognizes L(G). A $\Sigma \cup \Pi$-automaton is a 5-tuple M=(Q, $(\Pi \cup \{\epsilon\})\Sigma^*$,$\delta$, p_0, F) where 1) Q is a finite set of states of the finite control, 2) $(\Pi \cup \{\epsilon\})\Sigma^* = \{\pi\tau \mid \pi \in \Pi \cup \{\epsilon\}$ and $\tau \in \Sigma^*\}$, 3) δ is a transition function that maps the elements of the set Q×($(\Pi \cup \{\epsilon\})\Sigma^*$) into subsets of Q, 4) p_0 ∈Q is the initial state of the finite control, 5) F⊆Q is the set of final states. Without loss of generality, we can suppose that such an automaton has the following properties:
a) the initial state has no input arc,
b) arcs leaving from the initial state are labelled only with σ belonging to Σ^*,
c) any other arc is labelled with $\pi\tau$, where $\tau \in \Sigma^*$ and $\pi \in \Pi$,
d) for any state q_i the string belonging to Σ^* on the input arcs of q_i are equal and we denote by $InSymb(q_i)$ such sequence od symbols.

An example of $\Sigma \cup \Pi$-automaton which exhibits the above properties is depicted in Fig.2.12.

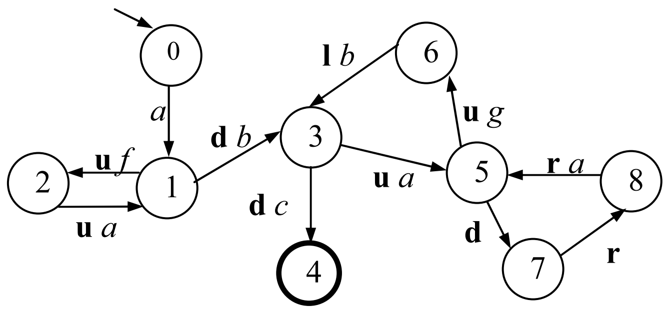

Figure 2.12: $\Sigma \cup \Pi$-*automaton M*

Thus, the *f-independency* problem for a regular $\Sigma \cup \Pi$-grammar is equivalent to verify:
1. if there exists an edge label τ that contains at least two occurrences of symbols in Σ; or
2. if there exists a path in the automaton described by a string (w)=$a_i \alpha a_j$ with $\alpha \in (\Pi \cup \Sigma)^*$, $a_i, a_j \in \Sigma$ such that *shift*(w)=(0,0).

In order to detect the *f-dependent* descriptions, let us introduce the following notions. A *simple path* p is a sequence of states $(q_1, ..., q_m)$ with m>0, $q_i \neq q_j$ for 1≤i,j≤m and i≠j. A path p=$(q_1, ..., q_m)$ forms a cycle if q_1=q_m and in p there are at least two different states. We say that p is *incident* on q_1. A cycle is *simple* if, in addition, $q_2 q_3, ..., q_m$ are distinct. Otherwise the cycle is named *nested*. With regards to the $\Sigma \cup \Pi$-automaton M depicted in Fig. 2.12 the simple path p_1=(1,2) exhibits the simple cycle (1,2,1) incident on state 1. Any nested cycle on p can be decomposed in a *main cycle* that is a simple cycle $q_i, p', q_i)$ on p, and in a set of cycles on p'. As an example the simple path p_2=(1,3,4) in Fig.2.12, exhibits the simple cycle (1,2,1) incident on state 1, and a nested cycle incident on state 3. Such nested cycle can be decomposed in the main cycle (3,5,6,3) and in the simple cycle (5,7,8,5). Any simple path p in M identifies a language, denoted by *lan(p)*, whose sentences are obtained by traversing once all the states of p and zero or more times each of the cycles on p. As an example, let us consider the simple paths p_1=(1,2) and p_2=(1,3,4) in M of Fig.2.12. The correspondent languages *lan*(p_1) and *lan*(p_2) can be described by the expressions $a(\mathbf{u}f\mathbf{u}a)^*\mathbf{u}f$ and $a(\mathbf{u}f\mathbf{u}a)^*\mathbf{d}b(\mathbf{u}a(\mathbf{drr}a)^*\mathbf{u}g\mathbf{l}b)^*$ $\mathbf{d}c$, respectively. It is obvious that the shift of any sentence in *lan(p)* is given by the sum of the shifts determined by the sentence on the simple path and the shifts determined by the sentences on the cycles multiplied for the number of times any cycle has been visited.

As an example, let us consider the sentence in $lan(p_1)$ obtained by traversing the cycle (1,2,1) two times. The shift of such a sentence is

$$\text{shift}(a(\mathbf{u}f\mathbf{u}a)^2\mathbf{u}f)=\text{ shift}(a\mathbf{u}f) + \text{shift}(a\mathbf{u}f\mathbf{u}a)*2 = (0,1)+(0,2)*2=(0,5)$$

where $a\mathbf{r}c$ is identified by p_1 and $a\mathbf{u}f\mathbf{u}a$ is identified by the simple cycle on node 1.

It is possible to represent the shifts of all the sentences contained in $lan(p)$ by using a couple formed by a shift determined by the simple path p and by a list (possibly nested) of shifts determined by the cycles on p. More formally.

Definition 2.10 *Let p be either a simple path or a simple cycle with $n \geq 0$ cycles, and let denote by mc_k the main cycle of the k-th cycle. Let w_p be the $\Sigma \cup \Pi$-word identified by p. The shift of lan(p) is defined as:*

$$lansh(p) = \begin{cases} (shift(w_p), []) & \textit{if } n = 0; \\ (shift(w_p), [lansh(mc_1), ..., lansh(mc_n)]) & \textit{if } n > 0 \end{cases}$$

In the sequel we denote the couple ((x,y),[]) with <x,y>. As an example, the shift of the previous languages $lan(p_1)$) and $lan(p_2)$ are $lansh(p_1)$=$lansh$((1,2))=shift($a\mathbf{u}f$), [$lansh$(1,2,1)])= ((0,1), [(shift($a\mathbf{u}f\mathbf{u}a$), [ ])])= ((0,1), [<0,2>])and $lansh(p_2)$)=((0,-2), [<0,2>,((-1,2), [<2,-1>])]), respectively.

Given $lansh(p)$, we can verify whether or not $lan(p)$ contains a sentence with shift (0,0). For sake of clarity, we will first consider the case where each cycle is without subcycles. The general case will be treated in the forth.

Let $lansh(p)$=((x,y),[<x_1,y_1 >,...,<x_n,y_n >]) then the shift of a sentence in $lan(p)$ is (x,y) + (x_1,y_1)h_1 +...+ (x_n,y_n)h_n where each h_i is a natural number which indicates how many times the i-th cycle is traversed. It is worth noting that $lan(p)$ contains a sentence with shift (0,0), if and only if there exist h_1,...,h_n $\in N$ such that:

$$\begin{cases} x + x_1h_1 + x_2h_2 + ... + x_nh_n = 0 \\ y + y_1h_1 + y_2h_2 + ... + y_nh_n = 0 \end{cases}$$

Let us observe that the above system is a linear Diophantine equations system because a different variable is associated to each shift $< x_i, y_i >$. In [56] it is shown that it is always possible to decide whether or not such systems have a nonnegative integer solution, thus it is decidable to establish if $lan(p)$ contains a sentence with shift (0,0) when p has no nested cycles.

As an example, let us consider the previous simple path p_1 with $lansh(p_1)$=((0,1), [<0,2>]). It describes a sentence with shift (0,0) if and only if the following linear equations system has a nonnegative integer solution:

$$\begin{cases} 0 = 0 \\ 1 + 2h_1 = 0 \end{cases}$$

It can be easily seen that the system has no such solution.

Let us now consider the case where p has nested cycles and verify whether $lan(p)$ contains a sentence with shift (0,0). In particular, given the simple path p_2 of the previous example, from $lansh(p_2)$=((0,-2),[<0,2>,((-1,2), [<2,-1>])]) we construct the following linear equation systems.

$$\begin{cases} h_1 * 0 = 0 \\ -2 + 2h_1 = 0 \end{cases}$$

$$\begin{cases} h_1 * 0 + (-1)h_2 + 2h_3 = 0 \\ -2 + 2h_1 + 2h_2 + (-1)h_3 = 0 \\ h_2 \geq 1 \end{cases}$$

where h_1, h_2 and h_3 are respectively the numbers of times that the simple cycles (1,2,1), (3,5,6,3) and (5,7,8,5) are traversed. The first system is obtained considering h_2=0 and h_3=0 because if the main cycle c=(3,5,6,3), is not traversed at least once then its subcycle (5,7,8,5) cannot be traversed either.

The second system considers the sentences in $lan(p_2)$ obtained by traversing c in M at least once. For this reason the variable h_2 associated to c must be greater than zero. Such condition is not considered in a linear Diophantine equations system. Nevertheless we can obtain an equivalent linear Diophantine equation system by setting h_2=(h_2'+1) as shown in the following:

$$\begin{cases} -(h_2' + 1) + 2h_3 = 0 \\ -2 + 2h_1 + 2(h_2' + 1) - h_3 = 0 \end{cases}$$

It can be easily seen that h_1=1 is a solution for the first system. So, $lan(p_2)$ contains at least a sentence w such that $shift(w)$=(0,0).

In general, let us observe that given a nested cycle c_i whose shift is (s_i, [s_{i_1},s_{i_2},...,s_{i_m}]) with m>0, if the main cycle of c_i, whose shift is s_i, is not traversed at least once then any subcycle with shift s_{i_k} cannot be traversed either. Thus, the value of the variable associated to s_{i_k} can be greater than zero only if the value of the variable associated to s_i is greater than zero. Such observation can be iterated to each nested subcycle of c_i. In order to verify if a sentence contained in $lan(p)$ has shift (0,0), we must resolve a set of linear Diophantine equation systems obtained from $lansh(p)$ by considering the combinations of the shifts of the cycles on p.

Finally, let G be a regular Σ∪Π-grammar G, and M a Σ∪Π-automaton that recognizes L(G), G is *f-dependent*, if one of these conditions hold:

1. there exists an edge label τ in M such that τ contains at least two occurrences of symbols in Σ;
2. there exists a simple path p=(q_0,q_i,...,q_j), in M with $InSymb(q_i) \neq \epsilon$,$InSymb(q_j) \neq \epsilon$, such that *lansh(p)* contains a sentence with shift (0,0).

This strategy can be summarized by the algorithm described in Fig.2.13.

ALGORITHM 3.1. *f-independency* test for regular symbolic picture description languages.

INPUT: a regular Σ∪Π-grammar G;

OUTPUT: true if G is *f-independent*; false otherwise.

1. **Let** M=(Q, (Π∪{ε})Σ^*, δ, q_0, F) be the Σ∪Π-automaton that recognizes the sentences in L(G) and satisfying the properties a),b),c),d);
2. **for** each edge label τ in M
3.   **if** τ contains two symbols in Σ **then return** false
4. **for** each simple path p=(q_0,q_i,…,q_j) in M
5.   such that InSymb(q_i), InSymb(q_j) ≠ε **do**
6.   construct *lansh*(*p*);
7.   determine the set of linear Diophantine equations systems corresponding to *lansh*(*p*);
8.   **if** at least one system has a nonnegative integer solution **then return** false;
9. **return** true.

Figure 2.13: *The algorithm for the f-independency.*

Thus we can state the following theorem.

Theorem 2.7 *Given a regular $\Sigma \cup \Pi$-grammar G, it is decidable whether or not G is f-independent.*

Proof. *Let M be the $\Sigma \cup \Pi$-automaton that recognizes the sentences in $L(G)$. Let us observe that M has a finite number of different simple paths. Moreover for any simple path p the number of linear Diophantine equations systems is bounded by the number $2^{(ns+1)(nl)}$ where nl is the maximum nesting level of a cycle in M and ns is the maximum number of cycles on a simple path in M. Thus the theorem follows from the decidability of the existence of nonnegative integer solutions for linear Diophantine equations systems.*

Chapter 3

The Membership Problem for Drawn Symbolic Pictures

In this chapter we analyze the membership problem for some subclasses of drawn symbolic picture languages. The picture membership problem for a $\Sigma \cup \Pi$-language L asks whether or not a given symbolic picture q can be described by a $\Sigma \cup \Pi$-word in L. It can be easily shown that drawn symbolic picture languages inherit from drawn picture languages the intractability of the membership problem. Indeed the following theorem can be proved as an extension of the analogous result for drawn picture languages [66].

Theorem 3.1 *The membership problem for regular drawn symbolic picture languages is NP-complete.*

To more efficiently solve the membership problem, in the last years several subclasses of drawn picture languages have been introduced. Indeed it has been proved that the membership problem for regular drawn stripe picture languages can be decided deterministically in linear time [66] and the membership problem for context-free three-way drawn picture languages and *k-retreat-bounded* drawn picture languages can be decided deterministically in polynomial time [42, 43]. In this chapter, we address the issue to identify conditions which allow us to preserve such nice complexity results in the case of our picture model. To this aim we will provide some characterizations involving properties of both string descriptions and merging functions. As we will show one of such properties is closed to the condition ensuring the monoid characterization of the set of drawn symbolic pictures.

3.1 Stripe Picture Languages

A drawn symbolic stripe picture language is a drawn symbolic picture language whose pictures fit into a stripe defined by two parallel lines. In particular, let $y = kx + d_1$, and $y = kx + d_2$ be the equations of two parallel lines, where k, d_1, d_2, are real numbers such that $d_1 < d_2$. The (k, d_1, d_2)-stripe consists of the integer grid points which are between these lines and is defined by

$\mathrm{M}_0^{(k,d_1,d_2)} = \{(\mathrm{i},\mathrm{j}) \in M_0 \mid k\mathrm{i} + d_1 \leq \mathrm{j} \leq k\mathrm{i} + d_2\}$.

The special case of the vertical stripe (i.e., $\mathrm{k} = \infty$) can be defined by

$\mathrm{M}_0^{(\infty,d_1,d_2)} = \{(\mathrm{i},\mathrm{j}) \in M_0 \mid d_1 \leq \mathrm{j} \leq d_2\}$.

In the sequel we only consider non vertical stripes. Nevertheless, the provided results can be easily extended to the case of vertical stripes. Now, we are ready to give the formal definition of drawn symbolic stripe picture language:

Definition 3.1 *A drawn symbolic picture $q = << p, s, e >, \Sigma, \delta >$ is a drawn symbolic* (k, d_1, d_2)*-stripe picture if W(<p,s,e>)* $\subseteq M_0^{(k,\mathrm{d}_1,\mathrm{d}_2)}$. *A drawn symbolic picture language L is a drawn symbolic* (k, d_1, d_2)*-stripe picture language if every drawn symbolic picture in L is a drawn symbolic* (k, d_1, d_2)*-stripe picture.*

Example 3.1 *Let $q = <<p, s, e>, \Sigma, \delta >$ be a drawn symbolic picture, where p=<{{(0,0), (0,-1)}, {(0,0), (1,0)}, {(1,0), (1,1)}, {(1,0), (2,0)}, {(2,0), (2,1)}}, (0,0), (2,1)>,* $\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$, *and δ is the function such that $\delta((0,-1)) = \phi$, $\delta((0,0))=\mathrm{a}$, $\delta((1,0))=c$, $\delta((1,1))=b$, $\delta((2,0))=\phi$, $\delta((2,1))=\mathrm{c}$. It can be easily verified that q is a drawn symbolic (1, -2, 0)-stripe picture (see Fig.3.1).*

The drawn symbolic picture language describing the repetitive structure of a rubber molecule given in example 2.3 is a drawn symbolic (0, -1, 2)-stripe picture language.

The next thorem states that we can decide whether a drawn symbolic picture grammar generates a drawn symbolic stripe picture language or not.

Theorem 3.2 *Let **G** be a context-free drawn symbolic picture grammar. It is decidable whether or not L(**G**) is a drawn symbolic stripe picture language or not.*

Proof. *Let $G = <G_{\Sigma\cup\Pi}$, $dspic_f()$> and G_Π be the projection of the grammar G in the set* Π. *G generates a drawn symbolic stripe picture language if and only if G_Π generates*

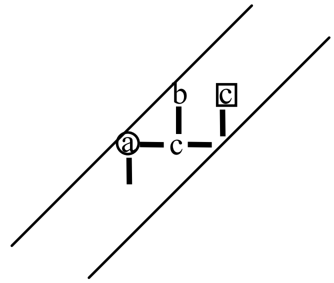

Figure 3.1: *A drawn symbolic (1,-2,0)-stripe picture*

a drawn picture language. Since G_Π is a context-free drawn picture grammar, the thesis follows by applying the Theorem 5.3 in [66].

Let us consider a (k, d_1, d_2)-stripe with $k = n/m$, such that n and m have no common divisor and $m > 0$ (the case with k=0, can be similarly analyzed), the (m, n, d_1, d_2)-unit point field, denoted by F_0, and the (m, n, d_1, d_2)-unit labeled point field, denoted by LF_0, are defined as follows:

F_0 ={(i,j) $\in M_0^{(k,d_1,d_2)}$ | $0 \leq$ i $<$ m}

LF_0 = {[a, (i,j)]| $a \in \Sigma$, (i,j)$\in F_0$}.

It is worth noting that $M_0^{(k,d_1,d_2)} = \bigcup_{i \in Z} t_{im,in}(F_0)$[1]. Thus, any point in the stripe can be mapped in a corresponding point in the set F_0, and the set F_0 can be used to represent any position of a drawn symbolic (k, d_1, d_2)-stripe picture. As an example, let us consider the (3/2, -3, 2)-stripe shown in Fig. 3.1. The shared region represents the (2, 3, -3, 2)-unit point field F_0. Given the point (3/2,1) belonging to F_0, we have that (7/2,4) = $t_{2,3}$((3/2,1)), and (-1/2, -2) = $t_{-2,-3}$((3/2,1)).

Now, we prove that the membership problem for some subclasses of regular drawn symbolic stripe picture languages can be decided deterministically in linear time. To this aim, we establish a correspondence between regular string languages and the previous regular languages. In order to prove such a correspondence, a stripe is divided into vertical stripes of equal width (named sliced portions) so that any picture is divided into a sequence of subpictures. The alphabet of the string language consists of the set of such possible different subpictures. Since in any stripe picture language only a finite number of different

[1]Let us remember that $t_{m,n}$=(i+m,j+n), where v=(i,j), and $t_{m,n}$(A)={{$t_{m,n}$(v),$t_{m,n}(v')$} | {v,v'}$\in$A}

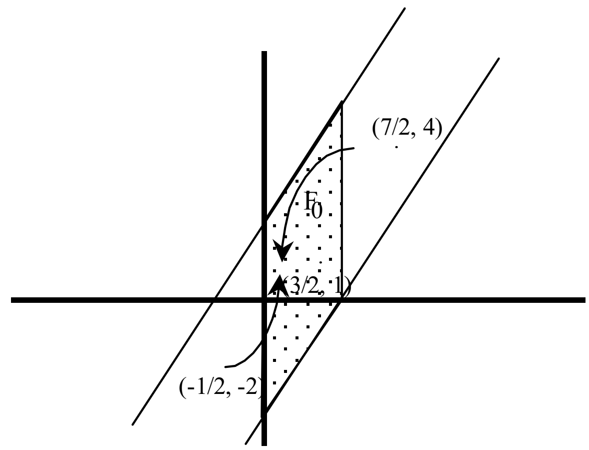

Figure 3.2: *The set F_0 is representative of any point in the (3/2, -3, 2)-stripe*

subpictures can occur, the alphabet is finite. The proof exploits a two-way finite state generator (*2FSG*, for short) with so-called set interpretations as introduced in [20]. This device has a writing head with a finite control and a two-way infinite working tape. Every cell of the tape contains an initially empty set of symbols from an alphabet Δ . At each step it adds a symbol to the set under the writing head, moves in either direction and changes state. More formally, we have the following definition.

Definition 3.2 *[20] A two-way finite state generator (*2FSG*, for short) is a tuple H = (Q, τ , d, A_0, A_f,Δ, pr), where* Q *is a finite nonempty set of states,* τ : Q $\rightarrow 2^Q$ *is a transition function,* d: Q $\rightarrow 2^{\{-1,0,1\}}$ *is a direction function, which indicates whether the writing head can move to the left (-1), to the right (1) or can remain in the same position, A_0 is the initial state, A_f is the accepting state, Δ is the printing alphabet and, finally* out: $Q^+ \rightarrow \Delta$ *is a printing function.*

In the sequel we recall from [66] the concepts of computations and generated language related to two-way finite state generators. The computations of a *2FSG H* are described by strings over ($Q \times$ Z), where (A, j) means that "the current state is A and the current position on the tape is j". The transition function τ induces a move relation $\vdash$ associated to a *2FSG*. Let c$\in$(Q $\times$ Z)*, (A, j)$\in$(Q $\times$ Z). Then

$$c(A,j) \vdash c(A,j)(A', j+i)$$

if $A' \in \tau$, i$\in$d(A). The transitive closure of $\vdash$ is denoted by $\vdash^*$ and it allows us to define the set of valid computations of H, denoted by *comp(H)*, and defined as

$$comp(H) = \{\text{c} \in (\text{Q} \times \text{Z})^*(A_f \times \text{Z}) \mid (A_0, 0) \vdash^* \text{c}\}.$$

For a prefix of a valid computation c$\in$(Q $\times$ Z)*, the leftmost (rightmost) position visited by c, denoted by *lm(c)*, (*rm(c)*, resp.) is defined as the minimal (maximal, resp.) j such that c can be written as $c = c_1$(A, j)c_2, for some c_1, c_2 $\in$(Q $\times$ Z)*, (A, j) $\in$ (Q $\times$ Z). Given a valid computation c, the history-j-homomorphism h_j: (Q $\times$ Z)* $\rightarrow$ Q* is defined as

$$h_j(A, i) = \begin{cases} A & \text{if i = j} \\ \epsilon & \text{if i} \neq \text{j} \end{cases}$$

Such a mapping allows us to describe the sequence of states of the finite control, in which the writing head passed position j on the tape. The interpretation function can be defined using a printing function pr: Q $\rightarrow$ Λ where Λ is an alphabet such that Δ=2^Λ. Finally, the cell j will contain the element $out_j(c)$ = { *pr(A)* | A $\in$ *alph(h_j(c))* }, where *alph(h_j(c))* denotes the set of symbols which occur in h_j(c). Thus the word generated by a computation c is

$$word(c) = out_a(\text{c})out_{a+1}(\text{c})out_{b-1}(\text{c})out_b(\text{c}) \in \Lambda^*,$$

where a = *lm(c)* and b = *rm(c)*. Finally, the language generated by H is defined as

$$lang(H) = \{word(c) \mid \text{c} \in comp(H)\} \subseteq \Lambda^*.$$

In [20] it has been proved that the language generated by a *2FSG* is a regular language. Now, we are ready to give the string language characterization for a subclass of regular drawn stripe picture languages.

Lemma 3.1 *Let k be a rational number and d_1 and d_2 real numbers ($d_1 \leq 0 \leq d_2$). There exists an alphabet Γ and an encoding μ from the set of drawn symbolic (k, d_1, d_2)-stripe pictures into Γ^* with the following properties:*

1. *For two drawn symbolic (k, d_1, d_2)-stripe pictures q_1 and q_2, we have $\mu(q_1) = \mu(q_2)$ if and only if $q_1 = q_2$.*

2. *For a drawn symbolic (k, d_1, d_2)-stripe picture q, we can compute $\mu(q)$ in linear time.*

3. *If D is a regular drawn symbolic (k, d_1, d_2)-stripe picture language generated by a drawn symbolic grammar $\mathbf{G} =< G_{\Sigma\cup\Pi}, dspic_f() >$ where f is commutative and associative merging function then*

$$\mu(D) = \{\ \mu(q) |\ q \in D\}$$

is a regular string language, which can be effectively constructed from D.

Proof. *Let $k = n/m$, such that n and m have no common divisor and $m > 0$. Consider the (m, n, d_1, d_2)-unit labeled line field:*

$$LF_1 = \{\{[a,\ v],\ [b,\ v']\}\ |\ a,\ b \in \Sigma,\ \{v,\ v'\} \in M_1,\ v \in F_0,\ v' \in (F_0 \cup t_{m,n}(F_0))\}$$

The alphabet consists of the set of subsets of $LF_0 \cup LF_1 \cup \{\xi,\ \$\}$. Any subpicture in the sliced portion will be encoded by a subset of LF_1. Moreover, the presence of the endpoint within the sliced portion of the subpicture will be indicated by means of an element in LF_0 and a \$. While ξ will denote the presence of the start point in the sliced portion. The position of the start point will be (0,0). More formally, let $q = <<r,\ (0,0),\ e>,\ \Sigma, \delta >$ be a drawn symbolic $(k,\ d_1,\ d_2)$-stripe picture. Let us define for all integers i,

$$\sigma_{-i} = \begin{cases} (\delta(r) \cap LF_1) \cup \{\xi\} & \text{if } i = 0,\ e \notin F_0 \\ (\delta(r) \cap LF_1) \cup \{\xi, \$, [\delta(e), e]\} & \text{if } i = 0,\ e \in F_0 \\ (t_{im,in}(\delta(r)) \cap LF_1) & \text{if } i \neq 0, t_{im,in}(e) \notin F_0 \\ (t_{im,in}(\delta(r)) \cap LF_1) \cup \{\$, [\delta(e), t_{im,in}(e)]\} & \text{if } i \neq 0, t_{im,in}(e) \in F_0 \end{cases}$$

where $\delta(r) = \{\{[a,\ v],\ [b,\ v']\}\ |\ a,\ b \in \Sigma_\phi,\ (v,\ v') \in r,\ \delta(v)=a,\ \delta(v')=b\}$, *and*

$$t_{im,in}(\delta(r)) = \{\{[a,\ t_{im,in}(v)],\ [b,\ t_{im,in}(v')]\} \in \{[a,\ v],\ [b,\ v']\} \in \delta(r)\}.$$

For example, given the drawn symbolic (1/3, -1, 2)-stripe picture depicted in Fig.3.3(a), the set

$F_0 = \{(i,\ j)\ |(i,\ j)|\ M_0, 0 \leq i < 3\}$ *and*

$\sigma_0 = \{\{[a,\ (0,0)],\ [b,\ (0,1)]\},\ \{[a,\ (0,0)],\ [d,\ (1,0)]\},\ \{[d,\ (1,0)],\ [b,\ (2,0)]\},\ \{[b,\ (2,0)],\ [c,\ (3,0)]\},\ \xi\}$

$\sigma_{-1} = \{\{[b,\ (3,2)],\ [c,\ (2,2)]\},\ \{[c,\ (2,2)],\ [d,\ (2,1)]\},\ \$,\ [d,\ (2,1)]\}$

σ_1 = {{[c, (0,-1)], [c, (0,0)]}, {[c, (0,0)], [b, (1,0)]}}

since

δ*(r)* = {{[a, (0,0)], [b, (0,1)]}, {[a, (0,0)], [d, (1,0)]}, {[d, (1,0)], [b, (2,0)]}, {[b, (2,0)], [c, (3,0)]}, {[c, (3,0)], [c, (3,1)]}, {[c, (3,1)], [b, (4,1)]}, {[b, (0,1)], [c, (-1,1)]}, {[c, (-1,1)], [d, (-1,0)]}}

$t_{3,1}$(δ(r)) = {{[a, (3,1)], [b, (3,2)]}, {[a, (3,1)], [d, (4,1)]}, {[d, (4,1)], [b, (5,1)]}, {[b, (5,1)], [c, (6,1)]}, {[c, (6,1)], [c, (6,2)]}, {[c, (6,2)], [b, (7,2)]}, {[b, (3,2)], [c, (2,2)]}, {[c, (2,2)], [d, (2,1)]}}

$t_{3,1}$(δ(r))$\cap$ LF_1 = {{[b, (3,2)], [c, (2,2)]}, {[c, (2,2)], [d, (2,1)]}}

$t_{-3,-1}$(δ(r)) = {{[a, (-3,-1)], [b, (-3,0)]}, {[a, (-3,-1)], [d, (-2,-1)]}, {[d, (-2,-1)], [b, (-1,-1)]}, {[b, (-1,-1)], [c, (0,-1)]}, {[c, (0,-1)], [c, (0,0)]}, {[c, (0,0)], [b, (1,0)]}, {[b, (-3,0)], [c, (-4,0)]}, {[c, (-4,0)], [d, (-4,-1)]}}

$t_{-3,-1}$(δ(r))$\cap$ LF_1 = {{[c, (0,-1)], [c, (0,0)]}, {[c, (0,0)], [b, (1,0)]}}

Now, let a be the minimal i such that $\sigma_i \neq \emptyset$ *and b be the maximal i such that* $\sigma_i \neq \emptyset$. *The string* $\mu(q) = \sigma_a\ \sigma_{a+1}...\sigma_0...\sigma_b$ *represents the encoding of the picture q. For the above example* $\mu(q) = \sigma_{-1}\sigma_0\sigma_1$, *which is shown in Fig.3.3(b).*
Thus, it is easy to verify that the first two statements of the lemma hold. In order to prove the point (3), namely the correspondence between regular string languages and regular drawn symbolic stripe picture languages generated by a **G** =<$\mathrm{G}_{\Sigma\cup\Pi}$, *dspic*$_f$*()*> *grammar where f is commutative and associative, we construct a two-way finite state generator H which generates the string language lang(H) =* μ*(D) where D is a regular language. The thesis of point (3) follows then from the regularity of lang(H).*

Let **G** =<$\mathrm{G}_{\Sigma\cup\Pi}$, *dspic*$_f$*()*> *be a regular drawn symbolic grammar with f commutative and associative such that D = dspic*$_f$*(L(G)) is a drawn symbolic (k,* d_1, d_2*)-stripe picture language. Without loss of generality, we can suppose that G=(* Σ, Π, *N, P, S) is in right linear form, i.e., G has productions of the form* $S \rightarrow \alpha A$, $S \rightarrow \alpha$, $S \rightarrow \epsilon$, $A \rightarrow \pi\alpha B$, $A \rightarrow \pi\alpha$, *where the start symbol S does not occur on the right-hand of the production in P,A,B*$\in$*N,* $\alpha \in \Sigma^*$, *and* $\pi \in \Pi$.

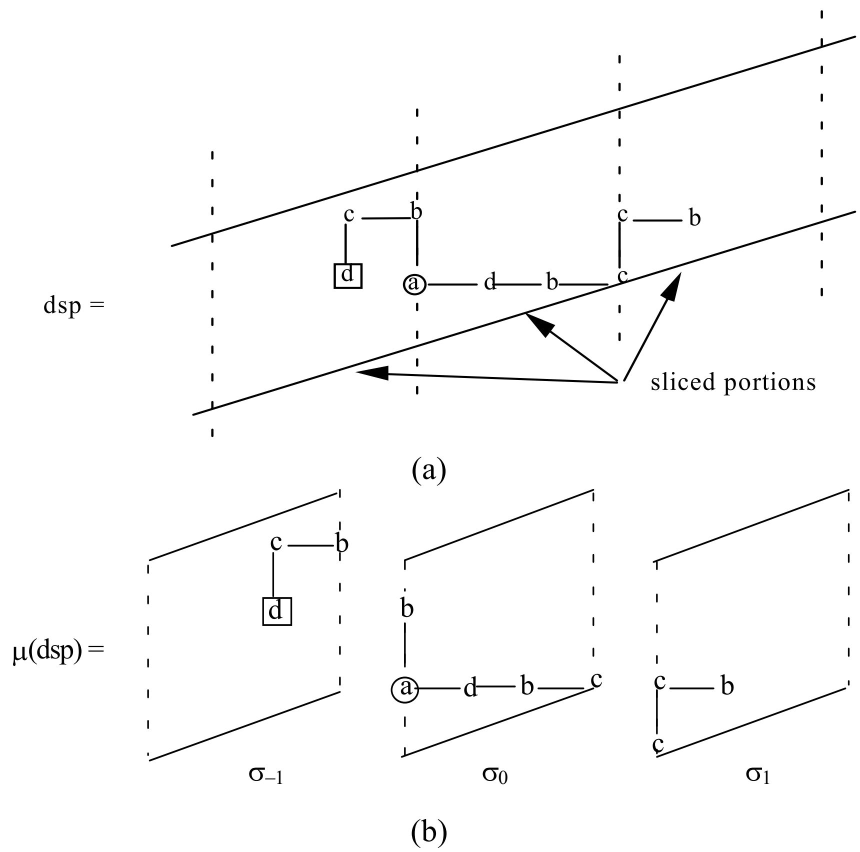

Figure 3.3: *(a) A drawn symbolic (1/3, -1, 2)-stripe picture. (b) Its corresponding sequence $\mu(q)$.*

Now, we show how to construct a 2FSG which simulates the drawing of a symbolic picture w in ***L(G)***. *The cells of the working tape of the automaton will correspond to the fields σ_i of the encoded drawn symbolic stripe picture. The 2FSG $H = (Q, \tau, d, A_0, A_f, \Delta, pr)$ is defined as follows:*

1. $Q=\{A_0, A_f\}$
   $\cup\{<A, v_1, v_2, i> \mid A{\in}N, \{v_1, v_2\}{\in} LF_1, i{\in}\{\text{-}1, 0, 1\}\}$
   $\cup\{<S, \xi, (0,0), i> \mid S \text{ is the start symbol of } G, i{\in}\{\text{-}1, 0, 1\}\}$
   $\cup\{<S, \xi, [a,(0,0)], i> \mid a \in \Sigma_\phi, i{\in}\{\text{-}1, 0, 1\}\}$
   $\cup\{<\epsilon, v, \$, 0> \mid v \in LF_0{\cup}\{(0,0)\}\}$
   $\cup\{<\epsilon, v_1, v_2, 0> \mid \{v_1, v_2\}{\in} LF_1\}$;

2. $\tau(A_0) = \{<S, \xi, (0,0), i> \mid i \in \{-1, 0, 1\}\}$;
   $\tau(A_f) = \emptyset$;
   $\tau(<\epsilon, v, \$, 0>) = \{A_f\}$ *for* $v \in LF_0 \cup \{(0,0)\}$;
   \- $<A, \xi, [a, (0,0)], i> \in \tau(<S, \xi, (0,0), i>)$ *if and only if* $S \rightarrow \alpha A$ *is in* P, *where* $\alpha \in \Sigma^*$, *a is in* α *and if* $\alpha = \epsilon$ *then* $a = \phi$;
   \- $<\epsilon, [a, (0,0)], \$, 0> \in \tau(<S, \xi, (0,0), i>)$ *if and only if* $S \rightarrow \alpha$ *is in* P, *where* $\alpha \in \Sigma^*$, *a is in* α *and if* $\alpha = \epsilon$ *then* $a = \phi$;
   \- *for* $<B, w_1, w_2, j>, <A, v_1, v_2, i> \in Q$ (v_1 *possibly* ξ)
   $<B, w_1, w_2, j> \in \tau(<A, v_1, v_2, i>)$ *if and only if* $A \rightarrow \pi\alpha B$ *is in* P, *where* $\pi \in \Pi$, $\alpha \in \Sigma^*$, $w_1 = [\phi, t_{-im,-in}(x,y)]$, $v_2 = [b, (x,y)]$ *and* $w_2 = [a, \pi(w_1)]$, *a is in* α *and if* $\alpha = \epsilon$ *then* $a = \phi$;
   \- *for* $<\epsilon, w_1, w_2, 0>, <\epsilon, w_2, \$, 0>, <A, v_1, v_2, i> \in Q$ (v_1 *possibly* ξ)
   $\{<\epsilon, w_1, w_2, 0>, <\epsilon, w_2', \$, 0>\} \subseteq (<A, v_1, v_2, i>)$
   if and only if $A \rightarrow \pi\alpha$ *is in* P, *where* $\pi \in \Pi$, $\alpha \in \Sigma^*$, $w_1 = [\phi, t_{-im,-in}(x,y)]$, $v_2 = [b, (x,y)]$, $w_2 = [a, \pi(w_1)]$, $w_2' = [\phi, \pi(w_1)]$, *a is in* α *and if* $\alpha = \epsilon$ *then* $a = \phi$;

3. $d(A_0) = d(A_f) = \{0\}$;
   $d(<A, v_1, v_2, i>) = \{i\}$ *with* $<A, v_1, v_2, i> \in Q$;

4. $\Delta = LF_0 \cup LF_1 \cup \{\xi, \$, (0,0)\}$

5. *pr*: $Q \rightarrow \Delta$, *is such that:*
   $pr(A_0) = \xi$,
   $pr(<S, \xi, (0,0), i>) = \xi$,
   $pr(<A, \xi, [a, (0,0)], i>) = \xi$,
   $pr(<\epsilon, v, \$, 0>) = v$,
   $pr(<\epsilon, v_1, v_2, 0>) = \{v_1, v_2\}, v_1 \neq \xi, v_2 \neq \$$,
   $pr(<A, v_1, v_2, i>) = \{v_1, v_2\}, v_1 \neq \xi, v_2 \neq \$$,
   $pr(A_f) = \$$.

In order to show that lang(H) is equivalent to $\mu(D)$, *we give the interpretation of the states for a 2FSG which simulates the generation of a word w in L(G). The state* $<S, \xi, (0,0), i>$ *means that the generation starts, and (0,0) is the current position of the drawing head in the field, and i denotes the position of the next move (-1 for 'left', 0 for 'no move'* *and +1 for 'right'). <A, ξ, [a, (0,0)], i> means that the generation is started by writing symbol a on the starting position (0,0). A state <A, v_1, v_2, i> with v_1 = [a, w_1], v_2 = [b, w_2] means that: the derivation uses a nonterminal A, a line w_1, w_2 is drawn in the cell under the writing head and the labels a, b are assigned to the points w_1 and w_2, respectively. Finally, the states < ϵ, v, \$, 0> mean that the generation ended in position v in the cell under the writing head.*

Let us observe that the cells of the working tape of a 2FSG could not correspond to the fields σ_i of the encoded drawn symbolic stripe picture, since two different symbols could be associated to the same position of a picture, thus lang(H) is not equivalent to $\mu(D)$. In order to overcome such problem, we will introduce two homomorphisms to be applied to lang(H) generated by the previous 2FSG. Since lang(H) is regular and the regular languages are closed under homomorphism [35], the language obtained is also regular. Let us introduce the homomorphism h_1 that substitutes in the fields σ_i of the encoded drawn symbolic stripe picture, the different symbols associated to the same position with the symbol obtained by applying the function f on such symbols. The homomorphism h_1:$\Gamma \rightarrow \Gamma$ is defined as in the following:

$$h_1(\sigma) = \begin{cases} h_1(\sigma') & \textit{if } \exists\{[a_1,(x,y)],v_1\},\ldots,\ \{[a_n,(x,y)],v_n\},[a,(x,y)] \in \sigma \textit{ with } n{\geq}1 \\ & \textit{where } \sigma' = \rho \cup \{\{[f(a_1,...,a_n,a),(x,y)],v_1\},\ldots,\{[f(a_1,...,a_n,a), \\ & (x,y)],v_n\},[f(a_1,...,a_n,a),(x,y)]\} \\ & \textit{with } \rho \cup \{\{[a_1,(x,y)],v_1\},\ldots,\ \{[a_n,(x,y)],v_n\},[a,(x,y)]\}{=}\sigma \\ h_1(\sigma') & \textit{if } \exists\{[a_1,(x,y)],v_1\},\ldots,\ \{[a_n,(x,y)],v_n\} \in \sigma \textit{ with } n{\geq}2 \\ & \textit{where } \sigma' = \rho \cup \{\{[f(a_1,...,a_n,a),(x,y)],v_1\},\ldots,\{[f(a_1,...,a_n,a), \\ & (x,y)],v_n\}\} \\ & \textit{with } \rho \cup \{\{[a_1,(x,y)],v_1\},\ldots,\ \{[a_n,(x,y)],v_n\}\}{=}\sigma \\ \sigma & \textit{otherwise} \end{cases}$$

Thus,

$$\begin{aligned} h_1(lang(H)) &= \{h_1(word(c)) \mid c \in comp(H)\} \\ &= \{h_1(out_a(c)out_{a+1}(c)...out_{b-1}(c)out_b(c)) \mid c \in comp(H)\} \\ &= \{h_1(out_a(c))h_1(out_{a+1}(c))...h_1(out_{b-1}(c))h_1(out_b(c)) \mid c \in comp(H)\} \end{aligned}$$

Let us observe that the points on the border of two contiguous slices are described by both, so two different points in two contiguous slices can represent the same position in the *plane. As an example, in Fig.3.3(b) the points (3,0) in σ_0 and (0,-1) in σ_1 correspond to the position (3,0) in the drawn symbolic picture of Fig. 3.3(a). So it can happen that such points have associated different symbols, even if the homomorphism h_1 has been applied. In order to resolve such ambiguity, we introduce the homomorphism h_2 that is applied to each contiguous couple of elements of the words in lang(H). Let $\Theta = \{w \in \Gamma^* | \ \ |w| = 2\}$, the homomorphism h_2:$\Theta \rightarrow \Theta$ is defined as follows:*

$$h_2(\sigma_1, \sigma_2) = \begin{cases} \sigma_1\sigma_2 & \textit{if not } \exists\{[a,(x,y)], v_1\} \in \sigma_1 \textit{ and } \{[c, t_{-m,-n}((x,y))], v_2\} \in \sigma_2 \\ h_2(\sigma'_1, \sigma'_2) & \textit{otherwise} \\ & \textit{where } \sigma'_1 = \rho_1 \cup \{[f(a,c),(x,y)], v_1\} \textit{ and } \sigma'_2 = \rho_2 \cup \{[f(a,c), \\ & \ t_{-m,-n}((x,y))], v_2\} \textit{ with } \rho_1 \cup \{[a,(x,y)], v_1\} = \sigma_1 \\ & \textit{and } \rho_2 \cup \{[c, t_{-m,-n}(x,y)], v_2\} = \sigma_2 \end{cases}$$

Now, let us introduce some notations.

Given a language L, the following languages can be defined.

$L^1 = \{w \mid w \in L \textit{ and } |w|=1\}$

$L^e = \{w \mid w \in L \textit{ and } |w| \textit{ is even}\}$

$L^o = \{w \mid w \in L \textit{ and } |w| \textit{ is odd}\}$

The above notations allow us to specify the languages L_1 and L_2.

$L_1 = h_1(lang(H))$

$L_2 = L_1^1 \cup h_2(L_1^e) \cup \{h_2(w) | \ w\mathrm{a} \in L_1^o\text{-}L_1^1\}$

Finally we are ready to define the regular language L.

$L = L_2^1 \cup \{h_2(w) | \ awb \in L_2^e \} \cup \{h_2(w) \mid aw \in L_2^o\text{-}L_2^1\}$.

From the properties of the merging function f it follows that $\mu(D)$ corresponds to the regular language L obtained by applying the homomorphisms h_1 and h_2 to lang(H) as described previously.

Now we provide an example of application of the homomorphisms h_1 and h_2. Let us consider the $< G_{\Sigma\cup\Pi}$, $dspic_f() >$ grammar where f is commutative and associative, and it is defined as: f(a,c)=c; f(e,c)=c; f(b,b)=b; f(d,a)=d; f(d,e)=d; f(a,a)=a.

Moreover, let H be the 2FSG constructed from the grammar $G_{\Sigma\cup\Pi}$ and $w = \rho_{-1}\rho_0\rho_1$ *lang*(H) such that:

$\rho_{-1} = \{\{[\phi, (3,2)], [c, (2,2)]\}, \{[\phi, (2,2)], [d, (2,1)]\}, \$, [d, (2,1)]\}$

$\rho_0 = \{\{[a, (0,0)], [d, (1,0)]\}, \{[\phi, (1,0)], [b, (2,0)]\}, \{[\phi, (2,0)], [c, (3,0)]\}, \{[\phi, (3,0)], [b,$

(2,0)]}, {[ϕ, (2,0)], [a, (1,0)]}, {[ϕ, (1,0)], [ϕ, (0,0] }, {[ϕ, (0,0)], [b, (0,1)]}, ξ }
ρ_1= {{[c, (0,-1)], [c, (0,0)]}, {[c, (0,0)], [b, (1,0)]}}
By applying the homomorphism h_1 on ρ_{-1}, ρ_0, ρ_1 we obtain
ρ'_1 = {{[ϕ, (3,2)], [c, (2,2)]}, {[c, (2,2)], [d, (2,1)]}, \$, [d, (2,1)]}
ρ'_0 = {{[a, (0,0)], [d, (1,0)]}, {[d, (1,0)], [b, (2,0)]}, {[b, (2,0)], [c, (3,0)]}, {[ϕ, (3,0)], [b, (2,0)]}, {[b, (2,0)], [a, (1,0]}, {[d, (1,0)], [a, (0,0)]}, {[a, (0,0)], [b, (0,1)]}, ξ}
ρ'_1 = {{[c, (0,-1)], [c, (0,0)]}, {[c, (0,0)], [b, (1,0)]}}.
Finally, the homomorphism h_2 produces the slices σ_{-1}, σ_0, σ_1 of Fig. 3.3(b).

In order to provide an intuitive justification of the need for the associative and commutative properties of the merging functions to ensure the equivalence between *lang*(H) and μ(D), let us observe that the 2FSG is not able to keep track of the symbol insertion order during the construction of the slices. Moreover, if a point is traversed more than once the homomorphism exploits the merging function to establish the actual symbol to associate to such point. As a consequence, if the merging function f is not associative or commutative then different symbols could be associated depending on the application order of *f*. Obviously, this could determine erroneous results because we have already lost such ordering. On the other hand, to keep track the symbol insertion order we should need a device enhanced with memory.

Now, we are ready to provide the following membership result.

Theorem 3.3 *The membership problem for a regular drawn symbolic stripe picture language L generated by a drawn symbolic picture grammar* $\boldsymbol{G}$ =<$G_{\Sigma\cup\Pi}$, *dspic*$_f$*()*> *with f commutative and associative can be decided in linear time.*

Moreover, it is worth noting that the associativity property is turned out to be crucial to ensure that the set (DSP,$\bullet_f$) is a monoid. Thus, we can restate theorem 3.3 as follows.

Theorem 3.4 *Let* (Σ_ϕ, f) *be an abelian monoid. The membership problem for a regular drawn symbolic stripe picture language L generated by a drawn symbolic picture grammar* $\boldsymbol{G}$ =<$G_{\Sigma\cup\Pi}$, *dspic*$_f$*()*> *can be decided in linear time.*

Examples of commutative and associative merging function are *addRGB* and *ord* presented in section 2.2.1. Let us observe that if **G** is *f-independent* then each point in σ_i have associated at most one symbol in Σ and/or a certain number of occurrences of ϕ.

As a consequence the merging function f always produces a unique symbol when applied by the homomorphisms h_1 and h_2 . Thus the associative and commutative hypothesis are no longer needed in order to determine whether or not $lang$(H) is equivalent to μ(D). As an immediate consequence, Lemma 3.1 can be easily extended to *f-independent* drawn symbolic (k,d_1,d_2)-stripe pictures languages, and the following complexity result can be stated.

Corollary 3.1 *The membership problem for a regular drawn symbolic stripe picture language L generated by an f-independent drawn symbolic picture grammar* $\boldsymbol{G} =< G_{\Sigma\cup\Pi}$, $dspic_f()>$ *can be decided in linear time.*

Moreover, the following decidability results can be proved by exploiting Lemma 3.1, corollary 3.1 and the properties of regular string languages.

Theorem 3.5 *Given two regular* $\Sigma \cup \Pi$*-languages* L_1 *and* L_2 *such that* $dspic_f(L_2)$ *is a drawn symbolic stripe picture language. It is decidable whether* $dspic_f(L_1)$=$dspic_f(L_2)$, *and* $dspic_f(L_1)\cap dspic_f(L_2)=\emptyset$ *in the following cases: 1) f is commutative and associative, or 2) L is f-independent.*

Finally, let us observe that in [45] it has been shown that there is a *1-reversal-bounded* linear Π-language describing a stripe picture language for which the membership problem is NP-complete. Such result can be easily extended to drawn symbolic picture languages. A *1-reversal-bounded* $\Sigma \cup \Pi$-language consists of $\Sigma \cup \Pi$-words in which the number of alternating **r**'s and **l**'s is bounded by 1.

3.2 Three-way Picture Languages

In this section we focus on the membership problem for three-way drawn symbolic picture languages and show that the condition provided in the previous section allow us to extend the result given for three-way drawn picture in [42].

In particular, we present a polynomial-time algorithm able to resolve the membership problem of context-free three-way drawn symbolic picture languages and a linear time algorithm for regular three-way drawn symbolic picture languages.

Regular three-way drawn symbolic pictures

Consider a three-letter subset Π' of Π. A picture is called a three-way drawn symbolic picture if it can be described by a $\Sigma \cup \Pi'$-word and a drawn symbolic picture language is called a three-way drawn symbolic picture language if it can be described by a $\Sigma \cup \Pi'$-language. For the discussion that follows, we shall fix Π' to be $\{\mathbf{u}, \mathbf{d}, \mathbf{r}\}$.

A three-way drawn symbolic picture can be described by a sequence of line-disjoint horizontal and vertical line (with symbols on Σ) which come in turn. For example, the drawn symbolic picture p depicted in Fig. 3.4(a) can be partitioned into eight subpictures p_1, p_2,..., p_8, as depicted in Fig. 3.4(b) from left to right, so that $p= p_1 \bullet_f p_2 \bullet_f ... \bullet_f p_8$ and it is denoted by $part_f(p)$=(p_1, p_2,..., p_8). Let us observe that the properties of function f (commutative and associative) ensure that, for i=1,2,..8, if w and w_{p_i} are such that $dspic_f(w)=p$ and $dspic_f(w_{p_i})=p_i$, $dspic_f(w_{p_1}) \bullet_f dspic_f(w_{p_2}) \bullet_f ... \bullet_f\ dspic_f(w_{p_8}) = dspic_f(w)$. It is worth noting that each p_i is a stripe picture, so we can reduce the membership problem for regular three-way drawn symbolic picture languages to the membership for drawn symbolic stripe picture languages.

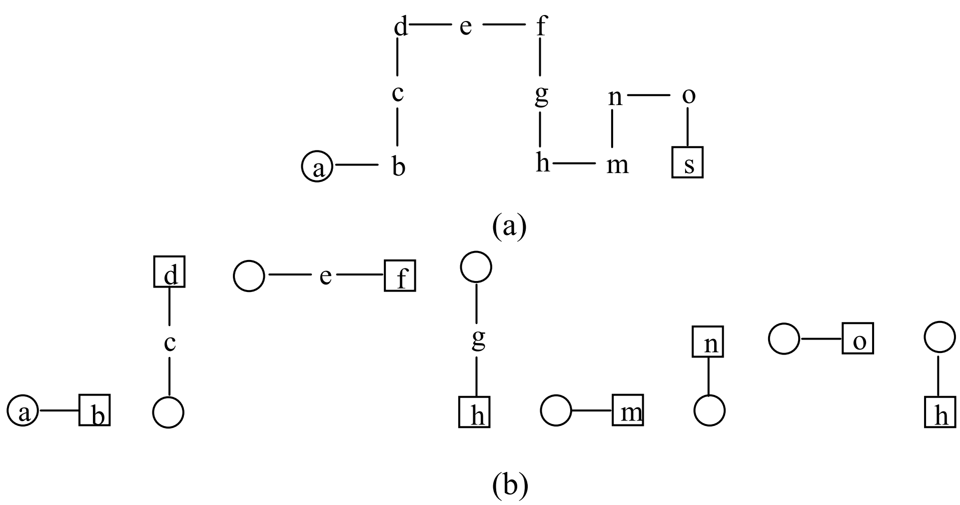

Figure 3.4: *(a) A three-way drawn symbolic picture and (b) its partition.*

In the following, we shall assume that the $\Sigma \cup \Pi'$-language is specified by a $\Sigma \cup \Pi'$-automaton M. An automaton M is in reduced form if, for each state q, q is reachable from the initial state by a finite sequence of transitions of M and a final state is reachable from

q by a finite sequence of transitions of M.

Let M=({0,1,2,...,t}, $(\Pi'\cup\{\epsilon\})\Sigma^*$, δ, 0, t) be a reduced $\Sigma\cup\Pi'$-automaton where {0,1,2,...,t} is a finite set of states of the finite control, $\Pi'\cup\{\epsilon\})\Sigma^*=\{\pi\tau \mid \pi \in(\Pi'\cup\{\epsilon\})$ and $\tau \in \Sigma^*\}$, δ is a transition function that maps the elements of the set {0,1,2,...,t} $\times((\Pi'\cup\{\epsilon\})\Sigma^*)$ into subsets of {0,1,2,...,t}, 0 is the initial state, and t is the final state. Given a drawn symbolic picture, we want to determine whether or not $p\in dspic_f$(M). Let $part_f(p)=(p_1, p_2,..., p_m)$ for some integer m. Without loss of generality we assume that p_1 is an horizontal line segment, m is an even number, the initial state 0 has no input arcs and the transition function is:

$\delta(0,\sigma)$ = i with 0<i$\leq$t and $\sigma \in \Sigma^*$

$\delta(0,x\sigma)$ = undefined $\forall$x$\in \Pi'$ and $\sigma \in \Sigma^*$

otherwise $\forall$i>0

$\delta(i,x\sigma)$ = j with 0<i$\leq$t and $\forall$x$\in \Pi'$ and $\sigma \in \Sigma^*$

$\delta(i,\sigma)$ = undefined $\forall\sigma \in \Sigma^*$

For i=1,2,...,t, j=1,2,...,t and k=0,1,..,t, construct new finite automata ${M_{0,k}}^h$,${M_{i,j}}^h$ and ${M_{i,j}}^v$ as follows:

$${M_{0,k}}^h=(\{0,1,..,t\}, (\Pi'\cup\{\epsilon\})\Sigma^*, \delta^h_{0,k}\ 0, k), \text{ where}$$

$$\delta^h_{0,k}(q,x\sigma) = \begin{cases} \delta(q,x\sigma) & \text{if (x = } \mathbf{r}\text{) or (x = } \epsilon \text{ and q = 0)} \\ \emptyset & \text{if x = } \mathbf{d} \text{ or } \mathbf{u} \end{cases}$$

$${M_{i,j}}^h=(\{1,..,t\}, (\Pi'\cup\{\epsilon\})\Sigma^*, \delta^h_{i,j}\ 0, j), \text{ where}$$

$$\delta^h_{i,j}(q,x\sigma) = \begin{cases} \delta(q,x\sigma) & \text{if (x = } \mathbf{r}\text{)} \\ \emptyset & \text{if (x = } \mathbf{d} \text{ or } \mathbf{u}\text{) or q = 0} \end{cases}$$

$${M_{i,j}}^v=(\{1,..,t\}, (\Pi'\cup\{\epsilon\})\Sigma^*, \delta^v_{i,j}\ 0, j), \text{ where}$$

$$\delta^v_{i,j}(q,x\sigma) = \begin{cases} \delta(q,x\sigma) & \text{if x = } \mathbf{u} \text{ or } \mathbf{d} \\ \emptyset & \text{if (x = } \mathbf{r}\text{) or (q = 0)} \end{cases}$$

The words accepted by ${M_{0,k}}^h$ are those obtained by running the $\Sigma \cup \Pi'$-automaton M, starting from state 0 and ending at state k, using only transitions with **r**-movements. The words accepted by ${M_{i,j}}^v$ (${M_{i,j}}^h$, respectively) are those obtained by running the $\Sigma \cup \Pi'$-automaton M, starting from state i and ending at state j. Thus, $dspic_f({M_{i,j}}^v)$ ($dspic_f({M_{i,j}}^h)$, respectively) is the set of all vertical (horizontal, respectively) line segments with symbols in Σ_ϕ that can be described by these words. For example let M be the $\Sigma \cup \Pi'$-automaton described by the transition graph depicted in Fig. 3.5. Then ${M_{0,3}}^h$, ${M_{1,2}}^v$ and ${M_{3,2}}^h$ are the $\Sigma \cup \Pi'$-automata depicted in Figg. 3.6(a),(b),(c) respectively.

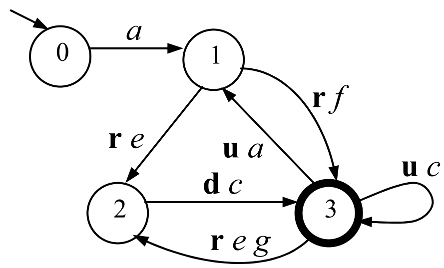

Figure 3.5: *The $\Sigma \cup \Pi'$-automaton M.*

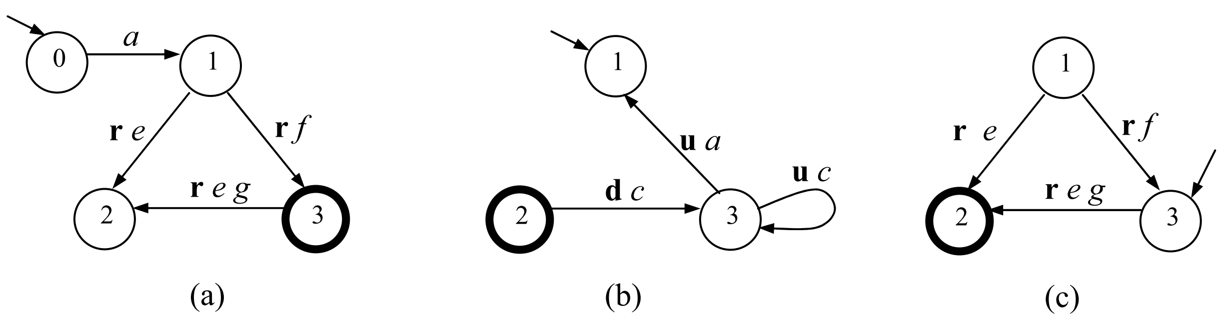

Figure 3.6: *The $\Sigma \cup \Pi'$-automata (a) ${M_{0,3}}^h$, (b) ${M_{1,2}}^v$, (c) ${M_{3,2}}^h$*

For s=1,2,...,m, let T_s be a Boolean matrix (t+1)×(t+1) defined by:

$$T_s[i,j] = \begin{cases} 1 & \text{if } p_s \in dspic_f(\mathrm{M}^a_{i,j}) \\ 0 & \text{otherwise} \end{cases}$$

where a=h if s odd, and a=v if s even. Let T be a Boolean matrix (t+1) (t+1) defined by

$$\mathrm{T}=\prod_{s=1}^{m}\mathrm{T}_s$$

Before introducing the algorithm for the membership we give the following lemma.

Lemma 3.2 *$p \in dspic_f(M)$ if and only if $T[0, t]=1$.*

Proof. *$p \in dspic_f(M)$ if and only if exists a sequence of finite $\Sigma \cup \Pi'$-automata $M_{i_1,j_1}{}^h, M_{i_2,j_2}{}^v$, ..., $M_{i_{m-1},j_{m-1}}{}^h$, $M_{i_m,j_m}{}^v$ such that $i_1=0$, $i_s=i_{s-1}$, for all $s=2,3,\ldots,m$, $j_m=t$, and $p_k \in dspic_f(M_{i_k,j_k}{}^a)$ for all $k=1,2,\ldots,m$, where $a=h$ if k odd, and $a=v$ if k even. From this, we have that $p \in dspic_f(M)$ if and only if $T_1[i_1, j_1]=T_2[i_2, j_2]=\ldots= T_m[i_m, j_m]= 1$ if and only if $T[0, t] = 1$.*

In the following we present an application of the previous lemma.

Example 3.2 *Let p be the three-way drawn symbolic picture depicted in Fig. 3.7. It can be described by the $\Sigma \cup \Pi'$-word $w = \mathrm{a}\boldsymbol{r}\mathrm{f}\boldsymbol{u}\mathrm{c}\boldsymbol{u}\mathrm{a}\boldsymbol{r}\mathrm{e}\boldsymbol{d}\mathrm{c}$ and $part_f(p)=(p_1, p_2, p_3, p_4)$ as depicted in Fig. 3.8. The $\Sigma \cup \Pi'$-words: $w_1=\mathrm{a}\boldsymbol{r}\mathrm{f}$, $w_2=\boldsymbol{u}\mathrm{c}\boldsymbol{u}\mathrm{a}$, $w_3=\boldsymbol{r}\mathrm{e}$, $w_4=\boldsymbol{d}\mathrm{c}$ describe p_1, p_2, p_3, p_4, respectively. The properties of f ensure that $dspic_f(w_1) \bullet_f dspic_f(w_2) \bullet_f dspic_f(w_3) \bullet_f dspic_f(w_4)= dspic_f(w)$.*
If M is the $\Sigma \cup \Pi'$-automaton defined in the last example, then $p \in dspic_f(M)$ because the sequence of $\Sigma \cup \Pi'$-automata $M_{0,3}{}^h$, $M_{3,1}{}^v$, $M_{1,2}{}^h$, $M_{2,3}{}^v$, as depicted in Fig. 3.9, is such that $T[0,3]=1$.

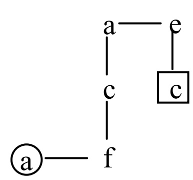

Figure 3.7: *A three-way drawn symbolic picture*

In Fig. 3.10 it is described the membership algorithm for three-way regular drawn symbolic picture languages. The algorithm take $O(|p|)$ because each step take $O(|p|)$ time. In particular in step (3), determining each entry of matrix T_k is the membership problem for a regular drawn symbolic stripe picture language; it takes $O(|p_k|)$ time. Therefore, step (3) takes $O(t^2|p1| + t^2|p_2| + \ldots + t^2|p_m|) = O(|p|)$ time. So the membership problem

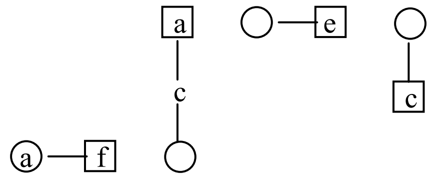

Figure 3.8: *The four subpictures of part(p).*

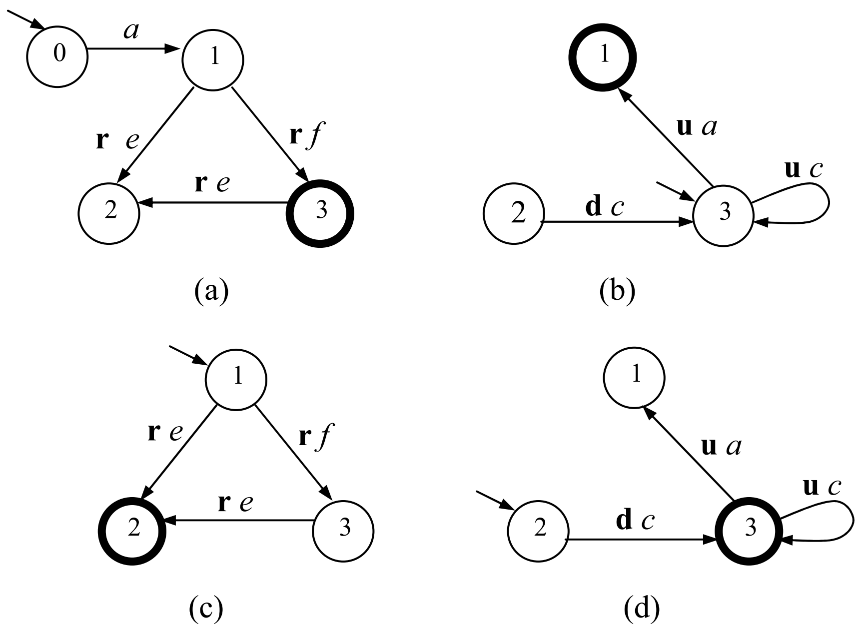

Figure 3.9: *The* $\Sigma \cup \Pi'$*-automata* $M_{0,3}{}^h$, $M_{3,1}{}^v$, $M_{1,2}{}^h$, $M_{2,3}{}^v$.

for three-way regular drawn symbolic picture languages is decidable deterministically in polynomial time.

As a consequence we have the following theorem.

Theorem 3.6 *Let* (Σ_ϕ, f) *be an abelian monoid. The membership problem for regular three-way drawn symbolic picture languages generated by drawn symbolic grammars* ***G***=<$G_{\Sigma\cup\Pi}$, *dspic*$_f$*()*> *can be decided deterministically in polinomial time.*

Let us observe that, as a consequence of corollary 3.1, if **G** is *f-independent* the previous result also holds if the function f is not commutative and associative. Thus, the following corollary can be stated.

Algorithm 3.1. Membership test.

INPUT: A three-way drawn symbolic picture p and a fixed finite $\Sigma\cup\Pi$'-automaton $M=(\{0,1, .., t\}, (\Pi'\cup\{\varepsilon\})\Sigma^*, \delta, 0, t)$.

OUTPUT: *yes* if $p\in dspic_f(M)$, *no* otherwise.

1. Compute $part_f(p)=(p_1, p_2, \ldots, p_m)$;
2. construct $M_{i,j}^{v}$ and $M_{k,j}^{h}$ for all i, j, k such that $1\leq i, j \leq t$ and $0\leq k \leq t$;
3. construct T_s for s=1, 2, …, m;
4. compute T;
5. **if** T[0, t]=1 **then** output "yes", **else** output "no".

Figure 3.10: *The membership test for three-way regular drawn symbolic picture languages.*

Corollary 3.2 *The membership problem for a regular three-way drawn symbolic picture language L generated by an f-independent drawn symbolic picture grammar* $\boldsymbol{G}$ $=<G_{\Sigma\cup\Pi}$, $dspic_f()>$ *can be decided deterministically in polynomial time.*

.

Context-free three-way picture languages

Let us consider a push down automaton (PDA) $M=(Q, (\Pi'\cup\{\epsilon\})\Sigma^*, \Gamma, \gamma, q_0, Z_0, q_f)$ and a drawn symbolic picture dsp=<<p, s, e>, Σ, δ^p >. We want to determine whether or not $dsp \in dspic_f(M)$.

A PDA works as described in the following. The value given by a transition function γ is a triple (q, X, d) where q is a state in Q, X is in {0,1}, and d in Γ. This means that the possible next actions of PDA are 1) entering next state q, 2) moving the input head to the right one cell or leaving it stationary, according to whether X is 1 or 0, and 3) adding the symbol d to the top of the pushdown store. Moreover if (q, X, pop) is the value of the transition function then M removes the top pushdown store symbol instead of adding a symbol to the pushdown store, in step 3). We can assume that M accepts by entering the final state q_f with the empty pushdown store, scanning the right endmarker of the input tape. A configuration of M is a 6-tuple $(q, w, p', v, v', \delta^{p'})$ which means that the PDA M is in state q, the current content of the pushdown store is w (where the first symbol of w is the symbol at the top of the pushdown store), and M has read some portion of the input tape, say x, such that $dspic_f(x)$=dsp'=<<p', v, v' >, Σ, $\delta^{p'}$> where for each $v \in$W(< p', v, v' >), $\delta^{p'}(v)$=$\sigma \in \Sigma_\phi$.

Given two configurations I and J, I $\vdash$ J if and only if the PDA M in configuration I can move in one step to configuration J. Moreover, I $\vdash^k$ J establishes that M can move from configuration I to configuration J after exactly k moves. $\vdash^*$ denotes the transitive reflexive closure of the binary relation $\vdash$.

A *7-tuple* (q, A, r, p', v, v', $\delta^{p'}$) is *dsp-valid* if the PDA M can go from state q with symbol A alone on the pushdown store to state r with an empty store, drawing the picture $dsp' \subseteq dsp$, starting from $v \in M_0$ and ending at $v' \in M_0$, i.e., if (q, A, $\emptyset$, v, v, δ) $\vdash^*$ (r, $\emptyset$, p', v, v', $\delta^{p'}$). Moreover, let W(<p, s, e>)= {v_1(=s), v_2,...,v_m(=e)} and X be a matrix such that, for each i and j with $1 \leq i,j \leq m$, X[i,j]= {(q, A, r, p', v_i, v_j, $\delta^{p'}$) | q,$r \in$Q, A$\in$ ($\Gamma \cup \{Z_0\}$) and (q, A, r, p', v_i, v_j,Σ, $\delta^{p'}$) is a *dsp-valid 7-tuple*} .

As a consequence, in order to test the membership of three-way drawn symbolic picture we need only to construct the matrix X. To this aim we can use a dynamic programming method defined by the following function YEALD$_M$.

YEALD$_M$(S$_1$,S$_2$)= {(q, B, t, p', v, v_3, $\delta^{p'}$) |

(q, B, $\emptyset$, v, v, δ) $\vdash$ (r, AB, p_1, v, v_1, δ^{p1}),

(r, A, s, p_2, v_1, v_2, δ^{p2})$\in S_1$,

(s, B, t, p_3, v_2, v_3, δ^{p3})$\in S_2$,

and << p', v, v_3>, Σ, $\delta^{p'}$>=

<<p_1, v, v_1>, Σ, δ^{p1}>$\bullet_f$<<p_2, v_1, v_2>, Σ, δ^{p2}>$\bullet_f$<<p_3, v_2, v_3>, Σ, δ^{p3}> }.

The idea behind the definition of function YEALD$_M$ is that, from the partial computations of M corresponding to the drawing of picture << p_1, v, v_1 >, Σ, δ^{p1}>, << p_2, v_1, v_2 >, Σ, δ^{p2} > and << p_3, v_2, v_3>, Σ, δ^{p3} >, we deduce (q, B, $\emptyset$, v, v, δ) $\vdash^+$(t, $\emptyset$, p', v, v_3, $\delta^{p'}$).

The dynamic programming method works as described in the following:

1. puts in the table X all *dsp-valid 7-tuples* corresponding to single pop transitions of M,

2. combines repeatedly the information in the table X that correspond to partial computations of M until no new information can be added to X, and

3. checks whether or not the element (q_0, Z_0, q_f, p, s, e, δ^p) has been added to X[1,m].

It is worth noting that the associative and commutative properties of merging fuctions play a crucial role in the construction of the matrix X. As a matter of fact, the above technique is not able to keep track of the symbol insertion order in the subpictures corresponding to the partial computations. Moreover, if a point is traversed more than once $YELAD_M$ exploits the merging function f to establish the actual symbol to associate to the point. In particular, in the application of YEALD$_M$, for each $v \in$W(p') if v is in p_1, p_2 and p_3, the symbol associated to v is given by $\delta^{p'}(v)=f(f(\ \delta^{p1}(v),\ \delta^{p2}(v)),\delta^{p3}(v))$.
As a consequence, if the merging function *f* is not associative or commutative then different symbols could be associated depending on the application order of *f*. Obviously, this could determine erroneous results during the combination of the partial computations in the construction of X because we have already lost such ordering.

In Figg. 3.11 and 3.12, we provide the Algorithm for the membership test for three-way drawn symbolic picture languages. It is an adaptation of Algorithm 3.5 introduced by Kim for drawn picture which has been modified according to the above definitions of matrix X and function YEALD$_M$.

Algorithm 3.2 Membership test for three-way drawn symbolic picture languages

INPUT: A three-way drawn symbolic picture *dsp*=<<p, s, e>, Σ, δ^p>, and a fixed finite PDA M=(Q, ($\Pi' \cup \{\varepsilon\})\Sigma^*$, Γ, δ, q_0, Z_0, q_f).

OUTPUT: *yes* if *dsp*∈*dspic$_f$*(M), *no* otherwise.

1. If *dsp* is not a three-way drawn symbolic picture, then output "no"
2. Set the stack T to be empty, and set X[i,j]=Ø, with 1≤ i,j ≤ m(=|W(<p, s, e>)|);
3. add to T all *dsp-valid 7-tuples*, say (q, A, r, p', v_i, v_j, $\delta^{p'}$) corresponding to single pop transitions of M. Simultaneously, store (q, A, r, p', v_i, v_j, $\delta^{p'}$) in X[i,j];
4. **while** T is not empty **do** call BUILDUP;
5. **if** X[1,m] contains (q_0, Z_0, q_f, p, s, e, δ^p >) **then** output "yes" **else** output "no".

Figure 3.11: *The membership test for three-way context-free languages.*

procedure BUILDUP;

(a) Pop the top entry, say top=(r, A, s, p', v_i, v_j, $\delta^{p'}$) of the stack T;

(b) /* right bulidup*/

for k=1 **to** m **do**

let S be the set of 7-tuple corresponding to X[j,k];

Compute E=YIELD$_M$({top}, S);

for each 7-tuple, say Y=(q, B, t, p', v_l, v_k, $\delta^{p'}$), in E

if dsp'=<<p', v_l, v_k>, Σ, $\delta^{p'}$> is a subpicture of dsp and Y is not in X

then add (q, B, r, p', v_l, v_k, $\delta^{p'}$) to X[l,k] and put Y in T;

endfor;

(c) /* left bulidup*/

for k=1 **to** m **do**

let S be the set of 7-tuple corresponding to X[k,i];

Compute E=YIELD$_M$(S, {top});

for each 7-tuple, say Y=(q, B, t, p', v_l, v_j, $\delta^{p'}$), in E

if dsp'=<<p', v_l, v_j>, Σ, $\delta^{p'}$> is a subpicture of dsp and Y is not in X

then add (q, B, r, p', v_l, v_j, $\delta^{p'}$) to X[l,j] and put Y in T;

endfor;

endprocedure.

Figure 3.12: *The procedure BUILDUP used by the algorithm for the membership test.*

In order to prove that every *dsp-valid 7-tuple* gets added to the table X we need the operator $\cup_f$. Given $\langle sc_1, \Sigma, \delta_1\rangle$ and $\langle sc_2, \Sigma, \delta_2\rangle$,
$(\delta_1 \cup_f \delta_2)(v) = \delta_1(v)$ if $v \in sc_1$ and $v \notin sc_2$
$(\delta_1 \cup_f \delta_2)(v) = \delta_2(v)$ if $v \notin sc_1$ and $v \in sc_2$
$(\delta_1 \cup_f \delta_2)(v) = f(\delta_1(v), \delta_2(v))$ if $v \in sc_1$ and $v \in sc_2$

Lemma 3.3 *Every dsp-valid 7-tuple gets added to the table X in the Algorithm 3.2*
Proof. *A dsp-valid 7-tuple (q, B, r, p', v, v', $\delta^{p'}$) is called (k,dsp)-valid if (q, B, $\emptyset$, p', v, v, δ) $\vdash^k$ (r, $\emptyset$, p', v, v', $\delta^{p'}$). Let us observe that every dsp-valid 7-tuple is (k,dsp)-valid for some k. It is easy to show by induction on k that every (k,dsp)-valid 7-tuple gets added to X. For k=1, it is obvious since in step 3 all (1,dsp)-valid 7-tuples are added to T. Let us suppose that the thesis is true for k<n, and prove for k=n. We have that (q, A, $\emptyset$, p', v,*

v, δ) $\vdash^k$ (r, $\emptyset$, p', v, v', $\delta^{p'}$) and the computation is (q, A, $\emptyset$, v, v, δ) $\vdash$ (s, BA, p_1, v, v_1, δ^{p1}) $\vdash^{l1}$ (t, A, $p_1 \cup p_2$, v, v_2, $\delta^{p1} \cup_f \delta^{p2}$) $\vdash^{l2}$ (r, $\emptyset$, $p_1 \cup p_2 \cup p_3$, v, v', $(\delta^{p1} \cup_f \delta^{p2}) \cup_f \delta^{p3}$) for some $l_1, l_2 <n$, where $p_1 \cup p_2 \cup p_3 = p'$. This means that ($s$, B, t, p_2, v_1, v_2, δ^{p2}) is a (l_1,dsp)-valid tuple and (t, B, r, p_3, v_2, v', δ^{p3}) is a (l_2,dsp)-valid tuple. By induction hypothesis both are added to X (and to the stack T). Let us consider the step in which the last one of these two is removed from T. At that time, the other one is in the table X since it is already removed from T. So, the 7-tuple (q, A, r, p', v, v', $\delta^{p'}$) is added to the table at this step by the function $YIELD_M$ in the procedure BUILDUP.

It can be easily proved by extending lemma 3.7 introduced in [42] that Algorithm 3.2 works in polinomial time.

Due to the presence of the symbols in the positions, we have to add a factor of $|\Sigma|$ to the time required for each operation in the steps of the algorithm. But given a $\Sigma \cup \Pi$-grammar $|\Sigma|$ is a constant, so the total time required does not change asyntotically. In particular, given a drawn symbolic picture dsp=<<p, s, e>, Σ, δ >, step 1), 2), 3) take O($|dsp|^2$) time. In order to calculate the time required for the step 4), consider that only dsp-valid tuples are added to X and T, and only when they are first generated. There are O($|dsp|^2$) connected drawn symbolic subpictures of a single orizontal or vertical line segment of dsp. Moreover for every fixed pair of points v, v' $\in$W(<p, s, e>) there are O($|dsp|^4$) subpictures p' of p such that <<p, s, e>, Σ, δ>=$dsp_1 \bullet_f$ << p',v, v' >, Σ, δ > $\bullet_f dsp_2$ for some dsp_1,$dsp_2 \subseteq dsp$. It follows that X[i,j] contains O($|dsp|^4$) elements for all i and j, and that there are O($|dsp|^6$) dsp-valid *7-tuples*. Thus, it means that the procedure BUILDUP is executed O($|dsp|^6$) times, once for every pop operation. Since YEALD_M operations are performed O($|dsp|$) times and each one takes O($|dsp|^5$) time, the total time required for step 4) is O($|dsp|^{12}$). Step 5) requires O($|dsp|^5$) since $|$ X$[1,m]$ $|$=O($|dsp|^4$) and the comparison of (q_0, Z_0, q_f, p, s, e, δ^p) with an element of X$[1,m]$ takes O($|dsp|$) time. Thus the total time required is O($|dsp|^{12}$).

As a consequence we have the following theorem.

Theorem 3.7 *Let (Σ_ϕ, f) be an abelian monoid. The membership problem for context-free three-way drawn symbolic picture languages generated by drawn symbolic picture grammars **G** = <$G_{\Sigma\cup\Pi}$, $dspic_f$()> can be decided deterministically in polinomial time.*

Let us observe that if **G** is *f-independent* then each point in the drawn symbolic pictures have associated at most one symbol in Σ and/or a certain number of occurrences of ϕ. Thus the associative and commutative hypothesis are no longer needed for the function YEALD$_M$ in order to determine (r, A, s, p_2, v_1, v_2, δ^{p2})$\in S_1$, (s, B, t, p_3, v_2, v_3, δ^{p3})$\in S_2$, such that (q, B, $\emptyset$, v, v, δ) $\vdash$ (r, AB, p_1, v, v_1, δ^{p1}) and $<< p'$, v, $v_3 >$, Σ, $\delta^{p'} >=<< p_1$, v, $v_1 >$, Σ, $\delta^{p1} > \bullet_f << p_2$, v_1, $v_2 >$, Σ, $\delta^{p2} > \bullet_f << p_3$, v_2, $v_3 >$, Σ, $\delta^{p3} >$. As a matter of fact, for each $v\in$W(p'), if v is in p_1, p_2 and p_3, the functions δ^{p1}, δ^{p2} and δ^{p3} associate to v the same symbol in Σ or the symbol ϕ. As an immediate consequence, theorem 3.7 can be easily extended to *f-independent* drawn symbolic pictures languages, and the following complexity result can be stated.

Corollary 3.3 *The membership problem for context-free three-way drawn symbolic picture languages generated by f-independent drawn symbolic grammars* $\boldsymbol{G} = <G_{\Sigma\cup\Pi}$, *dspic*$_f$*()*$>$ *can be decided deterministically in polinomial time.*

3.3 Context-free k-retreat Bounded Picture Languages

Other interesting subclasses have been introduced to solve the intractability of major decision problems. In particular, given an integer $k\geq 0$ and a $\Sigma\cup\Pi$-word x, x is *k-retreat-bounded* if it draws a picture in such a manner that the maximum distance of **l** moves, ignoring **u** and **d** moves, from a rightmost point of any partially drawn symbolic picture is bounded by k [43]. Let us observe that this subclass is an extension of three-way picture languages. By extending the results given in [43] for Π-grammar, it can be easily proved that it is decidable whether or not a context-free $\Sigma\cup\Pi$-language L is a *k-retreat-bounded language*.

In the following we show that the conditions on merging functions provided in section 3.1 allow us to extend also the membership result provided in [43] for context-free k-retreat bounded Π-language describing a nonvertical-stripe drawn picture language. In particular we present a recognition algorithm based on the dynamic-programming working in polinomial time

Let μ, d$_1$, d$_2$ be real numbers such that $\mu\neq\infty$, d$_1$<d$_2$ and M=(Q, ($\Pi\cup\{\epsilon\}$)Σ^*, Γ, γ, q_0, Z_0, q_f) be a *k-retreat-bounded* PDA such that *dspic*$_f$(M) is a drawn symbolic (μ, d$_1$, d$_2$)-stripe picture language. The transition function is defined as γ:Q$\times$($\Pi\cup\{\epsilon\}$)Σ^*)$\times\Gamma\rightarrow 2^{Q\times\Gamma}$,

and we shall assume that M either pushes or pops one symbol in each move and it accepts by entering the unique final state q_f with the empty stack, after scanning the input word. Let us observe that it is decidable whether or not M is a *k-retreat-bounded* PDA (by extending theorem 3.1 in [43]) and whether or not $dspic_f$(M) is a drawn symbolic (μ, d_1, d_2)-stripe picture language (see theorem 3.2 in section 3.1).

Let dsp=$<<p$, (0,0), $e>$, Σ, δ^p $>$ be a drawn symbolic picture given as input and $M_1^{(\mu,d_1,d_2)}$=$\{\{v,v'\}\in M_1 |\ v,\ v' \in M_0^{(\mu,d_1,d_2)}\}$. For $v,v' \in$W($<p$, (0,0), $e>$), let U(v, v')= ${M_1}^{(\mu,d_1,d_2)} \cap {M_1}^{(\infty,x(v)-k,x(v')+k)}$ and define: J(v,v')=U(v,v')-U(v,v)-U(v',v').

A drawn symbolic picture dsp'=$<< p'$, s', $e' >$, Σ, $\delta^{p'}$ $>$ is dsp-valid if $p' \subseteq p$, $\delta^{p'}$(v)= δ^p(v) for each $v \in$W($< p'$, s', $e' >$) and $p' \cap$J(s',e')= $p \cap$J(s',e'). For $s,t \in$Q and A$\in$ Γ, (dsp', r, A, t) is a dsp-valid tuple if dsp' is a dsp-valid drawn symbolic picture and M can go from state r with A alone in its stack to state t, emptying the stack and consuming a $\Sigma \cup \Pi$-word x from the input tape with $dspic_f(x)$=dsp' .

Let X be the set of dsp-valid tuples. Clearly, X is a finite set and it is easy to see that $dsp \in dspic_f$(M) if and only if (dsp, q_0, Z_0, q_f)$\in$X. As a consequence, in order to test if $dsp \in dspic_f$(M) we need only to construct the matrix X. We can use a dynamic programming method defined by the following function Y_M such that, for all $\tau \in$X (with τ=($<< p'$, s', $e' >$, Σ, $\delta^{p'}$$>$, r, A, t) and E$\in 2^X$, $Y_M(\tau$,E) consists of all dsp-valid $\underline{\tau}$ tuples defined as:

1. $\underline{\tau}$=($<<\{v,\ s'\}\cup p' \cup p''$, v, $v' >$, Σ, $\delta^p$$>$, r', B, t') if (r, AB)$\in\gamma(r',\ \sigma$, B), $dspic_f(\sigma)$=$<<\{v$, $s'\}$, v, $s' >$, Σ, $\delta' >$, ($<< p''$, e', $v' >$, Σ, $\delta^{p''}$ $>$, t, B, t')$\in$E and $<<\{v$, $s'\}$, v, $s' >$, Σ, $\delta^{p'} > \bullet_f << p'$, s', $e' >$, Σ, $\delta^{p'} > \bullet_f << p''$, e', $v' >$, Σ, $\delta^{p''}$ $>$=$<<\{v$, $s'\}\cup p' \cup p''$, v, $v' >$, Σ, δ^p $>$; and

2. $\underline{\tau}$=($<<\{v,\ v'\}\cup p'' \cup p'$, v, $e' >$, Σ, δ^p $>$, r', A, t) if (t', BA)$\in\gamma(r',\ \sigma$, A), $dspic_f(\sigma)$=$<<\{v$, $v'\}$, v, $v' >$, Σ, $\delta' >$, ($<< p''$, v', $s' >$, Σ, $\delta^{p''}$ $>$, t', B, r)$\in$E and $<<\{v$, $v'\}$, v, $v' >$, Σ, $\delta' > \bullet_f << p''$, v', $s' >$, Σ, $\delta^{p''} > \bullet_f << p'$, s', $e' >$, Σ, $\delta^{p'}$ $>$=$<<\{v$, $s'\}\cup p'' \cup p'$, v, $e' >$, Σ, δ^p $>$.

The Algorithm for the membership test depicted in Fig. 3.13 is an adaptation of the Algorithm provided by Kim for *k-retreat-bounded* drawn picture which has been modified to work according to the above definitions of matrix X and function Y_M.

Algorithm 3.3. Membership test for k-retreat bounded drawn symbolic picture languages

INPUT: A k-retreat-bounded drawn symbolic picture *dsp* and

a PDA $M=(Q, (\Pi \cup \{\varepsilon\})\Sigma^*, \Gamma, \delta, q_0, Z_0, q_f)$.

OUTPUT: *yes* if $dsp \in dspic_f(M)$, *no* otherwise.

1. If *dsp* is not a (μ, d_1, d_2)-stripe drawn symbolic picture, then output "no"
2. add all *dsp*-valid tuples corresponding to pop transitions of M into an initially empty set X and into an initially empty list T;
3. **while** T is not empty **do**

   remove an element τ from T,

   compute $Y= Y_M(\tau,X)$, and

   add each $\underline{\tau} \in Y-X$ to X
4. **if** $(dsp, q_0, Z_0, q_f) \in X$ **then** output "yes" **else** output "no".

Figure 3.13: *The algorithm for the membership test for k-retreat.*

The proof of the method is basically identical to the proof for the Theorem 3.7 since the algorithm used is structurally identical to the one used for the membership problem of three-way context-free drawn symbolic pictures. We have to consider the tuples (dsp', r, A, t) as $(m,dsp'))$-*valid tuples* if M can go from state r with A alone in its stack to state t, emptying the stack and consusming a $\Sigma \cup \Pi$-word w with $dspic_f(w)=dsp$, after exactly m moves. Thus, it is sufficient to show that each (m,dsp)-*valid tuple* can be added to X by the above algorithm. It can be proved by induction on m as done for three-way picture in the previous section.

Moreover, the running time of Algorithm 3.3 is $O(|dsp|^4)$. As a matter of fact, let *dsp* the k-retreat-bounded drawn symbolic picture to be recognized. Step 1 and step 2 take $O(|dsp|^2)$ time. $|X|=O(|dsp|^2)$ because $U(v,v)$ and $U(v',v')$ are of the constant size and the number of distinct *dsp-valid* drawn symbolic pictures of the form $<< p', v,v' >, \Sigma, \delta>$ are O(1). Therefore step 3 is iterated $O(|dsp|^2)$ times. $Y_M(\tau,X)$ can be computed in $O(|dsp|^2)$ time since the number of tuple to be added to X is $O(|dsp|)$ and the union operation take $O(|dsp|)$ time. Therefore step 3 takes $O(|dsp|^4)$ time.

Now, we are ready to provide the following theorem.

Theorem 3.8 *Let (Σ_ϕ, f) be an abelian monoid. The membership problem for k-retreat-bounded context-free drawn symbolic picture languages generated by drawn symbolic gram-*

mars $\boldsymbol{G}$*=*$<G_{\Sigma\cup\Pi}$*, dspic*$_f$*()*$>$ *can be decided deterministically in polinomial time.*

Let us observe that if **G** is *f-independent* then each point in the drawn symbolic pictures has associated at most one symbol in Σ and/or a certain number of occurrences of ϕ. Thus the associative and commutative hypothesis are no longer needed for the function Υ_M in order to determine the valid tuples $\underline{\tau}$.

As an immediate consequence, theorem 3.8 can be easily extended to *f-independent* drawn symbolic pictures languages, and the following complexity result can be stated.

Corollary 3.4 *The membership problem for k-retreat-bounded context-free drawn symbolic picture languages generated by f-independent drawn symbolic grammars* $<G_{\Sigma\cup\Pi}$*, dspic*$_f$*()*$>$ *can be decided deterministically in in polinomial time.*

Chapter 4

Symbolic Picture Languages

In this chapter we present the notion of symbolic picture and investigate the properties of symbolic picture languages. Informally, symbolic pictures can be considered as a generalization of drawn symbolic pictures by using the notion of invisible lines introduced in [36, 65]. Even though this aspect may appear trivial, it introduces another degree of freedom in picture string descriptions which determines different complexity properties. As a matter of fact, the string description language of a symbolic picture turns out to be a context-sensitive language. Simbolic picture languages strictly include the class of string languages and allow us to describe two-dimensional languages such as arithmetic expressions and colored bitmaps.

Informally, a symbolic picture consists of a set of symbols, each of them associated with a point of the plane. Such notion is formalized by a set P of points, a function δ labelling the positions with symbols from an alphabet Σ, and the definition of a a start point and an end point. More formally:

Definition 4.1 *A symbolic picture is a quintuple sp* $= <\mathrm{P}, \mathrm{s}, \mathrm{e}, \Sigma, \delta>$, *where* $\mathrm{P} \in M_0$ *is a set of points,* Σ *is an alphabet of symbols,* δ *is a function* δ: $\mathrm{P} \rightarrow \Sigma_\phi$, s = *(0,0) is the start point, and* e *is the end point. The start point and end point belong to* P *if sp is nonempty. An empty symbolic picture is denoted by* $< \emptyset, (0,0), (0,0), \Sigma, \delta >$, *where* δ *is only defined in (0,0) and* $\delta(0,0)=\phi$.

Example 4.1 *In Fig. 4.1 the symbolic picture sp* $= < \{(0,0), (1,0), (1,1), (2,2)\}, (0,0), (2,2), \Sigma, \delta >$ *is depicted, where* $\Sigma = \{a, b, c\}$, *and* δ *is the function such that* $\delta((0,0)) = a$, $\delta((1,0)) = a$, $\delta((1,1)) = c$, $\delta((2,1)) = b$. *The start point is marked with a circle and the end point with a square.*

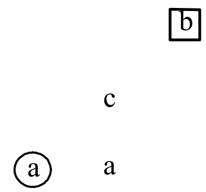

Figure 4.1: *A symbolic picture*

Example 4.2 *The arithmetic expression in Fig. 4.2 can be described by the symbolic picture* sp $= < \{(0,0), (1,0), (2,1), (2,0), (2,-1), (3,0), (4,1), (4,0), (4,-1)\}, (0,0), \Sigma, \delta >$, *where* $\Sigma = \{a, b, c, e, f, +, -\}$, *and* δ *is the function such that* $\delta(0,0) = a$, $\delta(1,0) = +$, $\delta(2,1) = b$, $\delta(2,0) =$ -, $\delta(2,-1) = c$, $\delta(3,0) = +$, $\delta(4,1) = e$, $\delta(4,0) =$ -, $\delta(4,-1) = f$.

$$a + \frac{b}{c} + \frac{e}{f}$$

Figure 4.2: *An arithmetic expression*

A *symbolic picture language* is a set of symbolic pictures. It can be easily shown that symbolic picture languages strictly include the class of string languages. Moreover, they allow us to describe two-dimensional languages such as arithmetic expressions and colored bitmaps.

In the following we will provide a string-based representation of symbolic pictures and the notion of symbolic picture grammar (section 4.1). From a theoretical point of view, in section 4.2 we will identify the conditions under which the set of symbolic pictures is a finitely generated monoid. Section 4.3 will be devoted to the analysis of some decidability problem for symbolic pictures. Finally, section 4.4 will present a classification of symbolic picture grammars.

4.1 Symbolic Picture Description

In order to give the string descriptions of symbolic pictures, we recall the notion of invisible lines which was introduced in [36, 65]. These lines correspond to moves which are executed by raising the pen. The set of invisible moves can be denoted by $\Pi_b = \{r_b, l_b, u_b, d_b\}$. According to this definition each move m in $\Pi_b = \{r_b, l_b, u_b, d_b\}$ is performed with the pen up so that no line is actually drawn on the plane. The description of a symbolic picture can be given by describing a walk (with the pen up) through the picture. Such a walk starts from the start point and touches at least once each point in the symbolic picture. The walk between two non-contiguous points is described by exploiting invisible symbols. Thus, a symbolic picture can be described by a $\Sigma \cup \Pi_b$-word w. As an example, the $\Sigma \cup \Pi_b$-word ‘$a\mathbf{r}_b a\mathbf{u}_b c\mathbf{r}_b\mathbf{u}_b b$’ describes the picture depicted in Fig. 4.1. In other words a string-based representation of a symbolic picture *sp* can be given by applying the following steps:

1. given a drawn picture *dp* whose vertices ”touch” all the symbols of the desired *sp*,

2. consider a drawn symbolic picture *dsp* that could be obtained from the ”superimposition” of *sp* with *dp* and let w be a $\Sigma \cup \Pi$-word representing *dsp*,

3. substitute each symbol from Π in w with the corresponding symbol from Π_b=$\{r_b, l_b, u_b, d_b\}$ to obtain a $\Sigma \cup \Pi_b$-word.

As in the case of drawn symbolic picture, a $\Sigma \cup \Pi_b$-word can describes more than one picture. As an example, the $\Sigma \cup \Pi_b$-word w=”a$\mathbf{r}_b$b$\mathbf{u}_b$c$\mathbf{d}_b$g” can specify the three pictures depicted in Fig. 4.3. Indeed, w specifies that two symbols b and g might be associated to the point (1,0). For the definition of symbolic picture only one symbol can be associated to each point, thus it is necessary to establish which symbol must be associated to each point. We can still use a merging function f as in the case of the drawn symbolic pictures. So, if the function f is such that $f(b,\ g)$=b then w describes the symbolic picture depicted in Fig. 4.3(i), if $f(b,\ g)$=g then w describes the symbolic picture of Fig. 4.3(ii), otherwise if $f(b,\ g)$=e different from b and g, then w describes the symbolic picture of Fig. 4.3(iii).

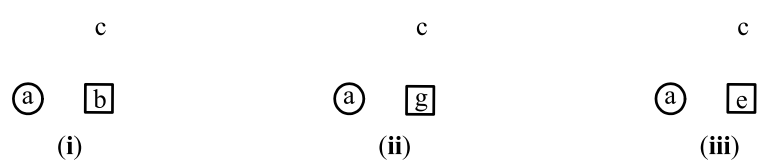

Figure 4.3: *Two drawn symbolic pictures described by w=* $a\boldsymbol{r}_b b\boldsymbol{u}_b c\boldsymbol{d}_b g$

The symbolic picture described by a $\Sigma \cup \Pi_b$-word w is denoted by $spic_f(w)$ and it is defined inductively in agreement with definition 4.2. Such function has as parameter a merging function on Σ_ϕ and ensuring that only one symbol is associated to a given position.

Definition 4.2 *Let w be a $\Sigma \cup \Pi_b$-word and f a merging function on Σ_ϕ. The symbolic picture described by* w *w.r.t. the merging function f, denoted by $spic_f(w)$, is defined inductively as follows:*

- *if $w = \epsilon$, then $spic_f(w)$ is the empty drawn symbolic picture;*
- *if $w = w'\tau$ for some $w' \in (\Sigma \cup \Pi_b)^*$, with $spic_f(w') = <P, s, e, \Sigma, \delta>$,*

$$spic_f(w) = \begin{cases} < P, s, e, \Sigma, \underline{\delta}_1 > & \text{if } \tau \in \Sigma \\ < P \cup \{\tau(e)\}, s, \tau(e), \Sigma, \underline{\delta}_2 > & \text{if } \tau \in \Pi_b \end{cases}$$

where $\underline{\delta}_1$: $P \to \Sigma_\phi$ is the labeling function such that:

$$\underline{\delta}_1(v) = \begin{cases} f(\delta(v), \tau) & \text{if } v{=}e \\ \delta(v) & \text{otherwise} \end{cases}$$

where $\underline{\delta}_2$: $P \cup \{\tau(e)\} \to \Sigma_\phi$ is the labeling function such that:

$$\underline{\delta}_2(v) = \begin{cases} \phi & \text{if } v \notin P \\ \delta(v) & \text{otherwise} \end{cases}$$

Informally, $spic_f(w)$ constructs a drawn symbolic picture by scanning the string w from left to right similarly to the construction of a drawn symbolic picture. The initial position in the plane is set to (0,0) and a pointer p points to the first symbol in w. Whenever p points to a symbol $\sigma \in \Sigma$, and ρ is the symbol associated to the current position in the plane then $f(\sigma, \rho)$ is written on that position, and p is moved to the next symbol in w. Whenever p points to a symbol $\pi_b \in \Pi_b$, then the current position on the plane is updated in agreement with π_b and ϕ is associated to that position if it is new. Then, as usual p points to the next symbol in w.

The definition of $spic_f()$ can be easily extended to the languages of $\Sigma \cup \Pi_b$-words in a natural manner, i.e. given a $\Sigma \cup \Pi_b$-language S, $spic_f(\mathrm{S}) = \{spic_f(w) \mid w \in \mathrm{S}\}$.

The set of words which describe a symbolic picture sp, defined as:

$$spdes_f(sp) = \{w \in \Sigma \cup \Pi_b \mid spic_f(w) = sp\}$$

is the description language of sp.

Informally, symbolic picture languages can be considered as drawn symbolic picture languages where the spatial relations are not visualized. Even though this aspect may appear trivial, it introduces another degree of freedom in picture string descriptions which determines different complexity properties. As a matter of fact, the string description language of a symbolic picture is a context-sensitive language. In the following we outline the proof of this result. More details can be found in [17]. The proof of context-sensitiveness is based on the concept of invisible path.

Definition 4.3 *An invisible path between two points* p_1 *and* p_2 *at distance* $|p_2 - p_1| = (i, j)$*, is a* $\Pi_b \cup \{\epsilon\}$*-word* w *such that* $|\#r_b(w) - \#l_b(w)| = i$ *and* $|\#u_b(w) - \#d_b(w)| = j$*. The set of all the invisible paths from* p_1 *to* p_2 *is denoted by* $I_{p_1 > p_2}$*.*

As an example, let p_1 and p_2 be two points at distance $|p_2 - p_1| = (1, 1)$, an invisible path in $I_{p_1 > p_2}$ is the $\Pi_b \cup \{\epsilon\}$-word $\mathbf{d}_b\mathbf{r}_b\mathbf{u}_b\mathbf{u}_b$. It can be noted that, for any given pair of points, the set of all the invisible paths on the set of moves $\Pi_b = \{r_b, d_b, l_b, u_b\}$ is a context-sensitive language. An intuition of this is obtained by noticing that an invisible path between two points at distance (i,j) is described by a $\Pi_b \cup \{\epsilon\}$-word containing $m + i$ occurrences of r_b, $n + j$ occurrences of u_b, m occurrences of l_b, and n occurrences of d_b. Any automaton should then remember both m and n for recognizing an invisible path. It is not difficult to build a context-sensitive grammar for such paths.

Theorem 4.1 *Let* $sp = < P, s, e, \Sigma, \delta >$ *be a symbolic picture. Then, spdes(sp) is a context-sensitive* $\Sigma \cup \Pi_b$*-language.*

On the other hand it is possible to consider subsets of string description languages that are regular languages. In particular, the *k-description language* of a symbolic picture is a regular description language. It consists of a subset of the description language obtained by allowing to describe only a limited number of points which does not belong to the given symbolic picture. In order to give the formal definition of k-description language, we introduce the following notation. Let w_b be a $\Sigma \cup \Pi_b$-word. We say that a $\Sigma \cup \Pi$-word w is such that $w =_b w_b$ if w is obtained from w_b by replacing any symbol $\pi_{b_i} \in \Pi_b$ with the corresponding element $\pi \in \Pi$.

Definition 4.4 *Let* $sp =< P, s, e, \Sigma, \delta >$ *be a symbolic picture and* k *a natural number. The* k-description language *of sp is the set:*

$$spdes^k(sp) = \{w_b \in \Sigma \cup \Pi_b | spic_f(w_b) = sp \textit{ and let } w =_b w_b, \textit{ with } dspic_f(w) =< sc, \Sigma, \delta > \textit{ and } |W(sc) - P| \leq k\}$$

In other words, $w_b \in spdes^k(sp)$ if the $\Sigma \cup \Pi - word\ w =_b w_b$, with $dspic_f(w) =< sc, \Sigma, \delta >$ is such that $|\{v \in W(sc)|\delta_\phi(v) = \phi\}| \leq k$.

Theorem 4.2 *Let sp=(P, s, e, Σ, δ) be a symbolic picture, and k be a natural number. Then the set $spdes^k(sp)$ is a regular $\Sigma \cup \Pi_b$-language.*

Proof. *Let SS^k_{sp} be the set of the drawn symbolic pictures dsp=<sc, $\Sigma, \delta >$ such that P$\cup\{s\}\subset W(sc)$ for sp and $\delta(v) = \phi$ for not more than k vertices $v \in W(sc)$. It is easily seen that the set is finite for any given natural number k and that $spdes^k(sp) = \bigcup_{dsp\in SS^k_{sp}} dspdes(dsp)$ once each move π is substituted with π_b. Since each dspdes(dsp) is a regular language and the regular languages are closed under finite union, then $spdes^k(sp)$ is regular language.*

Similarly to the case of drawn symbolic picture, the definition of a symbolic picture grammar can be given.

Definition 4.5 *A* symbolic picture grammar *(*SP grammar*, for short)* **G** *is a pair <* $G_{\Sigma\cup\Pi_b}$, *$spic_f$()>, where* $G_{\Sigma\cup\Pi_b}$ *is a $\Sigma \cup \Pi_b$-grammar and $spic_f$() is the function that translates a $\Sigma \cup \Pi_b$-word into a symbolic picture in agreement with a merging function f. Given an SP grammar* **G** *= <*$G_{\Sigma\cup\Pi_b}$*, $spic_f$()>, the SP language* **L** *generated by* **G***, denoted by* **L***(***G***), is:*

$$\mathbf{L}(\mathbf{G}) = spic_f(L(G_{\Sigma\cup\Pi_b})) = \{spic_f(x)|x \in L(G_{\Sigma\cup\Pi_b})\}.$$

In the following example we provide a symbolic picture grammar describing a set of text layouts with left justified lines. The whole text is ended with an end-of-text marker.

Example 4.3 *Let* **TLayout** *= << T,* Π_b, N, P, S>, *$spic_f$()> be an SP grammar with:*

S = {TEXT}, N = {TEXT, LINE}, T = { char, eot}

P = { TEXT→LINE $\mathbf{d}_b$ TEXT | eot

LINE→char $\mathbf{r}_b$ LINE $\mathbf{l}_b$ | char }.

A sentence from this grammar is:

char $\mathbf{r}_b$ char $\mathbf{r}_b$ char $\mathbf{r}_b$ char $\mathbf{l}_b$ $\mathbf{l}_b$ $\mathbf{l}_b$ $\mathbf{d}_b$ char $\mathbf{d}_b$ char $\mathbf{r}_b$ char $\mathbf{l}_b$ $\mathbf{d}_b$ eot

whose evaluation produces the symbolic picture shown in Fig. 4.4.

char char char char

char

char char

eot

Figure 4.4: *A left justified text*

4.2 The Symbolic Picture Set: a Finitely Generated Monoid

In this section we show that the set of symbolic pictures is a monoid and a finitely generated monoid.

First, we extend to the symbolic pictures the definition of concatenation operator given in section 2.3 for drawn symbolic pictures. Informally, the concatenation of two symbolic pictures can be obtained by overlapping the end point of the first picture with the start point of the second picture. As in the case of the drawn symbolic pictures, the presence of symbols requires a suitable extension of such procedure to determine the symbol associated to each overlapping position. For the definition of symbolic pictures only one symbol can be associated to each point, thus it is necessary to establish which symbol must be associated to each overlapping point. We can still use the a merging function f as in the case of the drawn symbolic pictures.

Thus, the concatenation operator can be defined w.r.t a merging function. In the following, we provide the formal definition.

Definition 4.6 *Let* $q_1 =< P_1, s_1, e_1, \Sigma, \delta_1 >$ *and* $q_2 = < P_2, s_2, e_2, \Sigma, \delta_2 >$ *be two symbolic pictures. Let* m*,* $n \in Z$ *be such that* $t_{m,n}(s_2)=e_1$*, and* f *is a merging function on* Σ_ϕ*. The* concatenation *of* q_1 *and* q_2 *w.r.t.* f*, denoted by* $q_1 \bullet_f q_2$*, is defined as*

$$q_1 \bullet_f q_2 = < P_1 \cup t_{m,n}(P_2), s_1, t_{m,n}(e_2), \Sigma, \delta >$$

where $\delta : (P_1 \cup t_{m,n}(P_2)) \to \Sigma_\phi$ *is such that*

$$\delta(v) = \begin{cases} \delta_1(v) & \textit{if } (v \in P_1) \textit{ and } (v \notin t_{m,n}(P_2)) \\ \delta_2(t_{-m,-n}(v)) & \textit{if } (v \notin P_1) \textit{ and } (v \in t_{m,n}(P_2)) \\ f(\delta_1(v), \delta_2(t_{-m,-n}(v))) & \textit{if } (v \in P_1) \textit{ and } (v \in t_{m,n}(P_2)) \end{cases}$$

It is easy to verify that the concatenation of two symbolic pictures is always defined and it is unique with respect to a given merging function f.

Moreover, by setting the properties of the merging function *f*, it can be easily proved that the set (SP, $\bullet_f$) is a monoid and a finitely generated inverse monoid.

Proposition 4.1 *Let SP be the set of symbolic pictures and* $\bullet_f$ *be the concatenation operator w.r.t. f. If f is associative then (SP,*$\bullet_f$*) is a monoid.*
Proof. *Let us observe that the associative property of concatenation operator directly follows from the associative property of f. Moreover, it is easy to verify that the empty symbolic picture* $< \emptyset$, *(0,0), (0,0),* Σ, $\delta >$ *where* $\delta(0,0) = \phi$ *is the neutral element of the concatenation operator.*

Let us observe that if f is associative then $(\Sigma_\phi,\ f)$ is a monoid. So, we could state Proposition 4.1 as follows:
"Let SP be the set of symbolic pictures on a set Σ_ϕ and $\bullet_f$ be the concatenation operator w.r.t *f*. If $(\Sigma_\phi,\ f)$ is a monoid then (SP,$\bullet_f$) is a monoid."

In order to prove that (SP,$\bullet_f$) is a finitely generated monoid we need the notion of length of symbolic pictures. Given a symbolic picture $sp =< P, s, e, \Sigma, \delta >$, $length(sp) = |P|$.

Proposition 4.2 *Let SP be the set of symbolic pictures on a set* Σ_ϕ *and* $\bullet_f$ *be the concatenation operator w.r.t f such that (*Σ_ϕ*, f) is a monoid. (SP,* $\bullet_f$*) is finitely generated monoid.*
Proof. *Since (*Σ_ϕ*, f) is a monoid, from 4.1 it follows that (SP,* $\bullet_f$*) is a monoid. We prove that (SP,* $\bullet_f$*) is finitely generated by induction on the length of the symbolic pictures. The basis of induction is given by the symbolic pictures of length 0 and 1. In this case the thesis can be easily proved. Indeed, the symbolic pictures* $< \emptyset$, *(0,0), (0,0),* Σ, $\delta >$ *and <v, (0,0), (0,0),* Σ, $\delta >$, *with* $v \in M_0$, *are finitely generated since* Σ *is finite. Let us suppose that the thesis holds for symbolic pictures of length n and prove that it also holds for symbolic pictures of length n+1. It can be easily verified from the definition of the concatenation, that a drawn symbolic picture q, with* $|q| = n + 1$, *can be obtained as the concatenation of two symbolic pictures* q_1 *and* q_2 *such that* q=$q_1 \bullet_f q_2$ *with* $|q_1| \leq n$ *and* $|q_2| \leq n$. *By inductive hypothesis* q_1 *and* q_1 *are finitely generated so q is finitely generated.*

Proposition 4.3 provides the conditions ensuring that monoid (SP,$\bullet_f$) is inverse.

Proposition 4.3 *Let SP be the set of symbolic pictures on a set Σ_ϕ and $\bullet_f$ be the concatenation operator w.r.t f such that (Σ_ϕ, f) is a monoid. (SP, $\bullet_f$) is an inverse monoid if the merging function f is such that for each a$\in \Sigma$ there exists a unique b$\in \Sigma$ with f(a, b)=ϕ and f is not idempotent.*

Proof. *From proposition we have that (SP, $\bullet_f$) is a monoid since (Σ_ϕ, f) is a monoid. In order to prove that (SP, $\bullet_f$) is inverse we show that for each q$\in$(SP, $\bullet_f$) there exists a unique element, denoted q^{-1}, such that $q \bullet_f q^{-1} \bullet_f q = q$ and $q^{-1} \bullet_f q \bullet_f q^{-1} = q^{-1}$. Let $q = < P, s, e, \Sigma, \delta_1 >$ be a nonempty symbolic picture, and let $q^{-1} = < P, e, s, \Sigma, \delta_2 >$ such that for each v$\in$ P f(δ_1(v),δ_2(v))=ϕ. It is worth noting that the hypothesis on merging function f ensure that for each q there exists a unique drawn symbolic picture q^{-1}. Now it is easy to verify that $q \bullet_f q^{-1} \bullet_f q = q$ and $q^{-1} \bullet_f q \bullet_f q^{-1} = q^{-1}$.*

From propositions 4.2 and 4.3, we have the following theorems.

Theorem 4.3 *Let SP be the set of symbolic pictures on a set Σ_ϕ and $\bullet_f$ be the concatenation operator w.r.t f such that (Σ_ϕ, f) is a monoid. (SP, $\bullet_f$) is a finitely generated inverse monoid if the merging function f is such that for each a$\in \Sigma$ there exists a unique b$\in \Sigma$ with f(a, b)=ϕ and f is not idempotent.*

4.3 Decidability Problems for Picture Languages

In this section we provide some decidability results for symbolic picture languages by extending analogous results introduced in chapter 2 for drawn symbolic picture languages. In particular, in section 4.3.1 and 4.3.2 we will analyze respectively the ambiguity problem and the f-independency problem for symbolic picture languages.

4.3.1 The Ambiguity Problem

In a symbolic picture grammar the concept of ambiguity is a little more complex than in a drawn symbolic picture grammar. In this case, we distinguish one more case of visual ambiguity besides the two ones individuated for drawn symbolic pictures.

In fact, in a symbolic picture grammar there are three reasons for which the evaluation of $spic_f()$ on different $\Sigma \cup \Pi_b$-words may produce the same symbolic picture:

1. the $\Sigma \cup \Pi_b$-words describe different drawn symbolic pictures for the same symbolic picture;

2. the $\Sigma \cup \Pi_b$-words describe different walks on the same drawn symbolic picture of the symbolic picture;

3. the $\Sigma \cup \Pi_b$-words describe the same walk on the drawn symbolic picture of the symbolic picture, and the walk intersects itself in a point. Due to the definition of $spic_f()$ the $\Sigma \cup \Pi_b$-words may present different symbols associated to that point yet producing the same picture.

In the first case, we have *visual ambiguity on different draws*, in the second one, *visual ambiguity on the same draw*, in the third one, finally, *visual ambiguity on intersection.*

Example 4.4 *Let* $\mathbf{G}=< G_{\Sigma\cup\Pi_b},\ spic_f()>$ *be an SP grammar where* $G_{\Sigma\cup\Pi_b} = < \{r_b, l_b, u_b, d_b\}, \{a, b, c, k\}, \{S, A, B, C\}, P, S >$, *f is such that f(c,c)=c, f(c,k)=c, and P is the set of productions* $\{S \rightarrow a\mathbf{r}_b cAaB,\ S \rightarrow a\mathbf{r}_b cD,\ A \rightarrow \mathbf{r}_b b\mathbf{d}_b,\ A \rightarrow \mathbf{d}_b b\mathbf{r}_b,\ B \rightarrow \mathbf{l}_b b\mathbf{u}_b C,\ B \rightarrow \mathbf{u}_b b\mathbf{l}_b C,\ C \rightarrow c\mathbf{u}_b a,\ C \rightarrow k\mathbf{u}_b a,\ D \rightarrow \mathbf{u}_b aE,\ E \rightarrow \mathbf{r}_b\mathbf{d}_b b\mathbf{d}_b a\mathbf{l}_b b\}$. *This grammar is an example of symbolic picture grammar that is visually ambiguous on different draws, visually ambiguous on the same draw, and visually ambiguous on intersection. It is visually ambiguous on different draws since there exists two different sentences* $w = a\mathbf{r}_b c\mathbf{r}_b b\mathbf{d}_b a\mathbf{l}_b b\mathbf{u}_b c\mathbf{u}_b a$ *and* $w_1 = a\mathbf{r}_b c\mathbf{u}_b a\mathbf{r}_b\mathbf{d}_b b\mathbf{d}_b a\mathbf{l}_b b$ *generated by* $G_{\Sigma\cup\Pi_b}$ *that are the "descriptions" of different drawn symbolic pictures of the same symbolic picture. It is visually ambiguous on the same draw since there exist two different sentences w and* $w_2 = a\mathbf{r}_b c\mathbf{d}_b b\mathbf{r}_b a\mathbf{u}_b b\mathbf{l}_b c\mathbf{u}_b a$ *generated by* $G_{\Sigma\cup\Pi_b}$ *that are the description of different walks on the drawn symbolic picture corresponding to the symbolic picture. It is visually ambiguous on intersection since there exist two different sentences w and* $w_3 = a\mathbf{r}_b c\mathbf{r}_b b\mathbf{d}_b a\mathbf{l}_b b\mathbf{u}_b k\mathbf{u}_b a$ *generated by* $G_{\Sigma\cup\Pi_b}$ *describing the same walk on the drawn symbolic picture of the symbolic picture intersecting itself in a point in which they have different symbols associated. Fig. 4.5 shows the symbolic picture generated by G which is the result of applying the function* $spic_f()$ *to the strings* w, w_1, w_2 *and* w_3.

In a similar way to the case of drawn symbolic pictures, we can exploit the regularity of the k-description language of a symbolic picture (see theorem 4.2) to prove some as-

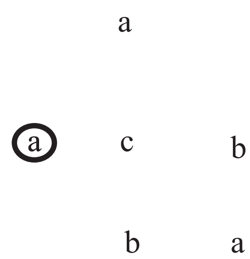

Figure 4.5: *The symbolic picture under the SP grammar of the Example 4.4*

serts about the ambiguity of symbolic picture grammars with respect to a given symbolic picture.

Corollary 4.1 *Let* G *be a* $\Sigma \cup \Pi_b$*-grammar, and* sp *a symbolic picture. The following problems are decidable:*

1. $L(\mathrm{G}) \cap spdes^k(sp) \neq \emptyset$ $\quad$ $(L(\mathrm{G}) \cap Cspdes^k(sp) \neq \emptyset)$
   i.e., $L(\mathrm{G})$ *contains a (consistent) k-description for the symbolic picture sp;*

2. $|L(\mathrm{G}) \cap spdes^k(sp)| < \infty$ $\quad$ $(|L(\mathrm{G}) \cap Cspdes^k(sp)| < \infty)$
   i.e., $L(\mathrm{G})$ *contains a finite number of (consistent) k-descriptions for the symbolic picture sp;*

3. $|L(\mathrm{G}) \cap spdes^k(sp)| = 1$ $\quad$ $|L(\mathrm{G}) \cap Cspdes^k(sp)| = 1)$
   i.e., $L(\mathrm{G})$ *contains only one (consistent) k-description for the symbolic picture sp.*

As an immediate consequence of this Corollary, given a symbolic picture sp and an arbitrary $\Sigma \cup \Pi_b$-grammar G it is possible to decide if sp is described by only one sentence of G.

4.3.2 The *f*-independency Problem

The notion of *f-independency* can be extended to the symbolic picture descriptions in the natural way. Informally, a $\Sigma \cup \Pi_b$-word w is *f-independent* if and only if it specifies at most one symbol in Σ for each point of the described picture. Otherwise we say that w presents a "conflict" or w is *f-dependent*. As an example, the $\Sigma \cup \Pi_b$-word $w = a\mathbf{r}_b b\mathbf{r}_b c\mathbf{l}_b$ is *f-independent*. On the contrary, the $\Sigma \cup \Pi_b$-words $w' = a\mathbf{r}_b b\mathbf{r}_b c\mathbf{l}_b e$ and $w'' = \mathbf{r}aa\mathbf{r}_b c\mathbf{l}_b e$

are *f-dependent* because two symbols (b and e in w', and two occurrences of a in w'') are specified for the same position.

As shown in the following, the formal definition of *f-independent* symbolic picture is similar to the definition of *f-independent* drawn symbolic picture.

Definition 4.7 *Let Φ be the family of merging functions on Σ_ϕ. Given a $\Sigma \cup \Pi_b$-word* w*,* w *is f-independent whenever the following condition holds:*

$$spic_f(w) = spic_g(w),\ \forall f,\ g \in \Phi.$$

A $\Sigma \cup \Pi_b$-language L is f-independent if $\forall$w$\in L$, w is f-independent.

The following characterization provides a condition ensuring the *f-indenpendency* of $\Sigma \cup \Pi_b$-words.

Property 4.1 *A $\Sigma \cup \Pi_b$-word w is f-independent if and only if one of the following conditions holds:*

- *$w = \epsilon$;*
- *$w = w'\tau$ for some f-independent $w' \in (\Sigma \cup \Pi_b)^*$, $spic_f(w') = <P,s,e\ \Sigma,\ \delta>$, and ($\tau \in \Sigma$ and $\delta(e)=\phi$).*

The set of *f-independent* words which describe a symbolic picture *sp*, defined as:

$$Cspdes(sp) = \{w \in \Sigma \cup \Pi_b |\ w \text{ is } f\text{-independent and } spic_f(w) = (sp)\}$$

is the *f-independent* description language of *sp*.

It is easy to prove by extending theorem 4.1 that the f-independent string description language of a symbolic picture is a context-sensitive languages. More formally:

Theorem 4.4 *Let $sp = < P, s, e, \Sigma, \delta >$ be a symbolic picture. Then, Cspdes(sp) is a context-sensitive $\Sigma \cup \Pi_b$-language.*

On the other hand it is possible to consider subsets of f-indepedency description languages that are regular languages: the *f-independency k-description languages*.

Definition 4.8 *Let $sp =< P, s, e, \Sigma, \delta >$ be a symbolic picture and k a natural number. The* f-independency k-description language *of sp is the set:*

$$Cspdes^k(sp) = \{w_b \in spdes^k(sp) | w_b \textit{ is consistent}\}$$

Moreover, we have the following theorem.

Theorem 4.5 *Let sp=(P, s, e,* Σ, δ*) be a symbolic picture, and k be a natural number. Then the set* $Cspdes^k(sp)$ *is a regular* $\Sigma \cup \Pi_b$*-language.*

Proof. *Let* SS^k_{sp} *be the set of the drawn symbolic pictures dsp=<sc,* Σ, δ *> such that* $P \cup \{s\} \subset W(sc)$ *for sp and* $\delta(v) = \phi$ *for not more than k vertices* $v \in W(sc)$*. Then* $Cspdes^k(sp) = \bigcup_{dsp \in SS^k_{sp}} Cdspdes(dsp)$ *once each move* π *is substituted with* π_b *and it is a regular language.*

As a consequence of regularity of the set of f-independency k-description languages we have the following decidability properties.

Corollary 4.2 *Let* G *be a a context-free* $\Sigma \cup \Pi_b$*-grammar, and sp a symbolic picture. The following problems are decidable:*

1. $L(\mathrm{G}) \cap Cspdes^k(sp) \neq \emptyset$
   i.e., $L(\mathrm{G})$ *contains a f-independent k-description for the symbolic picture sp;*

2. $|L(\mathrm{G}) \cap Cspdes^k(sp)| < \infty$
   i.e., $L(\mathrm{G})$ *contains a finite number of f-independent k-descriptions for the symbolic picture sp;*

3. $|L(\mathrm{G}) \cap Cspdes^k(sp)| = 1$
   i.e., $L(\mathrm{G})$ *contains only one f-independent k-description for the symbolic picture sp.*

Proof. *From theorem 4.5 and from the closure property of context-free languages with regular sets we have that the set* $L(\mathrm{G}) \cap Cspdes^k(sp)$ *is a context-free language. Thus, the decidability results follow from the decidability of the emptiness, the finitiness, and the singleton problems for context-free string languages [35].*

Thus, by exploiting theorem 4.2 and corollary 4.2, we have the following result which states that the problem of determining whether or not a symbolic picture has only *f-independent* k-descriptions in a language is decidable for context-free language.

Theorem 4.6 *Let* G *be a* $\Sigma \cup \Pi_b$*-grammar, and sp a symbolic picture. It is decidable whether* $L(\mathrm{G})$ *contains only f-independent k-descriptions for sp.*

Proof. *By Corollary 4.2 we have that it is decidable whether $L(\mathrm{G})$ contains an f-independent k-description for sp. Moreover, we show that it is decidable whether $L(\mathrm{G})$ contains a f-dependent k-description for sp. To this aim, we notice that the f-dependent k-description language of sp, defined as:*

$$NCspdes^k(sp) = spdes^k \text{ - } Cspdes^k(sp)$$

is a regular $\Sigma \cup \Pi$-language (by the closure properties of regular sets). Thus, by the closure properties of context-free languages, the set

$$L(\mathrm{G}) \cap NCspdes^k(sp)$$

is a context-free language. Then, the claim follows from the decidability of the emptiness problem of context-free languages.

The concept of *f-independency* can be extended to symbolic picture grammars in the natural way.

Definition 4.9 *A $\Sigma\cup\Pi_b$-grammar $G_{\Sigma\cup\Pi_b}$ is f-independent if any $\Sigma\cup\Pi_b$-word $w \in L(G_{\Sigma\cup\Pi_b})$ is f-independent. A symbolic picture grammar $\mathbf{G}$ $=< G_{\Sigma\cup\Pi_b}$, $spic_f()>$ is f-independent if $G_{\Sigma\cup\Pi_b}$ is f-independent.*

Now, we prove the decidability of *f-independency* problem for context-free $\Sigma \cup \Pi_b$-grammars, and to this aim we introduce the notion of shift of a $\Sigma \cup \Pi_b$-word, in analogy to the concept of shift for a drawn symbolic picture.

Definition 4.10 *Let w be a $\Sigma\cup\Pi_b$-word and $spic_f(w) =< P, s, e, \Sigma, \delta >$ be the corresponding symbolic picture. The shift of w is defined by shift(w)=(x(e)-x(s),y(e)-y(s)), where for a point v=(m, n), x(v)=m and y(v)=n.*

As an example, given the $\Sigma \cup \Pi_b$-word $w{=}a\mathbf{r}_b b\mathbf{r}_b c\mathbf{l}_b a$, the drawn symbolic picture resulting from $spic_f(w)$ has start point (0,0) and end point (1,0), so *shift*(w)=(1,0). On the contrary, the $\Sigma\cup\Pi_b$-word "$a\mathbf{r}_b b\mathbf{r}_b c\mathbf{l}_b a\mathbf{l}_b c$" has shift (0,0). In other words, a $\Sigma\cup\Pi_b$-word w has shift (0,0) if $\#_{r_b}(w){=}\#_{l_b}(w)$, $\#_{u_b}(w){=}\ \#_{d_b}(w)$.

Theorem 4.7 *Given a context-free $\Sigma \cup \Pi_b$-grammar G, it is decidable whether or not G is f-independent.*

Proof. *G $=<$ $G_{\Sigma\cup\Pi_b}$, $spic_f()>$ is f-independent if any $\Sigma \cup \Pi_b$-word $w \in L(G_{\Sigma\cup\Pi_b})$ is f-independent. Let L^1=suff(pref($L(G_{\Sigma\cup\Pi_b})$)). Let us consider the language: L^2=\{$w \in L^1$ |*

w=$ax'b$ with a,$b\in \Sigma$ and $x' \in (\Sigma \cup \Pi_b)^*\}$. If L^2 is context-free then M=$L^2_{\Pi_b}$ is context-free. Let P_1 be the vectors of Parikh of the words in $L^2_{\Pi_b}$, i.e. P_1=$\Psi(L^2_{\Pi_b})$. Let K be the set of all $\Sigma \cup \Pi_b$-words w such that shift(w)=(0,0) (i.e., $\#_{r_b}(w)$=$\#_{l_b}(w)$, $\#_{u_b}(w)$= $\#_{d_b}(w)$) and P_2 be the set of Parikh vectors of K, i.e. P_2= $\Psi(K)$. Then $L(G_{\Sigma\cup\Pi_b})$ is f-independent if and only if $P_1 \cap P_2$=$\emptyset$. Since $\Psi(X)$ is semilinear for each language X and it is well known that the intersection problem is decidable (see Th.5.2.1 and Th.5.6.1 in [28]) then it is decidable whether or not $P_1 \cap P_2$=$\emptyset$. Hence the theorem holds.

4.4 Notes on the Classification of Symbolic Picture Grammars

Both the drawn symbolic picture and symbolic picture grammars can be based on grammars whose sentences are strings alternating symbols with moves.

More formally, let Σ_ϕ the alphabet that includes the special symbol ϕ. A walk through a drawn symbolic picture (a symbolic picture, resp.), starting from the start point and ending in the end point, can be described by a $\Sigma_\phi \approx \Pi$ word where symbols from Σ_ϕ are alternated with moves in Π (Π_b, resp.).

As an example, the drawn symbolic picture of Fig. 4.6(i) (the symbolic picture of Fig. 4.6(ii), resp.) can be described by the word $a\mathbf{ru}c\mathbf{d}g\mathbf{d}e$ ($a\mathbf{r}_b\mathbf{u}_bc\mathbf{d}_bg\mathbf{d}_be$, resp.).

Figure 4.6: *Two pictures*

In agreement with this approach, a grammar generating drawn symbolic picture descriptions must be able to generate $\Sigma_\phi \approx \Pi$ word (i.e., strings where symbols are alteranted with moves). In the sequel, a $\Sigma_\phi \approx \Pi$-grammar, denoted by $G_{\Sigma_\phi\approx\Pi}$, will be used to generate drawn symbolic picture descriptions and a drawn symbolic picture grammar (symbolic picture grammar, resp.) is denoted by a pair < $G_{\Sigma_\phi\approx\Pi}$, *dspic$_f$()*>.

The Chomsky hierarchy can then be easily extended to these types of grammars. As an example, a DSP grammar $< G_{\Sigma_\phi \approx \Pi}$, *$dspic_f$ ()*$>$ is context-free iff $G_{\Sigma \approx \Pi}$ is context-free. A different type of classification can be devised if we look at a DSP (or SP) grammar as the interleaving of two string grammars: a Σ_ϕ-grammar and a Π-grammar (Π_b-grammar, resp.). In this case, we can characterize 16 types of DSP or SP grammars depending on whether the component grammars are regular, context-free, context-sensitive or type-0 grammars. To this aim we give the following definition:

Definition 4.11 *Let $G =< \Gamma, \Lambda, N, P, S >$ be a $\Gamma \approx \Lambda$-grammar and $A \subseteq (\Gamma \cup \Lambda)$ a subset of the terminal symbols. We define the string grammar $G_A =< A, N, P_A, S >$ as the projection of the grammar* G *on the set* A*, where P_A is the set of productions obtained from the productions in* P *by removing all the terminals not in* A*.*

Example 4.5 *Let $G =< \Sigma_\phi = \{a, b, c, \phi\}$, Π, $N = \{S, A, B\}$, $P = \{^{[1]}S \rightarrow AdB$, $^{[2]}A \rightarrow arA$, $^{[3]}A \rightarrow c$, $^{[4]}B \rightarrow Blb$, $^{[5]}B \rightarrow c\}$, $S>$ be a $\Sigma_\phi \approx \Pi$-grammar. By projecting* G *on the sets Π and Σ_ϕ, respectively, we obtain the grammars G_Π and G_{Σ_ϕ}, where:*

- $G_\Pi =< \Pi = \{r, l, u, d\}$, $N = \{S, A, B\}$, $P = \{^{[1]}S \rightarrow AdB$, $^{[2]}A \rightarrow rA$, $^{[3]}A \rightarrow \epsilon$, $^{[4]}B \rightarrow Bl$, $^{[5]}B \rightarrow \epsilon\}$, $S>$;
- $G_{\Sigma_\phi} =< \Sigma_\phi = \{a, b, c, f\}, N = \{S, A, B\}, P = \{^{[1]}S \rightarrow AB$, $^{[2]}A \rightarrow aA$, $^{[3]}A \rightarrow c$, $^{[4]}B \rightarrow Bb$, $^{[5]}B \rightarrow c\}$, $S>$.

4.4.1 Classification of DSP and SP Grammars

The classification of DSP and SP grammars depends on the traditional Chomsky classification of the two string grammars obtained by projecting their corresponding alternating grammar on the set Π and Σ_ϕ. In this paragraph, without loss of generality, we ignore the difference between the sets Π and Π_b.

It can be shown that if a $\Sigma_\phi \approx \Pi$-grammar is strictly of type i (i= 0, 1, 2, 3) according to the Chomsky hierarchy, then one of its two projections on Σ_ϕ and on Π must be strictly of type i, and the other of type j, with $i \leq j$.

Example 4.6 *Let $G =< \Sigma = \{a, b, c\}$, Π, $N = \{S, A, B\}$, $P = \{S \rightarrow brA\phi, A \rightarrow auB, B \rightarrow brA|\phi d\}$, $S>$ be a context-free $\Sigma_\phi \approx \Pi$-grammar. By projecting G on Σ_ϕ and on Π, we obtain:*

- $G_{\Sigma_\phi} =< \Sigma, N, P_{\Sigma_\phi} = \{S \rightarrow bA\phi, A \rightarrow aB, B \rightarrow bA|\phi\}, S>$.
- $G_\Pi =< \Pi, N, P_\Pi = \{S \rightarrow rA, A \rightarrow uB, B \rightarrow rA|d\}, S>$;

Note that G_{Σ_ϕ} is context-free but G_Π is also regular.

The classification of a string grammar is usually represented in a one-dimensional space. The classification of DSP and SP grammars may be represented in a two-dimensional Cartesian plane whose axes represent the Chomsky classification of the projections on the sets Σ_ϕ and Π, respectively.

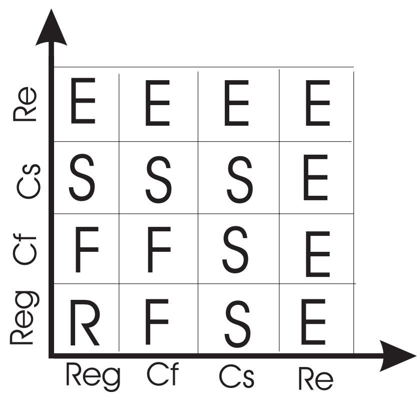

Figure 4.7: *Classification of DSP and SP grammars*

Fig. 4.7 states that the projections of a regular DSP (or SP) grammar on the sets Σ_ϕ and Π (or Π_b) are both regular: so, the regular picture grammars are in the zone 'R' of the plane. The projections of strictly context-free picture grammars may be regular or context-free, with the constrain that at least one of them is strictly context-free: they are disposed in one of the three zones 'F' of the plane, correspondingly to the type of its projections. Analogously, the context-sensitive picture grammars are in the five zones 'S', and the recursive enumerable ones are in the seven zones 'E'. By knowing the type of the two projected grammars, then, it is easy to derive the type of the projecting grammar.

4.4.2 Classification of Picture Languages

A *picture language* is a regular (resp. context-free, context-sensitive, recursive enumerable) picture language if there exists a regular (resp. context-free, context-sensitive, recursive enumerable) picture grammar generating it. The classification of picture languages is obtained analogously to picture grammars ones. It also depends on the following theorem proved in [52] and reported here for sake of completeness.

Theorem 4.8 *The class of context-sensitive drawn picture languages is equal to the class of recursive enumerable ones.*

By this theorem, the axis of classification of projection of a picture languages on Π is split in only three sections. This splitting produces twelve zones in the plane in which we pose regular, context-free, context-sensitive and recursive enumerable picture languages (See Fig. 4.8).

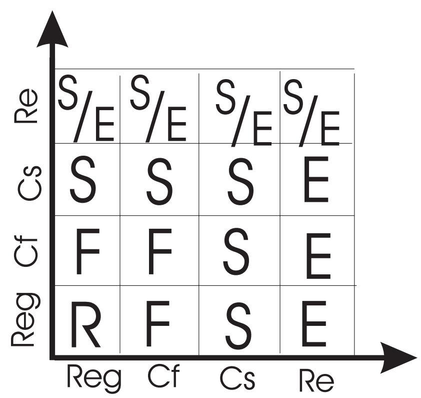

Figure 4.8: *Classification of Picture Grammars*

The projections of the regular picture languages (i.e. the languages generated by regular picture grammars) are both regular: so, the regular picture grammars are in the zone 'R' of the plane. The projection of context-free picture languages may be regular or context-free, with the constrain that at least one of them is context-free: they are disposed in one of three zones 'F' of the plane, correspondingly to the type of its projections. Analogously, the context-sensitive picture languages are in the five zones 'S'. By theorem 4.8, the recursive enumerable ones are in the six zones 'E'.

Chapter 5

Positional Grammars and Symbolic Picture Languages

The main motivation for the investigation of (drawn) symbolic pictures is to provide a theoretical framework to the *Positional Grammars model* as defined in [14].
Positional Grammars are a formalism for the description and recognition of visual languages. They have been successfully used in the context of the *Visual Language Compiler-Compiler* [14] tool (*VLCC*, in short) for the automatic generation of compilers for several visual languages such as two-dimensional arithmetic expressions, electric circuits, combinatorial networks, entity relationship diagrams, semi-structured flowcharts, Petri nets, [14]. The most appealing feature of *VLCC* is that it inherits and extends to the visual field concepts and techniques of traditional compiler generation tools like *YACC* [38]. This is due to the characteristics of the *Positional Grammar* model which represents a natural extension of the context-free grammars. The formalism is suited for visual language definition because it allows to specify new relations between symbols in addition to the string concatenation. This has allowed to extend the LR methodology [2] in order to obtain an efficient *LR*-like parser for positional grammars (*positional LR parser* or *pLR parser* for short [11, 14]).

Let us observe that while the experimentation and the enhancements of VLCC are constantly growing, little analysis has been made of the theoretical aspects of the Positional Grammars formalism. Thus, a goal of this thesis is to reduce this gap by investigating several decidability and complexity properties of some subclasses of Positional Grammars. In particular, the classes of (drawn) symbolic picture grammars formalized in the previous chapters can be seen as subclasses of Positional Grammars. This is due to the close similarity between the description of (drawn) symbolic pictures and the way two-dimensional sentences are generated by a linear positional grammar [14]. This feature turns out to be particular interesting since it allows these subclasses of Positional Grammars to be understood as a natural extension of the picture grammars introduced by Maurer et al. [52] and of the context-free grammars. Due to this formalization the theoretical background established for picture languages and context-free languages can be exploited to get insight into the features of the Positional Grammar formalism. Moreover, we can "apply" the techniques and methodologies developed in the field of visual languages on (drawn) symbolic picture languages. Following this line, in this chapter we start by presenting the Positional Grammars formalism and the subclass of Linear Positional Grammars. Then, we provide an LR parser algorithm for symbolic pictures by exploiting our experiences on pLR parsing.

5.1 The Positional Grammar Formalism

A Positional Grammar $PG = (G, PE)$ as defined in [14] is composed of an attributed string grammar G and a positional evaluator PE that translates positional sentences generated by G into visual sentences. More precisely, the grammar G = (T∪POS, N, S, P) is composed of a set of terminals partitioned into a set of symbols T and a set of relation identifiers POS, a set of nonterminals N, a starting symbol S and a set of productions P of the form:

$$\mathrm{A} \rightarrow \mathrm{x}_1\ \mathrm{R}_1\ \mathrm{x}_2\ \mathrm{R}_2\ ...\ \mathrm{R}_{m-1}\ \mathrm{x}_m,\ \Delta \text{ for } \mathrm{m}\geq 1$$

where $A \in N$, each $x_i \in N \cup \mathrm{T}$, and $R_i \in (POS \times I)^n$ with $1 \leq i \leq \mathrm{m}$, $\mathrm{n} \geq 1$, and I is the set of positive integers. Each symbol x_i has a set of attributes, named syntactic attributes, whose values are used to relate symbols to one another. As an example, if the symbols a and b have Cartesian coordinates as syntactic attributes, then we can write "a **right** b" to indicate that the values of the two pairs of coordinates must differ by (1, 0). Each pair (REL, k) in R_i, also denoted as REL^k, indicates that the relation identifier REL∈POS holds between the precedent symbol x_{i-k} ($k \in I$ and $0 \leq k < i$) and the successive symbol x_{i+1}. (In the case k=0, the superscript k is omitted). As an example, the string "a <**right**> b <**down**^{1}, **up**> c" states that the relations **right**, **down** and **up** hold between the pairs (a, b), (a, c) and (b, c), respectively.

The synthesis rule Δ is used to define the dependence of the syntactic attributes of the nonterminal A on the syntactic attributes of x_1, x_2, ..., x_m.

Let us consider a generic positional grammar PG=((T$\cup$POS, N, S, P), PE), we write $\alpha \Rightarrow \beta$ and say that β is derived from α in one step, if there exist δ, γ, A, η such that $\alpha =$ γAδ , A $\rightarrow$ $\gamma\Delta$ is a production in P and $\beta = \gamma[\eta\ \Delta]\delta$. The square brackets maintain the precedence in the application of the positional evaluator *PE*. We write $\alpha \Rightarrow \beta^*$ and say that β is derived from α, if there exist strings α_0, α_1,..., α_m (m$\geq$0) such that α=$\alpha_0 \Rightarrow \alpha_1 \Rightarrow$ ... $\Rightarrow \alpha_m$=β. The sequence α_0, α_1, ..., α_m is called a *derivation* of β from α. Let x be a grammar object: a *positional sentential form* from x is a string β such that $x \Rightarrow \beta$; *a positional sentence* from x is a positional sentential form from x which does not contain non-terminals; a *pictorial form* (*picture*, respectively) from x is the result of the evaluation of a positional sentential form (positional sentence, respectively) from x by *PE*. As usual, L(PG), the language generated by PG, denotes the set of the pictures from the starting symbol S.

Example 5.1 *The following positional grammar generates structured flow-charts with loops.*

$N = \{F, C, S, R\}$

$T = \{start, halt, begin, end, function, predicate\}$

$POS = \{JOINT\}$

$P = \{1)\ F \rightarrow start\ \textbf{1\_1}\ C\ \textbf{2\_1}\ halt$

$2)\ C \rightarrow begin\ \textbf{2\_1}\ R, \quad \Delta : C_1 = Start_1, C_2 = R_2$

$3)\ R \rightarrow S\ \textbf{2\_1}\ R, \quad \Delta : R_1 = S_1, R_2 = R_2$

$4)\ R \rightarrow end, \quad \Delta : R_1 = end_1, R_2 = end_2$

$5)\ S \rightarrow function, \quad \Delta : S_1 = function_1, S_2 = function_2$

$6)\ S \rightarrow predicate\ < \textbf{2\_1},\ \textbf{1\_2} >\ C, \quad \Delta : S_1 = predicate1 + C_2, S_2 = predicate_3$

$7)\ S \rightarrow predicate\ 2\_1\ C\ < \textbf{3}^1\textbf{\_1},\ \textbf{2\_2} >\ C, \quad \Delta : S_1 = predicate_1, S_2 = C_2 + C_2'\}$

As graphically depicted in Fig. 5.1, the terminal symbols start and halt have one attaching point as syntactic attribute; the terminal symbols begin, function and end have two attaching points as syntactic attributes; and the terminal symbol predicate has three attaching points as syntactic attributes. Observe that in the productions the syntactic attributes are shown as subscripts of the corresponding symbols.

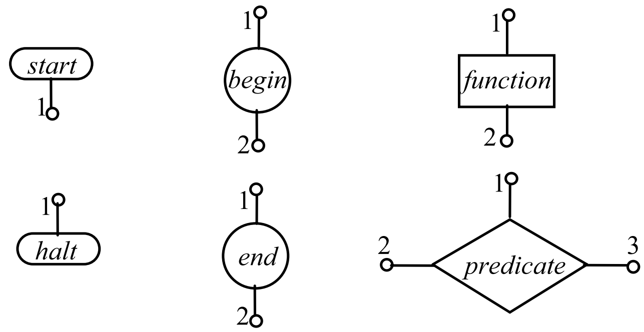

Figure 5.1: *The terminal symbols of the positional grammar of example 5.1*

The production 1) defines a flow-chart F as a compound statement C delimited by the start and halt terminal symbols. In production 2) a compound statement C is defined as a begin symbol connected to a box R. Productions 3) and 4) define a box R as a sequence of statements S ending with the end symbol. Productions 5), 6) and 7) describe a statement S as a simple function, a WHILE loop construct, and an IF THEN ELSE construct, respectively. Note that the starting non-terminal F has no attaching points, while all the other non-terminals have two attaching points. As an example in Fig 5.2 a flowchart sentence in L(G) is depicted.

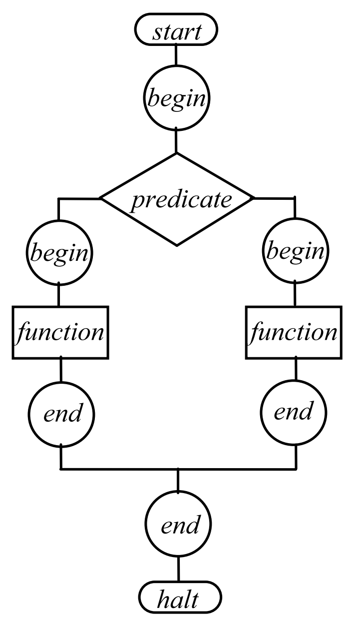

Figure 5.2: *A flowchart sentence generated by the positional grammar of example 5.1*

Linear Positional Grammars

Given the general formulation of Positional Grammars, many subclasses can be identified as described in [15]. In particular, in [8, 9] the class of *Linear Positional Grammars* (LPG, in short) has been characterized as follows:

1. each grammar symbol has only two attributes, namely "head" and "tail",

2. each production has only one type of synthesis rule, namely $\Delta \equiv$(A.head = x_1.head; A.tail = x_m.tail),

3. each sequence R_i in a production contains only one pair (REL, 0), and

4. the definition of each relation *REL* in *POS* between two symbols x and y (x REL y) involves only the attributes "tail" of x and "head" of y.

An example of linear positional grammars is given below.

Example 5.2 *Let* Chain *= ((T∪POS, S S, P), PE) be a Positional grammar with:*
POS = {***llink***},
$T = \{n,\ edge\}$
$P = \{\ S \rightarrow n$ ***llink*** *edge* ***llink*** *S* ***llink*** *edge* ***llink*** n'

Δ_1 *: [S.hnode = n.hnode, S.tnode =* n'*.tnode]*

$S \rightarrow n$ Δ_2 *: [S.hnode = S.tnode = n.hnode = n.tnode] }*

where the attribute values of "tnode" and "hnode" of a node n *are equal to the identifier for* n*. The attributes "hnode" and "tnode" for* S *correspond to the first and last nodes generated by* S*, respectively. The terminal edge has two attributes "head" and "tail" each corresponding to one end of* edge*. The relation* ***llink*** *is defined between a symbol and an* edge *end such that "x* ***llink*** *edge" iff the node identifier* x*.tnode is equal to* edge*.head (i.e.,* x *is connected to edge through its last generated node) and such that "edge* ***llink*** *x" iff* edge*.tail is equal to the node identifier* x*.hnode (i.e.,* x *is connected to* edge *through its first generated* node*).*

A positional sentence for this grammar is:

[n_1 ***llink*** *edge*$_1$ ***llink*** *[*n_2 ***llink*** *edge*$_2$ ***llink*** *[*n_3*]*Δ_2 ***llink*** *edge*$_3$ ***llink*** n_4*]*Δ_1 ***llink*** *edge*$_4$ ***llink*** n_5*]*Δ_1 *whose possible visualization is shown in Fig. 5.3.*

It can be easily seen that, due to the nature of the synthesis rules and to the type of definition of the relation **llink**, the positional sentence above is equivalent to the following:

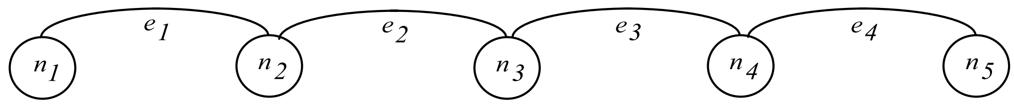

Figure 5.3: *A chain*

n_1 **llink** edge_1 **llink** n_2 **llink** edge_2 **llink** n_3 **llink** edge_3 **llink** n_4 **llink** edge_4 **llink** n_5

Thus, in a *Linear Positional Grammar* the synthesis rules can be omitted from the positional sentences and from each production in the grammar. In this case, the *PE* evaluator can build a visual sentence from the positional sentence by scanning the latter from left to right. An application of the Linear Positional Grammars to the two-dimensional mathematical expression language can be found in [8, 9].

It can be easily proved that a positional sentence from S is always an $(T \cup POS)$-word. Then, if we set $T = \Sigma$, $POS = \Pi_b$, $PE = spic_f()$, a language generated by a linear positional grammar is always a symbolic picture language. Note that not all the (T∪POS)-word are considered since the form of the positional productions allows us to generate only $(T \cup POS)$-words in which symbols in Σ are alternated with symbol relations in Π_b. As a matter of fact, while the immediate objective of the Linear Positional Grammars is to treat real world problems like the parsing of iconic languages, the immediate objective for the study of drawn pictures and (drawn) symbolic pictures is to provide theoretical insight in the theory of formal languages. The scope of this research is to follow both objectives to provide a complete study on the 2D symbolic languages.

Let us observe that, to satisfy real needs, LPG grammars have been studied so far under "ideal" conditions, i.e., they do not generate positional sentences whose evaluation through PE considers a position more than once. Note that this is a more strong simplification with respect to f-independent (drawn) symbolic pictures description grammars. If this condition is relaxed the LPG formalism needs to consider PEs defined in terms of a merging function to manage the overlapping of the symbols.

Although this extension is simple and intuitive, by considering the full scope of LPGs, new issues need to be addressed in terms of a parsing process. Let us consider the positional sentence $w = a\mathbf{r}_b a\mathbf{r}_b e\mathbf{d}_b c\mathbf{l}_b c\mathbf{u}_b e$ generated by the following positional grammar.

$G =<$ N={S, A},

T= {a, b, c, e},

S,

P ={S → a $\mathbf{r}_b$ A $\mathbf{u}_b$ e, A → a $\mathbf{r}_b$ A $\mathbf{d}_b$ c $\mathbf{l}_b$ c, A → e },

POS = {$\mathbf{r}_b$, $\mathbf{l}_b$, $\mathbf{u}_b$, $\mathbf{d}_b$},

PE = $spic_f()$, where f is such that f(a,e)=b >.

Fig. 5.4 depicts the picture obtained by applying the materialization function $spic_f()$ on w. Note that two symbols, a and e, are associated to the same position (1,0) in the plane. By using the function f, $spic_f()$ resolves the conflict and associates the symbol e to (1,0). So, the symbol a does not appear in the generated picture.

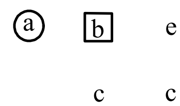

Figure 5.4: *A symbolic picture generated by a linear Positional Grammar*

It is worth noting that a pLR parser as defined in [14] recognizes an input picture by "rebuilding" the positional sentence corresponding to the picture. In the above example, the picture of Fig. 5.4, obtained by applying $spic_f()$, has lost information w.r.t. its corresponding positional sentence w.

Thus, the overlapping of the symbols introduces another degree of complexity in the construction of a parser for the symbolic pictures. In particular, a parser for such pictures has to consider the possibility of having more than one symbol associated to a position and has to embed the functionalities of $spic_f()$ in the parsing process.

In particular, let us consider again the example above. Let us suppose that a parser has processed the symbol a by rebuilding the subsentence $w = a$ and expects next a symbol a in position (1,0). Since it finds a symbol b in position (1,0), it may halt and declare a "parser error" although the picture is syntactically correct! As a consequence, in order to rebuild the corresponding sentence the parser has to consider that b is reachable in f starting from a, i.e. $f(a, e) = b$, and then accept b as a matching symbol. It will then annotate the position (1,0) with the expected symbol a so that when returning in (1,0) expecting for the final symbol e it will have to consider that the input symbol b in position (1,0) is the result of the application of f to the annotated symbol a and the expected e. It can then accept the picture.

Let us remember that, in the general definition of a positional grammar the form of the productions allow us to generate only $(\Sigma \cup \Pi_b)$-words in which symbols in Σ are alternated with symbol relations in Π_b.
By changing a little the definition of positional productions we can permit the presence of two or more consecutive symbols in Π_b and two or more consecutive symbols in Σ in the right hand side of the productions. This allows to generate sentence like the following $w = a\mathbf{r}_b\mathbf{r}_b ce\mathbf{d}_b g\mathbf{l}_b a$, where symbols in Σ are not strictly alternated with symbol relations in Π_b.

It is worth noting that in the pLR parsing process [14] the relations are used to scan the symbols of the input picture. Informally, given a sentential form "$\alpha\ a\ REL\ b\ \beta$" whenever the symbol a has been scanned, the parser has to find in the input picture a symbol b such that "a REL b" holds.
Thus, the new form of the production requires other issues to be investigated in order to extend the pLR parsing technique as defined in [14] to the symbolic picture languages.
The two consecutive relations $\mathbf{r}_b$ $\mathbf{r}_b$ in w are not alternated by a symbol of Σ and in the corresponding picture this means that the position (1,0) has not a symbol associated. What is then the symbol to be scanned by the parser when the symbol a has been seen and the parser tries to find the next by using the relation $\mathbf{r}$?
This problem can be easily overcome by introducing the notion of invisible symbol in the definition of an LPG. So, the absence of a symbol between two moves in a sentential form means that the invisible symbol has to be considered in the visualized picture.
Moreover, the two symbols c and e in w are not alternated by a move relation. What relation then the parser has to use to retrieve the next symbol to be parsed when the symbol c is scanned?
We know that in this case $spic_f()$ does not change position in the plane and associates $f(e, c)$ to that position. In terms of a scanning process, this action can be considered as the behaviour of a relation that indicates: "no move in the plane".

Summarizing, the main features to be considered to extend the pLR parsing technique to the symbolic picture languages are:
- the use of the invisible symbol during the scanning process;
- the introduction of a "no move relation";
- the use of the function f to establish at each time whether a non matching input symbol

may lead to the acceptation of the sentence;

- the use of a mechanism to remember the symbols annotating the positions of the picture.

In the rest of this chapter, we propose a general LR-like parsing algorithm for symbolic pictures, named in short picLR parser, which is the result of the above investigation. Moreover, we prove that picLR can work also on drawn symbolic pictures by providing a simple modification to the parsing algorithm. To simplify the notation, in the discussion that follows we indicate the moves in Π without the subscript $_b$

5.2 An LR parser for Symbolic Picture Languages

The schema of the picLR parser is the same as for a pLR parser [14] and consists of an input, an output, a driver program, and a parsing table (see Fig. 5.5).

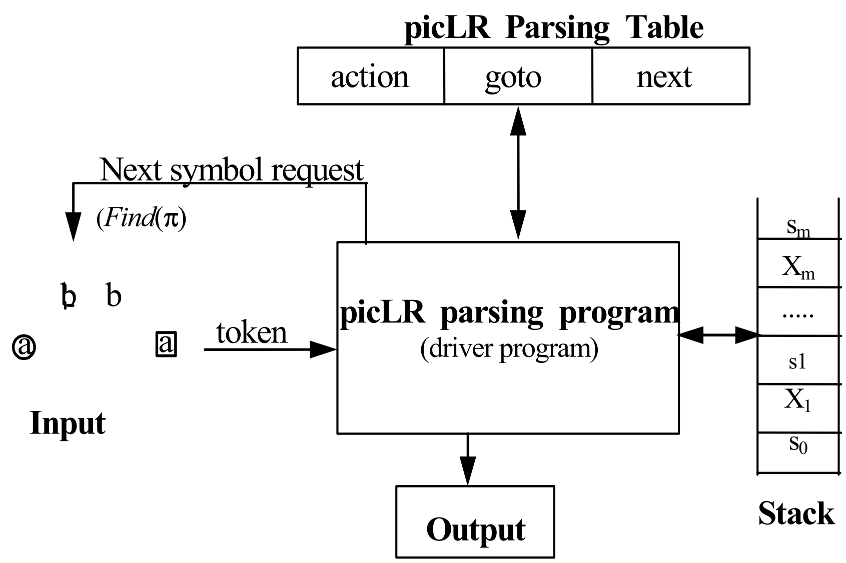

Figure 5.5: *The picLR parser schema*

The main difference with the traditional LR technique is in the scanning of the input picture. As a matter of fact, the input access is no longer sequential but driven by the relations in the alphabet Π contained in the grammar. This is implemented by adding a new column, named next, to the LR parsing table. The column next associates a movement π to each parser state I_k, where π is a relation in $\Pi \cup \{\mathbf{no}_\pi\}$ where $\mathbf{no}_\pi$ denotes the no move relation. Whenever the parser reaches state I_k, the movement π is used to select from the input the next object to be processed by the parser.

The input to the parser is a symbolic picture as produced by a visual editor, like that depicted in Fig. 5.6. A symbol of the picture is internally represented as a pair (b, (x,y)) where b is the symbol name and (x,y) denotes its position in the plane.

Moreover, we denote the *start* and *end symbols* of the symbolic picture p $\mathbf{start}(p)$ and $\mathbf{end}(p)$ respectively. As an example, let us consider the picture of Fig. 5.6, $\mathbf{start}(p)=(a,(0,0))$ is the symbol at the start point and $\mathbf{end}(p)= (c,(2,1))$ is the symbol at the end point. The parser starts analyzing the symbol at the start position.

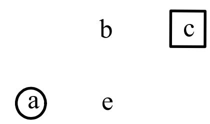

Figure 5.6: *A symbolic picture*

The parser retrieves the objects in the input picture by a *find* operation driven by the relations $\{$**r**,**l**,**u**,**d**, $\mathbf{no}_\pi\}$. Then it makes use of a merging function f to match expected and input symbols.
In the following we represent a merging function f as a labelled graph such that if $f(a,b) = c$ the graph contains two nodes labelled a and c and the edge labelled b leaving from a and entering in c. We will see that f plays a crucial role during the parsing process.

The parsing program reads symbols from the input and uses a stack. An instance of the stack has the general format $s_0g_1s_1g_2s_2....g_ms_m$, where s_m is the stack top; g_i is a grammar symbol and s_i is a generic state of the parser. The parsing algorithm uses the state on the top of the stack and the symbol currently under examination to access a specific entry of the parsing table in order to decide the next action to execute. A picLR parsing table (see Fig. 5.7) is composed of a set of rows and is divided in three main sections: *action*, *goto*, and *next*. The *action* and *goto* sections are similar to the ones used in LR parsing tables for string languages [2], while the *next* section is used by the parser to select the next symbol to be processed.

An action entry has three possible formats, exemplified in the following by considering Fig. 5.7:

1. $action[2, c] = s\ b, 6$: indicating the action "*shift the symbol b onto the stack and go to state 6*"; here c is the input symbol and b is the expected one;

	ACTION					GOTO		NEXT
	b	c	e	ϕ	$	S	B	
0	s b,2					1		SP
1					acc			ANY
2	s b,6	s b,6	s e,3				4	**r**
3	r 3	r 3	r 3	r 3	r 3			$\mathbf{no}_\pi$
4		s c,5						**l**
5	r 1	r 1	r 1	r 1	r 1			$\mathbf{no}_\pi$
6	s b,6	s b,6	s e,3				7	**u**
7	s ϕ,8	s ϕ,8	s ϕ,8	s ϕ,8	s ϕ,8			**r**
8			s e,9					**d**
9	r 2	r 2	r 2	r 2	r 2			$\mathbf{no}_\pi$

Figure 5.7: *A picLR parsing table*

2. $action[3, b] = r\ 3$: indicating the action "*reduce using production rule 3*";

3. $action[1, e] =$ "" : indicating the "error" action.

Due to the possibility that an expected symbol b be different from the input symbol c and that b be reachable in f from c, a shift action on input c needs to push in the stack the expected symbol b and annotate the position of c with b. In order to reach a *success*, the parser will have to visit that position at least once more and the all new expected symbols in that position must reach in f the input symbol c starting from b. In other words when the parsing process ends, all the annotations on the input positions must match the corresponding input symbols.

In its implementation, the picLR parser uses a data structure X such that for each position v of the picture, X(v) contains the symbol currently associated to v. Initially, X(v) contains the invisible symbol ϕ for each position v of the picture. During each shift action "$s\ sym, i$" the parser stores in X(v) the symbol obtained by applying the function f to the expected symbol sym and the previously expected symbol in X(v). At the end of the scanning, the parsing program verifies for each position v if X(v) is the symbol visualized in the input picture.

As an example, let us consider the picture depicted in Fig. 5.9 and suppose that it is obtained by applying the function $spic_f()$ on the sentence w=$b\mathbf{r}b\mathbf{u}e\mathbf{rd}e\mathbf{l}c$ generated by the following positional grammar.

G= $< G_{\Sigma\cup\Pi}$, spic$_f()$> where $G_{\Sigma\cup\Pi}$ = <{b,c,e}, Π, {S, B,}, P, S> and P is

1) S $\rightarrow$ b **r** B **l** c

2) B $\rightarrow$ b **u** B **r** **d** e

3) B $\rightarrow$ e

and Fig. 5.8 shows the full specification of the function f in terms of a labelled graph. From this specification we can deduce that $f(b, c) = c$ since there exists the edge labelled c leaving b and entering in c. The corresponding parsing table is depicted in Fig. 5.7.

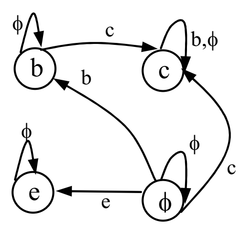

Figure 5.8: *A specification of a merging function by a labelled graph*

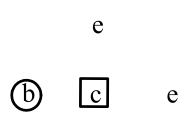

Figure 5.9: *A symbolic picture*

Fig. 5.10 shows the parsing trace of the symbolic picture of Fig. 5.9.

In step 1 the picLR parser pushes on top of the stack the symbol b then state 2. Moreover it stores in X(0,0) the symbol b and by using the entry of next[2]=**r** it retrieves the symbol c from the input picture. In step 2, c is the next symbol to process and *s b,6* is the action to execute. This encodes the fact that $f(b, c) = c$. The picLR parser then stores the symbol b in X(1,0) and continues with the parsing process. Thus, the parser goes to state 6 and uses the entry next[6]=**u** to retrieve the next symbol to be parsed. In step 4, the symbol e is shifted on top of the stack and stored in X(1,1), then the parser goes to state 3. In step 5, the picLR parser accomplishes a reduction action by using production 3). Skipping some steps, in step 8 the symbol c is retrieved and the parser executes action$[4, c]$=*s c,5*. In step 9, the symbol c is shifted on the top of the stack and the symbol $f(c, X(1, 1)) = f(c, b) = c$ is stored in X(1,0), which is the input symbol.

Step	X	Stack	Input	action/goto	Next Move
1	ϕ	0	e b $\boxed{c}$ e	action[0,b]=s b, 2	next[2]=**r**
2	b	0 b 2	e b $\boxed{c}$ e	action[2,c]=s b, 6	next[6]=**u**
3	b b	0 b 2 b 6	e b $\boxed{c}$ e	action[6,e]=s e, 3	next[3]=**no**$_\pi$
4	e b b	0 b 2 b 6 e 3	e b $\boxed{c}$ e	action[3,e]=r 3	next[7]=**r**
5	e b b	0 b 2 b 6 B 7	e b $\boxed{c}$ e	action[7,ϕ]=s ϕ, 8	next[8]=**d**
6	e b b	0 b 2 b 6 B 7 ϕ 8	e b $\boxed{c}$ e	action[8,e]=s e, 9	next[9]=**no**$_\pi$
7	e b b e	0 b 2 b 6 B 7 ϕ 8 e 9	e b $\boxed{c}$ e	action[9,e]=r 2	next[4]=**l**
8	e b b e	0 b 2 B 4	e b $\boxed{c}$ e	action[4,c]=s c, 5	next[5]=**no**$_\pi$
9	e b c e	0 b 2 B 4 c 5	e b $\boxed{c}$ e	action[5,c]=r 1	next[1]=**ANY**
10	e b c e	0 S 1	e b $\boxed{c}$ e	action[1,$]=acc	

Figure 5.10: *The parsing trace of the picture of Fig. 5.9*

In step 10, the parser has rebuilt the positional sentence for the input picture, and for each position v X(v) contains the corresponding input symbol. All the symbols of the input picture have been scanned, the last symbol visited is the symbol at the end point, and then the parser declares the "parsing success".

In the next two sections we give the picLR parsing algorithm and its time complexity. Then, in section 5.5 we show how a picLR parsing table is constructed starting from a $\Sigma \cup \Pi$-grammar. In section 5.6 a discussion on possible parsing table conflicts is provided, and the class of *picLR* $\Sigma \cup \Pi$-grammar is defined. Section 5.7 describes how the picLR parser can work in linear time for subclasses of context-free symbolic picture description languages and how it can be easily adapted to work on drawn symbolic pictures and drawn pictures.

5.3 The picLR Parser: the Proposed Algorithm

In order to provide the picLR parsing program, the function *Find* has to be defined. It uses the stack and the input picture as global data structures and takes its arguments from the column next of the parsing table. So, if next[s]=**r** and $(z,(x,y))$ is the symbol on the stack top then *Find* looks for a symbol $(b,(x+1,y))$ in the input picture such that $z\mathbf{r}b$ holds. Moreover, if next[s]=SP then *Find* returns the symbol in the start position (i.e., the first symbol to be parsed) and if next[s]=ANY then *Find* returns the end-of-input symbol (\$,(-,-)) if all the symbols in the input picture have been marked. Furthermore, LAST is a global variable storing the last symbol scanned and it will be used by the parsing algorithm to verify that the last symbol parsed is equal to the symbol visualized in the end point of the input picture.

function Find(NEXT)
  //NEXT is an entry of the column next
begin
  case NEXT of
  NEXT = SP:
    return the symbol in **start**(p) where p is the input picture
  NEXT = ANY:
    if all the objects have been marked then
      return the end-of-input symbol (\$,(-,-))
    else return "syntax error"; exit
  NEXT = π, where $\pi \in \mathbf{\Pi} \cup \{\mathbf{no}_\pi\}$
    let z be the symbol on the stack top
    if a is the symbol visualized in π(z)
      then
      LAST=(a,π(z))
      return LAST
    else return "syntax error"; exit
  endcase

end

As it can be seen from the algorithm that follows, the picLR parsing program checks the entry next[s] of the parsing table corresponding to the state s on the top of the stack. Then next[s] is given in input to the function *Find* that returns the next terminal (b,(x,y)) to be scanned. In this case, if action[s, b] = "$shift\ sym, s'$" then the symbol (sym,(x,y)) is shifted on the stack top or the input picture is accepted (if action[s, b] = "acc") and b is the end-of-input marker ($,(-,-)). Moreover, the algorithm stores in X(x,y) the symbol obtained by applying the merging function f to sym and X(x,y).

The reduction action[s, b] = "reduce A $\rightarrow$ $\alpha_1\pi_{\mathbf{1}}\ \alpha_{\mathbf{2}}...\pi_{\mathbf{m-1}}\alpha_{\mathbf{m}}$", with $\alpha_i \in \Sigma^*$, is accomplished by popping n elements out of the stack, and pushing A on the stack top, where n is the lenght of the part (on the top) of the stack that matches the symbolic right hand side of the production including the not shown ϕ. If s' is the state on the stack top after popping the n elements, then the next state s'' of the parser is given by the entry goto[s', A].

Algorithm 6.1. The picLR parsing program.

Input: An input symbolic picture q=$< P, start(q), end(q), \Sigma, \delta >$ and a picLR parsing table.

Output: A bottom-up parse of q, if q is syntactically correct, otherwise error.

Method: At the beginning, the initial state s_0 is on the top of the stack.

repeat forever
begin
  let s be the state on the top of the stack
  let (b,(x,y)) the symbol returned by Find(next[s])
  mark the corresponding entry in the input picture if it is not already marked
  if action [s, b] = " shift sym, s' " then
    push $(sym, (x, y))$ then s' on the top of the stack
    set X(x,y) to symbol f(X(x,y),sym)
  else if action [s, b] = reduce A $\rightarrow \alpha_1\pi_{\mathbf{1}}\alpha_{\mathbf{2}}...\pi_{\mathbf{m-1}}\alpha_m$ then
      begin
        Let $n = Detect(\alpha_1\pi_{\mathbf{1}}\alpha_{\mathbf{2}}...\pi_{\mathbf{m-1}}\alpha_{\mathbf{m}})$
        pop n elements out of the stack
        let s' be the state now on the top of the stack

let goto[s', A] = "s''"

push the grammar object A then s'' on the top of the stack

output the production A $\rightarrow \alpha_1\pi_\mathbf{1}\ \alpha_2\ ...\ \pi_\mathbf{m-1}\alpha_\mathbf{m}$

end

else if (action [s, b] = "acc"

and LAST = **end**(q) and all input symbols are marked

and X(v)=δ(v) for each $v\in$W(P)

then *"success"*

else *"syntax error"*

end

Let us observe that whenever $action[s, b]$ = "acc", the parser verifies that:

- all the picture symbols have been parsed;
- the symbols in X correspond to the symbols of the input picture;
- and the symbol LAST is equal to the end point of the picture.

5.4 Parsing Time Complexity

The time complexity of the parsing algorithm depends on the number of objects in the input and on the grammar. Given a grammar G, let n_s be the maximum number of grammar symbols in the right-hand side of a production. Let us consider an input picture q=$<\ P, s, e, \Sigma, \delta\ >$ containing n graphical objects, which correspond to the positions W(P) of the picture, i.e. n=$|P|$. The worst case complexity is achieved for correct input pictures when all the input symbols are visited. At each step, the parser performs a shift or a reduce action. Therefore, the number of shifts and the number of reductions will be O($t_n$$*$$n$), where t_n is the maximum number of times a symbol (i.e. a position) in the input picture can be "traversed". Let us compute separately the time complexity for shift and reduce actions.

To perform a shift action the parsing program must access the input, if any. Let t_F be the time required to perform the function *Find* (on next[s]). As the push operation takes time O(1), that is the time to calculate the value of the symbol position in the plane, the total time complexity to perform a shift action is O(t_F). It includes the time required to insert the symbol retrieved in the data structure X that is a constant.

To reduce a production, the parser has to perform the following steps: (i) calculate the symbol position of the left-hand side non-terminal; (ii) pop the records corresponding to the right-hand side grammar objects from the stack; (iii) push the record corresponding to the left-hand side non-terminal onto the stack. The cost of step (i) is O(1). As the stack pop operation takes time O(1), step (ii) will cost O(n_s). The final push operation (step (iii)) takes time O(1). Therefore, the total time complexity for a reduce action is O(n_s). Moreover, the time required to verify for each $v \in$W(P) if the symbol annotated in X(v) is equal to the symbol visualized δ(v) in the input picture is O(n). Then, the time complexity of the parser is O($t_n$$*$($n$$*$($t_F$+$n_s$))). For a fixed grammar n_s is constant and the parameters t_F may vary from a constant to O(n), depending on the chosen implementation of the input picture.

For a fixed grammar the time complexity is then O($t_n$$*$($n$$*$$n$)) and if the function *Find* works in constant time the expected time complexity reduces to O($t_n$$*$($n$)). Thus the parsing time complexity is strongly related to the maximum number of times each symbol can be traversed.

In section 5.7, subclasses of symbolic picture languages, for which t_n is constant, will be presented. Thus, the picLR parser in this case works in linear time.

5.5 The picLR Parsing Table Construction

In the following, the algorithm used to construct the picLR parsing table is presented. First, the notion of items is given. A picLR(0) *item* of a $\Sigma \cup \Pi$-grammar G is a production from G with a dot at some position in the right-hand side. However, a dot can never be between a relation identifier and the terminal or non-terminal on its right. Therefore, a production A $\rightarrow a_1 a_2 \pi_1 \pi_2 a_3$ yields the following five types of items:

[A $\rightarrow$. $a_1 a_2 \pi_1 \pi_2 a_3$]

[A $\rightarrow a_1$. $a_2 \pi_1 \pi_2 a_3$]

[A $\rightarrow a_1 a_2$. $\pi_1 \pi_2 a_3$]

[A $\rightarrow a_1 a_2 \pi_1$. $\pi_2 a_3$]

[A $\rightarrow a_1 a_2 \pi_1 \pi_2 a_3$.]

Intuitively, an item indicates how much of a production has already been examined during the parsing process and what is yet to come. If G is a $\Sigma \cup \Pi$-grammar with starting non-terminal S, then G', the *augmented* $\Sigma \cup \Pi$-grammar for G, is obtained from G by adding a new starting non-terminal S' and the production $S' \rightarrow$ S. The algorithm for the construction of the sets of picLR(0) items for an augmented grammar is obtained by properly extending the corresponding one in [2].

Algorithm 6.2. Construction of the sets of picLR(0) items.

Input: An augmented $\Sigma \cup \Pi$-grammar G$'$.

Output: The sets of picLR(0) items.

Method: The items are constructed by the main routine ITEMS
which calls the functions CLOSURE and GOTO.

function CLOSURE (I)

// I is a set of picLR(0) items

begin
repeat
for each item [A $\rightarrow \alpha$. π B b] or [A $\rightarrow$. B b] in I
and each production B $\rightarrow \beta$ in G$'$
do add [B $\rightarrow$. β] to I
until no more items can be added to I
return I
end

function GOTO (I, x)

// I is a set of picLR(0) items; x is a grammar symbol

begin
if (x$\neq \epsilon$) then
let J = {item | item = [A $\rightarrow \alpha\ \pi$ x . β],
such that [A $\rightarrow \alpha$. π x β] $\in$ I, with $\pi \in \Pi \cup \{\epsilon\}$
and α does not terminate with a symbol $\pi' \in \Pi$}
$\cup$ {item | item = [A $\rightarrow$ x . β], such that [A $\rightarrow$. x β] $\in$ I};
else let J={item | item = [A $\rightarrow \alpha\ \pi_{\mathbf{1}}$. $\pi_{\mathbf{2}}\ \beta$],
such that [A $\rightarrow \alpha$. $\pi_{\mathbf{1}}\ \pi_{\mathbf{2}}\ \beta$] $\in$ I}
$\cup$ {item | item = [A $\rightarrow \alpha\ \pi_{\mathbf{1}}$.] such that [A $\rightarrow \alpha$. $\pi_{\mathbf{1}}$] $\in$ I}

```
  return CLOSURE (J)
end
procedure ITEMS (G′)
  // G′ is an augmented Σ ∪ Π-grammar
begin
  set C to CLOSURE ([S′ → .S])
  repeat
    for each set of items I in C and each symbol x∈ Σ ∪ {ε} ∪ N
      such that GOTO (I, x) is not empty and not in C do
        add GOTO (I, x) to C;
  until no more sets of items can be added to C
end
```

Similarly to the LR case, the sets of picLR(0) items correspond to the states of a finite automaton for *viable prefixes* [2] where the transitions are determined by the function *GOTO*. Given a set of items I containing an item with a dot before a non-terminal B, the function *CLOSURE* adds to I all the items with B in the left-hand side and the dot preceding the first symbol of the right-hand side. This means that if the non-terminal symbol B is expected next, then any symbol starting a sentential form from B is expected next. Once a grammar symbol x has been seen, the function *GOTO* determines the set of items containing the objects that can be seen next.

In order to define the picLR(0) parsing table we need to define a function $Ann_f()$. This function is defined as the set of all the reachable terminal symbols in f from a given symbol. More formally, for each symbol $b \in \Sigma$:
$Ann_f(b)$={c | there exist $a_1, ..., a_n$ with $n \geq 1$ such that $f(b, a_1) = a_2, f(a_2, a_3) = a_4$,..., $f(a_{n-1}, a_n) = c$}. Note that, as said before, the full specification of f can be given as a labelled graph.

As an example if the merging function f is defined as specified in Fig. 5.11
then
$Ann_f(b) = \{b, a, c, e\}$, $Ann_f(a) = \{a, e\}$, $Ann_f(c) = \{c\}$, $Ann_f(e) = \{e\}$.

The function $Ann_f()$ will be used in the construction of the parsing table to determine the entry of the action part. In particular, if [A $\rightarrow \alpha$. π a β] or [A $\rightarrow$. a β] is in the set of

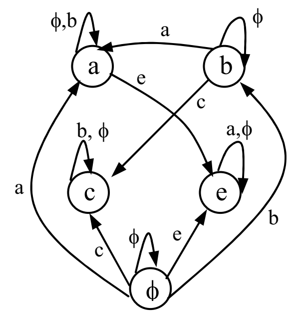

Figure 5.11: *A specification of a merging function f*

items I_i and GOTO $(I_i, a) = I_j$ then for each symbol b in $Ann_f(a)$ we insert "$shift\ a, j$" into action$[i, b]$.

The algorithm for the construction of a picLR parsing table follows.

Algorithm 6.3. Construction of the picLR(0) parsing table.

Input: An augmented $\Sigma \cup \Pi$-grammar G'.

Output: The picLR(0) parsing table for G'.

Method:

1) Construct C = I_0, I_1, ..., I_m, the collection of sets of picLR(0)items
   as described in Algorithm 6.2.

2) State i of the parsing table is constructed from I_i. The entries for state i
   of the parsing table action and next parts are determined as follows:

   <u>*SHIFT ENTRIES*</u>

   a) If [A $\rightarrow \alpha$. π a β] or [A $\rightarrow$. a β] is in I_i, with $\pi \in \Pi \cup \{\epsilon\}$
      and GOTO $(I_i, a) = I_j$ then
         for each symbol $b \in Ann_f(a)$
            insert "shift a, j" into action $[i, b]$. (Here a and b must be terminals)
      If [A $\rightarrow \alpha$. $\pi_{\mathbf{1}}$ $\pi_{\mathbf{2}}$ β] or [A $\rightarrow$. $\pi_{\mathbf{1}}$ $\pi_{\mathbf{2}}$ β] is in I_i
      and GOTO $(I_i, \epsilon) = I_j$ then
         for each symbol $b \in Ann_f(\phi)$
            insert " shift ϕ, j" into action $[i, b]$.

   <u>*REDUCE ENTRIES*</u>

   b) If [A $\rightarrow \alpha$.] is in I_i then
      for each terminal $a \in \Sigma_\phi$

insert "reduce A $\rightarrow \alpha$" into action [i, a].

NEXT AND ACCEPT ENTRIES

c) If [A $\rightarrow \alpha$. π x β] is in I_i, with $\pi \in \Pi$ then insert π in next[i].

if ($x \in \Sigma \cup N$) and ($\pi = \epsilon$) then insert $\mathbf{no}_\pi$ in next[i].

d) if [A $\rightarrow \alpha$.] is in I_i insert $\mathbf{no}_\pi$ in next[i].

e) If [S$'$ $\rightarrow$ .S] is in I_i then insert SP into next[i]. If [S$'$ $\rightarrow$ S.] is in I_i then insert ANY into next[i] and insert "accept" into action [i, \$].

3) The entries for state i and non-terminal X of the goto part are determined as follows: if [A $\rightarrow \alpha$. π X β] or [A $\rightarrow$. X β] is in I_i, with $\pi \in \Pi \cup \{\epsilon\}$ and GOTO (I_i, X) = I_j then insert " j" into goto [i, X].

Example 5.3 *Let G' be the augmented $\Sigma \cup \Pi$-grammar constructed from* $\boldsymbol{G}$= < $G_{\Sigma\cup\Pi}$, *$spic_f$ ()> where $G_{\Sigma\cup\Pi}$ = <{a,b,c,e}, Π, {S, B, C, E}, P, S> and P is*

*1) $S \rightarrow a$ **r** S **l** b*

*2) $S \rightarrow c$ **u** B **r** **d** e*

*3) $B \rightarrow a$ e **r** **d** B*

4) $B \rightarrow c$

and f is specified in Fig. 5.12 and the reachable sets in f are so defined:

$Ann_f(b) = \{b, a, c\}$, $Ann_f(a) = \{a\}$, $Ann_f(c) = \{c\}$, $Ann_f(e) = \{e, a\}$, $Ann_f(\phi) = \{a, b, c, e, \phi\}$.

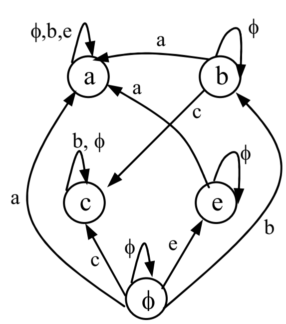

Figure 5.12: *The specification of function f for the $\Sigma \cup \Pi$-grammar of example 5.3*

Fig. 5.13 depicts the set of pLR(0) items for the previous $\Sigma \cup \Pi$-grammar and Fig. 5.14 graphically depicts its corresponding function GOTO obtained by applying algorithm

6.2. Here each edge is labelled also with the corresponding $Ann_f()$ set.

I_0: S' → . S S → . a **r** S **l** b S → . c **u** B **rd** e	I_1: S' → S .	I_2: S → a . **r** S **l** b S → . a **r** S **l** b S → . c **u** B **rd** e	I_3: S → a **r** S . **l** b
I_4: S → a **r** S **l** b .	I_5: S → c . **u** B **r** **d** e B → . e a **r** **d** B B → . c	I_6: S → c **u** B . **rd** e	I_7: S → c **u** B **r** . **d** e
I_8: S → c **u** B **r** **d** e .	I_9: B → c .	I_{10}: B → e . a **r** **d** B	I_{11}: B → e a . **r** **d** B
I_{12}: B → e a **r** . **d** B B → . e a **r** **d** B B → . c	I_{13}: B → e a **r** **d** B .		

Figure 5.13: *The set of pLR(0) items for the $\Sigma \cup \Pi$-grammar of example 5.3*

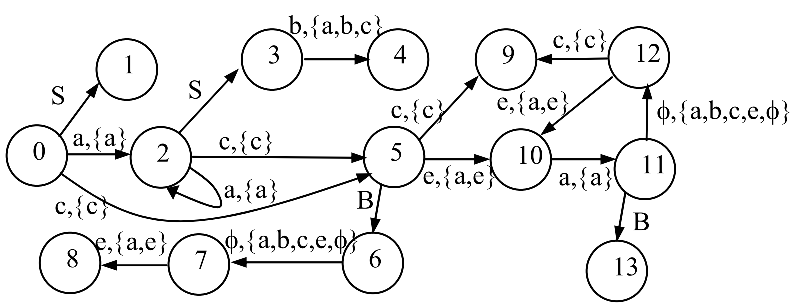

Figure 5.14: *The goto function for the $\Sigma \cup \Pi$-grammar of example 5.3*

Then by using the algorithm 6.3, the picLR parsing table depicted in Fig. 5.15 is obtained.

5.6 The picLR Parsing Table Conflicts

A conflict in a picLR parsing table arises when multiple actions are contained in a single entry in the action, goto or next parts. Besides the usual *shift-reduce* and *reduce-reduce* conflicts, [2], a picLR parsing table may present *shift-shift* and *positional conflicts*. Such ambiguities have been widely analyzed in [2, 11, 14].

	ACTION						GOTO		NEXT
	a	b	c	e	ϕ	$	S	B	
0	s a,2		s c,5				1		SP
1						acc			ANY
2	s a,2		s c,5				3		**r**
3	s b,4	s b,4	s b,4						**l**
4	r 1	r 1	r 1	r 1	r 1	r 1			**no**$_\pi$
5	s e,10		s c,9	s e,10				6	**u**
6	s ϕ,7	s ϕ,7	s ϕ,7	s ϕ,7	s ϕ,7	s ϕ,7			**r**
7	s e,8			s e,8					**d**
8	r 2	r 2	r 2	r 2	r 2	r 2			**no**$_\pi$
9	r 4	r 4	r 4	r 4	r 4	r 4			**no**$_\pi$
10	s a,11								ε
11	s ϕ,12	s ϕ,12	s ϕ,12	s ϕ,12	s ϕ,12	s ϕ,12		13	**r**
12	s e,10		s c,9	s e,10					**d**
13	r 3	r 3	r 3	r 3	r 3	r 3			**no**$_\pi$

Figure 5.15: *The picLR parsing table for the grammar of example 5.3*

A shift-shift conflict occurs whenever multiple shift actions occur in a single entry of the action part. Shift-shift conflicts are generated whenever a set of picLR(0) items I contains at least two dotted terminals a and b and $Ann_f(a) \cap Ann_f(b) \neq \phi$. In fact, in the case $Ann_f(a) \cap Ann_f(b) = \{c, ...\}$ when the parser enters I and the next input symbol is c then it cannot decide whether to shift a or b since c can be reached in f starting form both a and b. Moreover, a positional conflict occurs whenever multiple values π are presented in a single entry of the next column. This conflict is generated whenever a set of pLR(0) items contains two or more items with the dot preceding different relations.

Example 5.4 *Let G' be the augmented $\Sigma \cup \Pi$-grammar constructed from* $\boldsymbol{G}$= < $G_{\Sigma\cup\Pi}$, *spic$_f$()> where $G_{\Sigma\cup\Pi}$ = <$\{a,b,c,e\}$, Π, $\{S, B, C, E\}$, P, S> and P is*

1) $S \rightarrow a\ \mathbf{r}\ S\ \boldsymbol{l}\ b$

2) $S \rightarrow c\ \mathbf{u}\ B\ \boldsymbol{r}\ \boldsymbol{d}\ e$

3) $B \rightarrow a\ e\ \boldsymbol{r}\ \boldsymbol{d}\ B$

4) $B \rightarrow c$

and f is specified in Fig. 5.16 and the reachable sets in f are so defined:

$Ann_f(b) = \{b, a, e\}$, $Ann_f(a) = \{a, e\}$, $Ann_f(c) = \{c, e\}$, $Ann_f(e) = \{e\}$, $Ann_f(\phi) =$

$\{a, b, c, e, \phi\}$.

The set of picLR(0) items is the same as the one in example 5.3 and the function GOTO is depicted in Fig. 5.17.

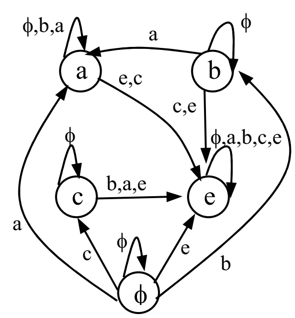

Figure 5.16: *The specification of function f for the* $\Sigma \cup \Pi$*-grammar of example 5.4*

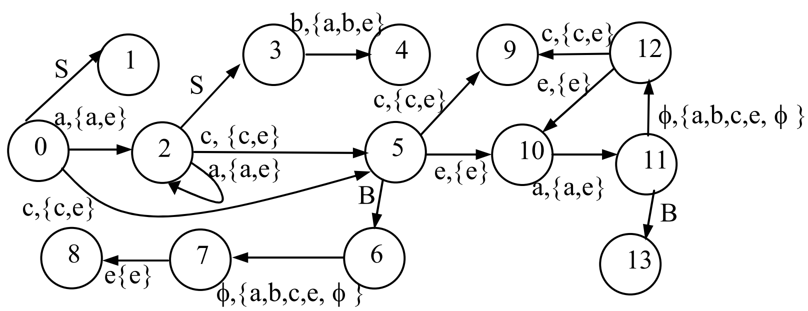

Figure 5.17: *The goto function for the* $\Sigma \cup \Pi$*-grammar of example 5.4*

The corresponding picLR parsing table, obtained by applying algorithm 6.3 and depicted in Fig. 5.18, presents shift/shift conflicts in state 0, 2, 5 and 12. In particular considering state 0, it can be seen that $Ann_f(a) \cap Ann_f(c) = \{a, e\} \cap \{c, e\} = \{e\}$

A $\Sigma \cup \Pi$-grammar for which it is possible to construct a picLR parsing table without conflicts is said to be a *picLR grammar*. The grammar of example 5.3 is a picLR grammar, while the grammar of example 5.4 is not a picLR grammar as it can be verified by analyzing the corresponding parsing table in Figure 5.18.

Let us observe that if a $\Sigma \cup \Pi$-grammar is *f-independent* then the ambiguity of multiple shifts for an entry as those in Fig. 5.18 never occurs since at most one symbol different from ϕ is associated to each position. So, at each step the parsing algorithm retrieves

	ACTION							GOTO		NEXT
	a	b	c	e		ϕ	$	S	B	
0	s a,2		s c,5	s a,2	s c,5			1		SP
1							acc			ANY
2	s a,2		s c,5	s a,2	s c,5			3		**r**
3	s b,4	s b,4		s b,4						**l**
4	r 1	r 1	r 1	r 1		r 1	r 1			**no**$_\pi$
5			s c,9	s e,10	s c,9				6	**u**
6	s ϕ,7	s ϕ,7	s ϕ,7	s ϕ,7		s ϕ,7	s ϕ,7			**r**
7				s e,8						**d**
8	r 2	r 2	r 2	r 2		r 2	r 2			**no**$_\pi$
9	r 4	r 4	r 4	r 4		r 4	r 4			**no**$_\pi$
10	s a,11			s a,11						ε
11	s ϕ,12	s ϕ,12	s ϕ,12	s ϕ,12		s ϕ,12	s ϕ,12			**r**
12			s c,9	s c,9	s e,10				13	**d**
13	r 3	r 3	r 3	r 3		r 3	r 3			**no**$_\pi$

Figure 5.18: *The picLR(0) parsing table for the grammar of Example 5.4*

at most one symbol on which it can shift (i.e., it is the symbol visualized in the input picture).

Moreover, note that in the case of self-avoiding $\Sigma \cup \Pi$-grammars (i.e., they do not generate non self-avoiding $\Sigma \cup \Pi$-words), the corresponding picLR parser can work in linear time if the time required for the function *Find* is constant. As a matter of fact, in this case the factor t_n in the total time complexity $\mathrm{O}(t_n * (n))$ is equal to 1 since loops are not admitted and so each position is traversed only once. Let us remember that in section 2.5 we have proved that it is decidable whether or not a $\Sigma \cup \Pi$-grammar generates non self-avoiding $\Sigma \cup \Pi$-words.

5.7 The picLR Parser for (k,j)-reversal-bounded Symbolic Picture Description Languages

In this section we investigate the class of (k,j)-reversal-bounded $\Sigma \cup \Pi$-languages and we show how the picLR parser algorithm works in linear time in this case. This result can be added to the other decidability results for the membership problem investigated in this thesis and in the field of picture languages theory ([66, 44, 42, 43]).

First, we introduce the class of *(k,j)-reversal-bounded* $\Sigma \cup \Pi$-languages.

Definition 5.1 *Let k and j be two nonnegative integers. A* $\Sigma \cup \Pi$*-word is (k,j)-reversal-bounded if the number of alternating* ***r****'s and* ***l****'s is k and the number of alternating* ***u****'s and* ***d****'s is j.*
A set of (k,j)-reversal-bounded $\Sigma \cup \Pi$*-words is a (k,j)-reversal-bounded* $\Sigma \cup \Pi$*-language.*
G*=<* $G_{\Sigma\cup\Pi}$*,* $spic_f$*()> is a (k,j)-reversal-bounded grammar, if* $G_{\Sigma\cup\Pi}$ *generates only (k,j)-reversal-bounded* $\Sigma \cup \Pi$*-words.*

Let us observe that, given two nonnegative integers k and j, it is decidable whether or not an arbitrary context-free $\Sigma \cup \Pi$-language L is (k,j)-*reversal-bounded*.
The result follows from the decidability of the same problem for *k-reversal-bounded* provided in [45]. It is sufficient to prove that L is *k-reversal-bounded* for **r**,**l** moves and *j-reversal-bounded* for **u**,**d** moves.

It is easy to verify that for (k,j)-*reversal-bounded* $\Sigma \cup \Pi$-grammars the corresponding picLR parser works in $O(t*(|q|*(t_F+1)))$ where $t=2(k+j-1)$, q is the symbolic picture to be recognized and t_F is the time required for *Find* algorithm. As a matter of fact the worst case is represented by input pictures which are described by (k,j)-*reversal-bounded* $\Sigma \cup \Pi$-words of this type: "$\underbrace{(rl)(rl)...(rl)}_{2k-1}\underbrace{(ud)(ud)...(ud)}_{2j-1}\alpha$" where α is in $\{\mathbf{r},\mathbf{u}\}^*$ or $\{\mathbf{r},\mathbf{d}\}^*$ or $\{\mathbf{l},\mathbf{u}\}^*$ or $\{\mathbf{l},\mathbf{d}\}^*$.

Given a (k,j)-reversal-bounded $\Sigma \cup \Pi$-grammar, k and j are fixed. Thus, if t_F is constant then the picLR parser for (k,j)-*reversal-bounded* $\Sigma \cup \Pi$-grammars works in linear time.

Finally, let us observe that the parser proposed in this chapter for the symbolic picture languages can be easily extended to drawn symbolic picture languages by accomplishing few modifications. In particular, a drawn symbolic picture to be parsed has to be internally represented as a set of points with the associated symbols and a set of segments stored as pairs $\{v, v'\}$ with $v, v' \in M_0$. Furthermore, whenever the parser executes a shift $action[s, b] = s\ sym, j$ with $next[s] = \pi$, it has to mark the symbol b and the the segment corresponding to the move relation π in the input picture.

Moreover, the extension can also be provided for drawn picture languages by considering only the symbol ϕ in the set Σ_ϕ. So, the membership result provided for the subclass of (k,j)-*reversal-bounded* $\Sigma \cup \Pi$-languages can be extended to the subclass of (k,j)-*reversal-bounded* Π-languages in a natural manner.

Chapter 6

Related Work

In recent years, there has been considerable interest in studying high-dimensional rewriting systems for the generation, description and recognition of *pictures* [22, 23, 29, 49, 50, 52, 58]. Each of these studies considers a particular data structure (a string, an array, a tree, a plex, a graph) or an algebric structure [51] for representing the pictures. The patterns consisting of geometric curves are particularly suitable for description by strings. Pioneering work in suggesting and applying string languages has been done by Freeman [24, 25], which introduced the notion of 'chain code' as a *digital representation* of line drawing, that was originally described using a string of symbols over an eight-letters alphabet to denote the movements *north*, *south*, *east*, *west*, *northeast*, *northwest*, *southeast* and *southwest*. This encoding method represents a geometric curve by a sequence of octal digits called a chain. Since a chain is a string, these pattern languages can be treated as string languages [26]. The chain-encoding scheme relates to a research area involving methods for the 'linearization' of the pattern recognition problem. Such methods were studied by many authors. Among them, Narashiman [53] has defined a grammar generating the *English alphabet*, Pavlidis [54] has characterized *convex polygons*, Shaw [63] has described *graphs* representing patterns and Feder [21] has defined and characterized *line patterns*. The linearization was achieved either by assuming that line-like patterns only are presented for recognition (e.g. Narashiman, Feder) or by specifying certain primitives which could be combined to form all 'interesting' patterns (e.g. Shaw). In the second case a grammar was used to specify the rules for combining the primitives.

Such an approach of describing languages consisting of multidimensional objects by string languages has the potential advantage of the use of a considerably rich body of knowledge about *string grammars and languages* (see, e.g. Ginsburg, 1966, Salomaa , 1973) to study languages consisting of structures more complicated than strings.

Another important chain-code model is the picture description method used in 'turtle geometry' [1], where the trace left by a *turtle* (or *robot*) while moving in the plane is interpreted as a picture. Good surveys for picture description models, some of which are similar to the chain code scheme, can also be found in [60] for picture language automata and grammars, and in [27] for general syntactic pattern recognition methods.

Subsequently, some efforts toward classifying the complexity of picture languages [52, 66] have simplified the chain codes to strings over a four letters alphabet $\{\mathbf{r}, \mathbf{l}, \mathbf{u}, \mathbf{d}\}$ to represent the movements *right*, *left*, *up* and *down*, respectively. One of the most elegant and successful four-letters chain code methods for working with line-drawing was introduced in 1982 by Maurer, Rozemberg and Welzl [52]. They define a *drawn picture* as a connected set of axis parallel unit lines from the Cartesian plane considered as a square grid, with the specification of its start and end points. A *picture description* is a word over the set $\Pi=\{\mathbf{r}, \mathbf{l}, \mathbf{u}, \mathbf{d}\}$ describing the picture by the graphical representation of the movement sequence. Such picture model has motivated the introduction of drawn symbolic pictures (see chapter 2) which have been widely investigated in this thesis.

Many authors have continued the study of chain code pictures (see for example [3, 4, 16, 18, 33, 34, 47, 65, 64]). By changing the semantic of the alphabet Π, in [47] a description of picture made of *pixel* (*unit square*) instead of segments has been proposed. Some details on such semantic are presented in section 6.1.
Moreover, additional features allow to describe different *colours* for the move letters, including the "invisible" one (by the "pen up"). A more realistic description is achieved this way, but also a more difficult one. As a matter of fact, problems like the membership that are decidable for context-free picture description with "pen down" results to be undecidable in the case of "pen up" [36]. In section 6.2 we first present some basic notions about this description model, and then we provide a short comparison between the general concept of picture descripion languages with "pen up", and the (restricted) case of the picture description languages with "pen down".

More recently, new chain code have been proposed for shapes composed of regular cells and for representing *3D curves* (see [6, 7]). In particular, in [6] the author represents *3D discrete curves* composed of constant *straight-line segments*. Two contiguous straight-line segments define a *direction change* and two direction changes define a *chain element*. Such discrete curves are isolated from the real world and are obtained by a pre-processing. Moreover, they are described by only five orthogonal direction changes. The use of relative direction changes allow to consider curve descriptions invariant under translation and rotation, and optionally, under mirroring transformation (see [6]). In section 6.3, such approach will be analyzed in order to provide a comparison with our charaterization of pictures. In particular, we will show that the string based repesentation indroduced for drawn symbolic picture (see section 2.2.1) is able to describe 3D discrete curves.

Other interesting approaches proposed for the descriptions of the pictures are: the *two dimensional string languages*[30, 31, 32, 39, 40, 67] and the *shape grammars* [5].

In the context of two dimensional languages pictures are conceived as sets of rectangular arrays of symbols and represent a generalization of the notion of string languages to picture languages. In [30] the notion of recognizability of a set of pictures in terms of *tiling systems* has been introduced. In particular, the authors define the notion of a local picture language by giving a set of authorized 2x2 tiles over $\Sigma \cup \{\#\}$ where # is a boundary symbol that surrounds the pictures. They define the class of recognizable picture languages (generated by finite state machines) which is of interest since they has been proved that the following families of two dimensional languages coincide: languages recognized by on-line tesselation automata [39, 40], languages expressed by formulas of existential monadic second-order logic [32, 67], languages recognized by finite tiling systems [30, 31], languages corresponding to regular expressions of special type [31].

A *shape grammar* represents a *pictorial computability model* which operate on polygons consisting of unit square orthogonally connected. Pixel are the elementary objects and by *orthogonally adjoining* them without overlapping polyomonoes are obtained. Shape grammars combine the approach of deriving linear descriptions (strings) of line segments [26, 52, 36] and connected set of pixel [63, 47] (that are the typical atomic structures used) and the approach of chataracterize pictorial languages in terms of 2D [60, 31]. Moreover, in [5] the authors characterize the recursively enumerable languages in this framework.

6.1 Pixel Pictures

In this section we provide basic notions of the pixel picture model as defined in [47]. Pictures are considered as a finite set of pixel, where a pixel can be seen as a *unit square* described by [i,i+1]×[j,j+1] with (i,j)∈Z^2 and denoted as *pix*(i,j).
More formally, a *"pixel picture"* is a triple $q = (p, d, a)$ where p is a set of pixel and is denoted as *base*(q), d∈Z^2 is the *start point* of q and it is denoted as *dep*(q), and a∈Z^2 is *the end point* of q and it is denoted as *arr*(q). The start and the end point allow to concatenate two *"pixel picture"*. In particular given two *pixel picture* p and q, the concatenation of p and q, denoted as $p \bullet q$ is defined if $dep(q)$=$arr(p)$, and $p \bullet q$=(base(p)∪base(q), dep(p), arr(q)). In order to formally define the notion of *connected pixel picture*, two concept have been defined [47]: the *armour* of a pixel picture, and the *neighbors* of a pixel picture in analogy with the concepts of neighbor of a point for the drawn pictures.

Given a pixel picture q=(p, d, a) the armour of p is defined as:
$arm(p)$=$\bigcup_{pix(i,j)\in p}${(i,j),(i+1,j),(i,j+1),(i+1,j+1)}
and the armour of q as: $arm(q)$=arm(base(q)).

Given a pixel pix(i,j), its neighbors are defined as:
neigh(pix(i,j))={ pix(k,l) | i-1≤k≤i+1 and j-1≤l≤j+1, with (k,l)≠(i,j)}

These two concepts are useful to formally define the notions of connected pixel picture. In particular, given a pixel picture q=(p, d, a)

1. base(q) is connected if:
   $\forall a, b \in p\ \exists c_1, c_2, ..., c_n \in p$ such that $c_1 = a$, $c_n = b$ and ∀1≤i<n $c_{i+1} \in neigh(c_i)$
2. q is connected if :
   a) $base(q)$ is connected; b) $dep(q) \in arm(q)$; c) $arr(q) \in arm(q)$

In other words a pixel picture is connected if for each pixel pix(i,j) it is possible to go from pix(i,j) to any other pixel by following the neigbors of pix(i,j).

A connected pixel picture is shown in Fig. 6.1, where the white circle represents the start point, and the black circle represents the end point.

According to the above definitions, two pictures are equivalent if one can be translated into the other by using the following *translational mapping.*
Given (x,y)∈Z^2, $\tau_{m,n}$(x,y) = (x+m,y+n). Thus given a pixel pix(x,y) and (i,j)∈Z^2,

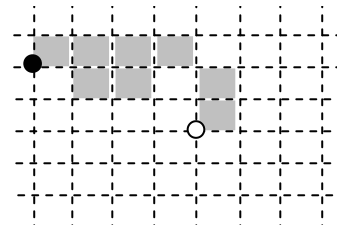

Figure 6.1: *A pixel picture.*

$\tau_{i,j}$(pix(x,y))=(pix(x+i,y+j). As a consequence, given two pixel picture p and q, p is *translational equivalent* to q, if there exist two integer i and j such that: $\tau_{i,j}$(base(p))=base(q), $\tau_{i,j}$(dep(p))=dep(q),$\tau_{i,j}$(arr(p))=arr(q).

For this approach the string descriptions of pictures on the alphabet Π is such that, the letters of Π induce always a unit move but it is not plot the segment under the move but the pixel in the right of the move. Thus, the pixel picture of Fig 6.1 can be described by the Π-word **uulldull**.

It is worth noting that the pixel picture model is an adaptation of the classic picture languages proposed in [52]. On the other hand, in [47] the authors have proved that the set of connected pixel pictures is a finitely generated inverse monoid.

Let us observe that the pixel picture model could be extended by introducing the notion of symbol associated to pixel. As an example, it can be interesting to consider colors associated to the pixels of a picture. As in the case of drawn symbolic pictures, such new picture model can require that at most one symbol has to be associated to a pixel of a picture.

On the other hand, the string based representation of this picture model will contain symbols from the alphabet Π and symbols from an alphabet Σ. Again, a $\Sigma \cup \Pi$-word can associate several symbols to a pixel and so, the problem of the symbol overlapping has to be tretated. We can still exploit a merging function f to determine the unique symbol to be associated to any pixel.

Moreover, the notion of $dspic_f$ introduced for drawn symbolic pictures can be easily extended to work on pixel pictures. Indeed, the new $dspic_f$ only differs in the interpretation of the move letters. The letters of Π induce always a unit move but it is not plot the segment under the move but the pixel in the right of the move.

6.2 Non-Connected Pictures

An *uncolored basic picture* is a set of *unit line* in $\mathrm{M}_1 = \{\{v, v'\} | v, v' \in M_0 \ and \ v' \in N(v)\}$. Given a vector of colors C=$(c_1, c_2, ..., c_n)$, a *colored basic picture* is a vector of uncolored basic picture $(p_1, p_2, ..., p_n)$, where the i-th picture is understood to be drawn with color c_i. If the set of colors contains only one symbol then we can consider colored pictures as uncolored pictures. A colored drawn picture is a triple $q =< p, s, e >$ where p is a basic picture denoted as $base(q)$, s and e are the *start* and the *end* points. Two pictures are called translational equivalent if they only differ by their relative position in the plane, but the shape is the same.

Let us observe that the best investigated picture description based on the alphabet Π (the drawn picture description proposed in [52]) can be used to describe connected uncolored pictures, where **r**=(1,0), **l**=(-1,0), **u**=(0,1) and **d**=(0,-1). In order to describe disconnected colored pictures it is possible to add the set of colors CS= $\{\uparrow, c_1, c_2, ..., c_n\}$ to the alphabet Π, where ↑ denotes the "empty color". A word w over Π∪CS is in alternating normal form if the color letters and the move letters alternate. More formally the drawn picture described by a word can be defined inductively by the following function *dpic*:

- if w=ϵ then $dpic(\epsilon)$= <(∅,...,∅),(0,0),(0,0)>
- if w=w'↑ b where $dpic(w)$=$< (p_1, p_2, ..., p_n), s, e >$ and b∈ Π then

$$dpic(w) = < (p_1, p_2, ..., p_n), s, e + b >$$

- if w=$w'c_ib$ where $dpic(w)$=$< (p_1, p_2, ..., p_n), s, e >$ and b∈ Π and c_i ∈CS then

$$dpic(w) \ = < (p_1, ..., p_i \cup \{\{e, e + b\}\}, ..., p_n), s, e + b >$$

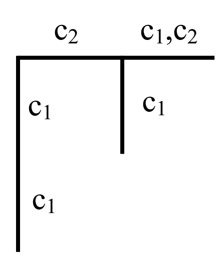

Figure 6.2: *A picture.*

In the general case of a $\Pi \cup$CS-word, a color letter occurring in the word specifies the color of the sequence of moves described by the subsequent move letters. As an example in the $\Pi \cup$CS-word "$c_1uuc_2rrc_3c_1ld$" representing the picture of Fig. 6.2, c_1 indicates the color of the moves "**uu**" and "**ld**, and " c_2 of the moves "**rr**". c_3 does not specify a color of move because it is not followed by a move letter. This is the motivation for the trasformation of an arbitrary word over $\Pi \cup$CS into a word in alternated normal form. This can be done by using the generalized sequential machine ALT introduced in [36], such that:

- ALT(w)=w for each w that is in alternated normal form, and
- $dpic(w)$=$dpic$(ALT(w)) for each general word w over $\Pi \cup$CS.

Given a string language L over the alphabet $\Pi \cup$CS, L generates: the picture language pic(L)=$\{base(pic(w)) \mid w \in L\}$ and the drawn picture language $dpic$(L)=$\{dpic(w) \mid w \in$L$\}$.

An interesting case is represented by the alphabet $\Pi \cup \{\uparrow, \downarrow\}$ which can be used to describe disconnected pictures. Moreover we can also consider the picture on this alphabet as uncolored, i.e it is the set of pictures obtained by "pen up" moving in the plane.

It is worth noting that it is possible to obtain connected pictures (i.e., all the unit lines of the picture are visible) from a "pen up" description pictures by applying an homorphism h_0. We can consider a new color c_0 that makes invisible lines visible. The new set of colors will be $\{c_0\}\cup$CS-$\{\uparrow\}$ and the homomorphism is such that: $h_0(\uparrow)= c_0$ and $h_0(b) = b$ for $b \in \Pi \cup \{c_0\}\cup$CS-$\{\uparrow\}$. Thus, a picture language B is a "pen down" picture language if there exists a language L such that B=$pic(h_0$(L)).

One of the most relevant results about "pen down" languages states that the set of all descriptions of one fixed picture is always regular. This does not hold for "pen up" languages [36]. We have verified the same result for the set of symbolic pictures descriptions in section 4.1.

Let us observe that we can manage process colors of the segments in a similar way to the symbols of drawn symbolic pictures. As a matter of fact we can associate only one symbol to each segment.

Formally we can have the following definition of a *drawn colored picture.*

Definition 6.1 *A drawn colored picture is a triple* $q = << p, s, e >, C, \rho >$, *where* C=$\{c_1,c_2,\ldots,c_n\}$ *is an alphabet of colours,* ρ *is a function* ρ: $W(< p, s, e >) \times W(<$

$p, s, e >) \rightarrow C_{\overline{\phi}}$ that specified the colors associated to the segments. $C_{\overline{\phi}}$=C$\cup\overline{\phi}$ where $\overline{\phi}$ is a special symbol that denotes the absence of colors and it play the same role of the special symbol ϕ for drawn symbolic pictures. The start and end points of q are s and e, respectively. The empty drawn colored picture is denoted by $<< \emptyset, (0,0), (0,0) >, C, \rho >$.

Moreover, we can provide a string based representation of a drawn colored picture, in terms of a word on alphabets C and Π. Whenever more symbols are described for the same segment we can apply a merging function g to determine the unique symbol. Formally, we could use the following function *dpic* which has as a parameter a merging function g and use a vector S to store for the positions the last symbol associated to them. As a matter of fact, during the application of *dpic* on a string description, we consider the symbol associated to the position of the picture as the color of the segment that follows.

Definition 6.2 *Let w be a $C \cup \Pi$-word and g be a merging function on $C_{\overline{\phi}}$. The drawn colored picture described by w w.r.t. the merging function g, denoted by $dpic_g(w)$, is defined inductively as follows:*

- *if $w = \epsilon$, then $dpic_g(w)$ is the empty drawn colored picture;*
- *if w= $w'\tau$ for some $w'\in(C \cup \Pi)^*$, with $dpic_g(w') = <sc,\ C,\ \rho >$, $sc = <p,\ s,\ e>$, and $S(e)=c$*

$$dpic_g,(w) = \begin{cases} dpic_g(w') \text{ and set } S(e) = \tau & \text{if } \tau \in C \\ << p \cup \{e, \tau(e)\}, s, \tau(e) >, C, \underline{\rho} > & \text{if } \tau \in \Pi \end{cases}$$

where $\underline{\rho}$: $(W(sc)\cup\{\pi(e)\}) \times (W(sc) \cup \{\pi(e)\}) \rightarrow C_{\overline{\phi}}$ is the function such that:

$$\underline{\rho}(v, v') = \begin{cases} g(\rho(v, v'), S(e)) & \text{if } (v, v') \in (W(sc) \times W(sc)) \text{ and } (v, v') = (e, \pi(e)) \\ S(e) & \text{otherwise} \end{cases}$$

As a consequence of the above characterization, the notion of *g-independency* can be provided also for drawn colored picture. A $C \cup \Pi$-word w is *g-independent* if and only if it specifies at most one colour in C for each segment of the described picture. Otherwise we say that w presents a "conflict" or w is *g-dependent*.

6.3 The Drawn Symbolic Pictures to Specify Discrete 3D Curves

In the last years, new chain code models have been proposed for representing *3D curves*. In particular, in [6] the author represents *3D discrete curves* composed of constant straight-line segments. Two contiguous straight-line segments define a direction change and two direction change define a chain element. Such discrete curves are isolated from the real world and are obtained by a pre-processing. Moreover, they are described by only five relative (orthogonal) direction changes which are shown in Fig. 6.3.

The use of relative direction changes allow to consider curve descriptions invariant under translation and rotation, and optionally, under mirroring transformation (see [6]).

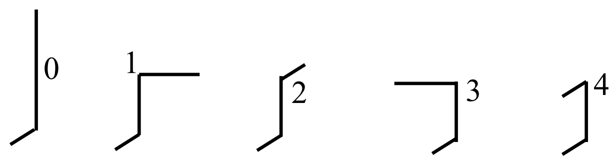

Figure 6.3: *The possible directions for 3D discrete curves*

The element labelled 0 represents the direction change which goes *straight* through the contiguos straight-line segments following the direction of the last segment; the element labelled 1 represents a direction change to the *right*; the element labelled 2 represents a direction change *upwards*; the element labelled 3 represents a direction change to the *left*; and the element labelled 4 indicates a direction change *going back*. The chain of a curve is a sequence of relative direction changes calculated around the curve. Fig. 6.4(i) shows an example of 3D discrete curve and Fig. 6.4(ii) shows the corresponding chain.

Formally, a chain A is an ordered sequence of elements a_1, a_2,..., a_n where n indicates the number of chain elements.

In the following we provide a characterization of 3D discrete curves in terms of string descriptions based on the alphabet of movements Π={**r**,**l**,**u**,**d**} and of two symbols. In other words, we simulate the use of the five possible relative direction changes used by Bibriesca through the four movements in Π and two symbols which are used to describe the changing of Cartesian Plane.

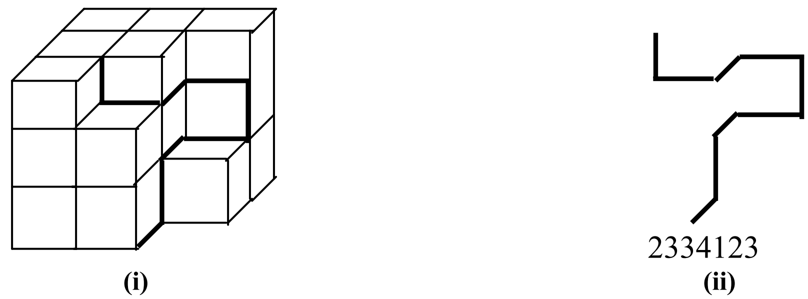

Figure 6.4: *An example of 3D discrete curves (i) and the corresponding chain (ii).*

Our analysis starts from this observation. We can consider the unit-size segments as basic elements of the 3D curves. So, the 3D discrete curve of Fig. 6.4(ii) is composed of 9 segments. Moreover, let v be a position in the 3D discrete curve where the direction changes. Next segment can be described by a move on the alphabet Π with respect to a standard plane with origin in v. It is worth noting that we need only to consider two of the three standard planes XZ, YZ, and XY in order to completely describe the 3D discrete curves. As an example, let us consider the 3D discrete curve of Fig. 6.4(ii), we show that it can be described by using only the symbols of Π and the two standard planes XZ and YZ. Each segment will be represented by a move in Π w.r.t. one of the two standard planes considered. We start by considering the first three segments of the 3D curve of Fig. 6.4(ii) described by a $left$, an *up* and a $left$ moves w.r.t. the standard plane XZ as depicted in Fig. 6.5(i). In order to describe the fourth, the fifth and the sixth segments in Fig. 6.4(ii) we need to change the cartesian plane, indeed they can be represented respectively by a *right*, an *up* and a $left$ moves w.r.t. the standard plane YZ as depicted in Fig. 6.5(ii). Thus, by using the two fixed standard plane XZ and YZ we are able to describe the 3D discrete curve of fig. 6.4(ii) as shown in Fig. 6.5.
In the sequel, we indicate the cartesian plane XZ with the symbol X and the cartesian plane YZ with symbol Y. Moreover, the string based representations of 3D discrete curves are represented by $\Sigma \cup \{X, Y\}$. So, in a string description if we see a symbol X then the next move letters refer to the standard plane XZ. Thus, the 3D discrete curve of Fig. 6.4(ii) can be represented by the $\Pi \cup \{X, Y\}$-word "$X\mathbf{lul}Y\mathbf{rul}X\mathbf{r}Y\mathbf{lu}$".

Several concepts and notions provided for chain code pictures in [6] can be extended to our characterization of 3D discrete curves.

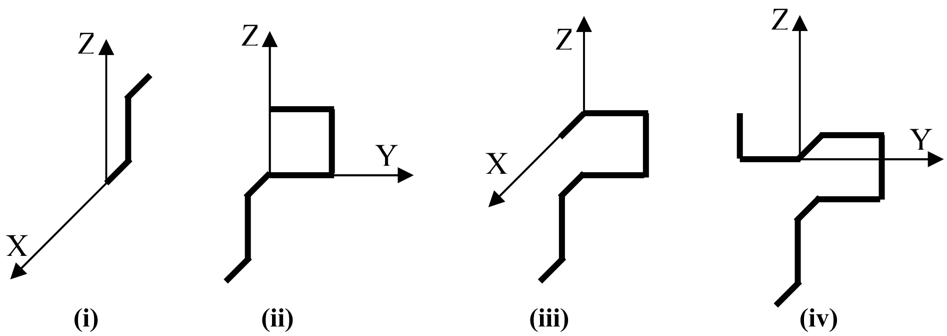

Figure 6.5: *The unit-size segments of Fig. 6.4(ii) w.r.t. to the standard planes XZ and YZ*

The *length* L of a chain is the sum of the length of its elements, i.e. the sum of the segments of size one. Thus, given a $\Pi \cup \{X, Y\}$-word w, $length(w)$ is the number of move letters in w. As an example $length(X\mathbf{lul}Y\mathbf{rul}X\mathbf{r}Y\mathbf{lu})$=9.

The *inverse* of a chain is the chain obtained by arranging the inverse of each element in reverse order. Obviously, the inverse of $\mathbf{r}$ ($\mathbf{u}$, resp.), denoted by Inv($\mathbf{r}$) (Inv($\mathbf{u}$), resp.), is $\mathbf{l}$ ($\mathbf{d}$, resp.), and vice versa. As an example, the inverse of the chain code describing the 3D discrete curve of Fig. 6.4(ii) is the $\Pi \cup \{X, Y\}$-word "$Y\mathbf{dr}X\mathbf{l}Y\mathbf{rdl}X\mathbf{rdr}$".

It can be easily verified that the chain code proposed is invariant under rotation. Fig. 6.6 depicts some rotations of the discrete curves of Fig. 6.4(ii). Figg. 6.6(a) and 6.6(b) show some trasformations performed on the axis of rotation "X", Figg. 6.6(c) and 6.6(d) describe some rotations w.r.t axis "Y" and Figg. 6.6(e) and 6.6(f) w.r.t. axis "Z". Let us observe that all chains are equal, and so they are invariant under rotation.

Moreover, according to our string representation of 3D discrete curves, the chain code is invariant under mirroring trasformation.

The chain of the mirror of a 3D discrete curve described by the $\Pi \cup \{X, Y\}$-word "$a_1a_2...a_n$" is "$a'_1a'_2...a'_n$" which is obtained as follows:
if the mirroring trasformation is provided w.r.t. the standard plane XY then

$$a'_i = \begin{cases} a_i & \text{if } a_i \in \{\mathbf{r}, \mathbf{l}\} \text{ or } a_i \in \{X, Y\} \\ Inv(a_i) & \text{if } a_i \in \{\mathbf{u}, \mathbf{d}\} \end{cases}$$

if the mirroring trasformation is provided w.r.t. the standard plane XZ then

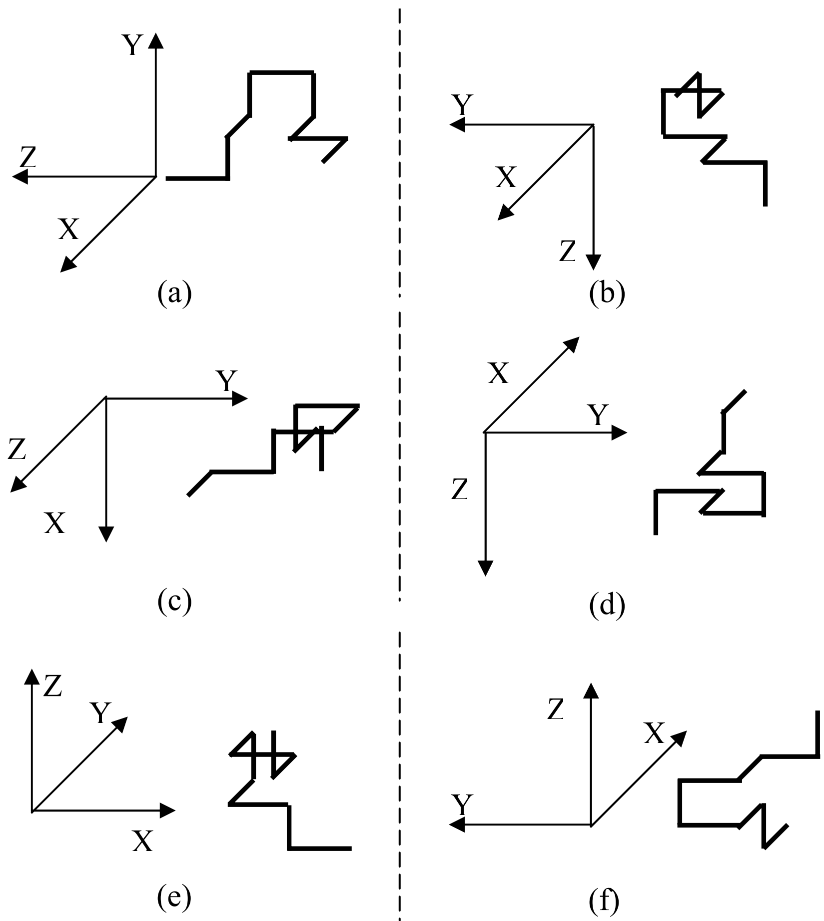

Figure 6.6: *Some rotations of the curve of Fig. 6.4(ii) on the axis "X" (a) and (b), on the axis Y (c) and (d) and on the axis Z (e) and (f)*

$$a'_i = \begin{cases} a_i & \text{if } a_i \in \{\mathbf{r}, \mathbf{l}\} \text{ w.r.t. XZ or } a_i \in \{\mathbf{u}, \mathbf{d}\} \text{ or } a_i \in \{X, Y\} \\ Inv(a_i) & \text{if } a_i \in \{\mathbf{r}, \mathbf{l}\} \text{ w.r.t. YZ} \end{cases}$$

if the mirroring trasformation is provided w.r.t. standard plane YZ then

$$a'_i = \begin{cases} a_i & \text{if } a_i \in \{\mathbf{r}, \mathbf{l}\} \text{ w.r.t. YZ or } \mathrm{a}_i \in \{\mathbf{u}, \mathbf{d}\} \text{ or } a_i \in \{X, Y\} \\ Inv(a_i) & \text{if } a_i \in \{\mathbf{r}, \mathbf{l}\} \text{ w.r.t. XZ} \end{cases}$$

As an example, Fig. 6.7 depicts the mirroring trasformation of 3D discrete curve of Fig. 6.4(ii) w.r.t. to the standard plane XY (a), w.r.t. XZ (b) and w.r.t. YZ (c).

The properties described above ensure that a unique curve description can be generated, which allow to easily perform curve comparison.

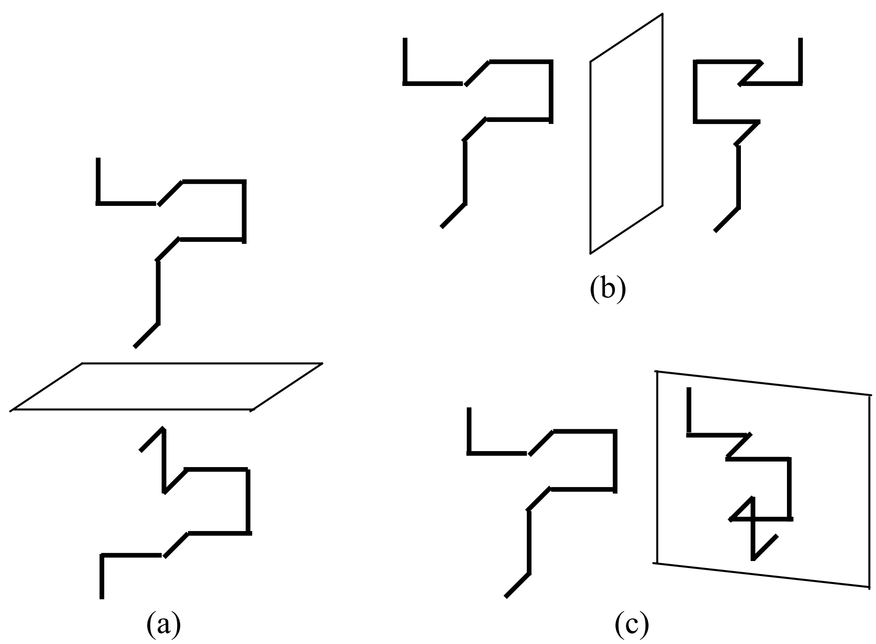

Figure 6.7: *The mirroring trasformation of 3D discrete curves in Fig. 6.4(ii)*

Final Remarks

The model of drawn symbolic pictures is turned out to be a simple, intuitive and useful formalism to describe more complex pictures. Nevertheless the introduction of symbols requires new issues to be addressed. In the thesis we have investigated several of these issues by providing descriptive and generative models. In particular, a string-based representation has been used to describe a drawn symbolic picture, and a mapping $dspic_f$ has been defined in order to obtain a unique drawn symbolic picture from a string description. Such a mapping is parametric with respect to a merging function f, which has a crucial role in establishing which symbol must be visualized whenever more than one symbol is associated to a position. As exemplified in the thesis, many different merging functions f can be considered, making the mechanism for drawn symbolic picture description and construction to be very general. Moreover, we have identified the properties of the merging function f ensuring that the set of drawn symbolic pictures is a finitely generated monoid. Such a property holds for all picture semigroups used in "picture language theory", such as the set of drawn pictures [52], the set of non-connected pictures [36] and the set of connected pixel pictures [47]. Let us observe that the function $dspic_f$ is a morphism from the monoid of the string descriptions into the monoid of drawn symbolic pictures.

The analysis of the string descriptions has allowed us to point out also the notion of f-*independency* and the decidability of the f-independency problem for regular languages has been proved. Since the model of drawn symbolic pictures represents an extension of drawn pictures, it is intuitive that properties holding for drawn picture languages not always can be inherited by their symbolic version. Thus, the issue to determine the possible conditions ensuring the preservation of these properties deserves special attention. In the thesis we have focused on the Membership results for the class of stripe, three-way, and k-retreat-bounded languages and conditions able to ensure the preservation of such results have been provided. Such conditions involve the definition of the merging function and the f-independency of the string descriptions- Moreover, we have provided the notion of *symbolic picture* as a generalization of drawn symbolic picture by using the notion of *invisible lines* introduced in [36, 65]. Informally, symbolic picture languages can be considered as drawn symbolic picture languages where the spatial relations are not visualized. Even though this aspect may appear trivial, it has introduced another degree of freedom in picture string descriptions which has determined different complexity properties. As a matter of fact, it has been proved that the string description language of a symbolic picture is context-sensitive languages while the string description language of a drawn symbolic picture is a regular language.

One of the motivation for the study of (drawn) symbolic picture languages is the close similarity between the description of symbolic pictures and the way two-dimensional sentences are generated by a Linear Positional Grammar [8]. Thus, a discussion on the relationship between drawn symbolic picture grammars and Positional Grammars has been presented. Then starting from our experiences on the pLR parsing [11], we have provided the construction of an LR parser for context-free (drawn) symbolic picture languages and we have proved that it works in linear time for some subclasses of (drawn) symbolic pictures.

Other interesting issues can been foreseen as future work, such as the extension of the analysis to symbolic picture languages and the identification of other classes of drawn symbolic picture languages with nice decidability and complexity properties.

References

[1] H. Abelson, and A. Di Sessa, *Turtle Geometry*. The MIT Press, Cambridge, MA, 1986.

[2] A.V. Aho, R. Sethi, and J.D. Ullman, Compilers, principles, techniques and tools, Addison Wesley, New York, 1985.

[3] F.J. Brandenburg: 1989 "On Minimal Picture Words", *MIP 8903, Tech. Report, University of Passau.*

[4] F.J. Brandenburg and J. Dassow: 1993 "Reduction of Picture Words", *RAIRO TCS* **27**, pp. 49-56.

[5] P. Bottoni, G. Mauri, P. Mussio, G. Păun, "Computing with Shapes", TUCS technical Report No 136. October 1997.

[6] E. Bribiesca, "A New Chain Code', *Pattern Recognition* **32**,N.2, (1999), pp. 235-251.

[7] E. Bribiesca, "A chain code for representing 3D curves", *Pattern Recognition* **33**,N.5, (2000), pp. 755-765.

[8] G. Costagliola and S.K. Chang, "Parsing Linear Pictorial Languages by Syntax-Directed Scanning", *Languages of Design* **2**, (1994), pp. 229-248.

[9] G. Costagliola and S.K. Chang, "Using Linear Positional Grammars for the LR Parsing of 2-D Symbolic", *Grammars*, Kluwer Publishing.

[10] G. Costagliola, V. Deufemia, F. Ferrucci and C. Gravino "Decidability of the consistency problem for regular symbolic picture description languages", *Information Processing Letters* **80** (2001) 119-124.

[11] G. Costagliola, V. Deufemia, F. Ferrucci and C. Gravino G, "On the pLR Parsability of Visual Languages" in *Proceedings of IEEE International Symposia on Human-Centric Computing Languages and Environments (HCC'01)*, Stresa, 5-7 Settembre, 2001, pp. 48-49.

[12] G. Costagliola, V. Deufemia, F. Ferrucci, C. Gravino, "On Regular Drawn Symbolic Picture Languages". Submitted to *Information and computation* (in second revision).

[13] G. Costagliola, V. Deufemia, F. Ferrucci, C. Gravino, M. Salurso, "Symbolic picture description languages", in C. Martin-Vide & V. Mitrana, eds. Grammars and Automata for String Processing: from Mathematics and Computer Science to Biology, and Back (Taylor and Francis, London, 2003).

[14] G. Costagliola, A. De Lucia, S. Orefice, and G. Tortora, "A Parsing Methodology for the Implementation of Visual Systems", *IEEE Trans. Software Engineering* **23**, (1997), pp. 777-799.

[15] G. Costagliola, A. De Lucia, S. Orefice, G. Tortora, "A Framework of Syntactic Models for the Implementation of Visual Languages". *Procs. 13th IEEE Symposium on Visual Languages*, Capri, Italy, (1997), pp. 58-65.

[16] G. Costagliola, and F. Ferrucci: "Symbolic Picture Languages and their Decidability and Complexity Properties", Special Issue on the Theory of Visual Languages, *Journal of Visual Languages and Computing* **10**, 1999, pp. 381-419.

[17] G. Costagliola, F. Ferrucci, R. Francese and C. Gravino, "Using Alternating Words to Describe Symbolic Pictures", Technical Report

[18] G. Costagliola and M. Salurso, "Drawn Symbolic Picture Languages", *Procs. International Workshop on Theory of Visual Languages*, Capri, Italy, Sept. 27, (1997), pp. 1-14.

[19] G. Costagliola and G. Polese, "Extended Positional Grammars", *Proceedings of 2000 IEEE Symposium on Visual Languages*, Seattle, WA, USA.

[20] K. Culik and Welzl, "Two Finite State Generator", *Lecture Notes on Computer Science* **158**, (Springer, Berlin, 1983), 106-114.

[21] J. Feder, "Languages of Encoded Line Patterns", *Information and Control*, **13**, (1968), pp. 230-244.

[22] J. Feder, "Plex Languages", *Information Sciences* **3**, (1971), pp. 225-241.

[23] F. Ferrucci, G. Pacini, G. Satta, M.I. Sessa, G. Tortora, M. Tucci, and G. Vitiello, "Symbol-Relation Grammars: A Formalism for Graphical Languages", *Information and Computation* **131**, (1996) pp. 1-46.

[24] H. Freeman, "On Encoding Arbitrary Geometric Configuration", *IRE Trans Electroning Computation*, **10**, (1961), pp. 260-268.

[25] H. Freeman, "On the Digital Computer Classification of Geometrics Line Patterns", *in Proc. 18th Nat. Elect. Conf.*, (1962), pp. 312-334.

[26] H. Freeman, "Computer Processing of Line-Drawing Images", *Computer Surveys*, **6**, (1974), pp. 57-97.

[27] K.S. Fu *Syntattic Pattern Recognition and Applications.* Prentice-Hall, Inc. Englewood Cliffs, N.J., 1982.

[28] S. Ginsburg, *The mathematical Theory of Context-Free Languages*, McGraw Hill, New York, (1966)

[29] E. J. Golin and S. P. Reiss, "The Specification of Visual Language Syntax", *Journal of Visual Languages and Computing* **1**, (1990), pp. 141-157.

[30] D. Giammaresi, A. Rstivo, "Recognizable Picture Languages", in: M. Nivat, A. Saoudi and P.S.P. Wand, eds., *Proc. 1st Internat. Colloq. on Parallel Image Processing, Internat. J. Pattern Recognition Artif. Intell.* **6** (1992) 231-256.

[31] D. Giammaresi, A. Rstivo, "Two dimensional languages", Chapter 4 in Volumes 3 of G. Rozenberg, A. Salomaa, Eds., Handbook of Formal Languages, Springer-Verlag, Heidelberg, 1997.

[32] D. Giammaresi, A. Rstivo, S. Seibert and W. Tomas, "Monadic second-order logic over rectangular pictures and recognizability by tilling systems", *Information and Computation*, **125**, (1996), pp. 32-45.

[33] R. Gutbrod, "A Transformation System for Generating Description Languages of Chain Code Pictures", *Theoretical Computer Science*, **68**, (1989), pp. 239-252.

[34] F. Hinz, "Classes of Picture Languages that Cannot be Distinguished in the Chain Code Concept Deletion of Redundant Retreats", *Procs. STACS'89, Lecture Notes in Computer Sciences* **349**, Springer Berlin, (1989), pp. 132-143.

[35] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA,1979.

[36] F. Hinz and E. Welzl, "Regular chain code picture languages with invisible lines". Tech. report 252, IIG, Tech. Univ. Graz, Austria, 1988.

[37] R. C. Holt, A. Winter, and A. Schrr, "GXL: Toward a Standard Exchange Format", *7th Working Conference on Reverse Engineering, IEEE Computer Soc.*, 2000, 162-171.

[38] S. C. Johnson, "YACC: Yet Another Compiler Compiler", Bell Laboratories, Murray Hills, NJ, 1978.

[39] K. Inoue and A. Nakamura, "Some properties of two-dimensional on-line tessellation acceptors", *Information Science* **13** (1977) 95-121.

[40] K. Inoue and I. Takanami, "A characterization of recognizable picture languages", in: *Proc. 2nd Internat. Colloq. on Parallel Image Processing, Lecture Notes in Computer Science*, Vol. 654 (Springer, Berlin, 1992) 133-143.

[41] C. Kim, "Picture Iteration and Picture Ambiguity", *Journal of Computer and System Sciences*, **40**, (1990), pp. 289-306.

[42] C. Kim, "Complexity and Decidability for Restricted Class of Picture Languages", *Theoretical Computer Science*, **73**, (1990), pp. 295-311.

[43] C. Kim, "Retrat Bounded Picture Languages", *Theoretical Computer Science* **132** (1994) 85-112.

[44] C. Kim, and I.H. Sudborough, "The Membership and Equivalence Problems for Picture Languages", *Theoretical Computer Science*, **52**, (1987), pp. 177-191.

[45] C. Kim, and I.H. Sudborough, "On Reversal Bounded Picture Languages", *Theoretical Computer Science*, **104**, (1992), pp. 185-206.

[46] C. Kim and I.H. Sudborough, "Leftmove-bounded Picture Languages", *Theoretical Computer Science* **237** (2000) 183-195.

[47] M. Latteux, D. Robilliard and D. Simplot, "Figures composes de pixels et monode inversif", *Bullettin of Belgian Mathematical Society* 4 (1997), pp. 89-111.

[48] M. Latteux and D. Simplot, "Recognizable Picture Languages and Domino Tiling", *Theoretical Computer Science* **178** (1997), pp. 275-283.

[49] K. Marriott, "Constraint Multiset Grammars", *in Procs. of IEEE Symposium on Visual Languages*, St. Louis, Missouri, (1994), pp. 118-125.

[50] G. Masini, and R. Mohr, "MIRABELLE, a system for structural analysis of drawings", *Pattern Recognition*, **16**, n. 4, (1982), pp. 363-372.

[51] R. Mohr, "Precompilation of syntactical descriptions and knowledge directd analysis of patterns", *Pattern Recognition*, **19**, n. 4, (1986), pp. 255-266.

[52] H.A. Maurer, G. Rozenberg, and E. Welzl, "Using String Languages to Describe Picture Languages", *Information and Control* **54**, (1982), pp. 155-185.

[53] R. Narasimhan, "Sintax-Direct Interpretation of Classes of Pictures", *Commm. ACM*, **9**, (1966), pp. 166-173.

[54] T. Pavlidis, "Analisis of Set Patterns", *Pattern Recognition*, **1**, (1968), pp. 165-178

[55] M. Petrich, "*Inverse Semigroups*", Wiley, New York (1984).

[56] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall (1982).

[57] M. Pelletier and J. Sakarovitch, "EasyMultiplication II - extensions of rational semigroups", *Information and computation* **88** (1990), 18-59.

[58] J. Rekers, and A. Schurr, "A Graph Grammar Approach to Graphical Parsing", *Procs. of the 1995 IEEE Symposium on Visual Languages*, Darmstadt, Germany, (1995).

[59] D. Robilliard and D. Simplot, "Undecidability of Existential Properties in Picture Languages", *Theoretical Computer Science* **233** (2000) 51-74.

[60] A. Rosenfeld, *Picture Languages - Formal Model of Picture Recognition.* Academic Press, New York, 1979.

[61] J. Sakarovitch, "Description des monoides de type fini", *Journal of Information Processing and Cybernetics* **17**.

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

[63] A.C. Shaw, "A Formal Picture Description Scheme as a Basis for Picture Processing System", *Information and Control*, **14**, (1969), pp. 9-52.

[64] P. Séébold, and K. Slowinski, "The Shortest Way to Draw a Connect Picture", *Computer Graphics Forum*, **10**, (1991), pp. 319 327.

[65] K. Slowinski, "Picture Words with Invisible Lines", *Theoretical Computer Science*, **108**, (1993), pp. 357-363.

[66] I.H. Sudborough, and E. Welzl, "Complexity and Decidability for Chain Code Picture Languages", *Theoretical Computer Science*, **36**, (1985), pp. 173-202.

[67] W. Tomas, "On logics, tilings, and automata", in: *Proc. 18th Internat. Colloq. on Automata, Languages and Programming, Lecture Notes in Computer Science*, Vol. 510 (Springer, Berlin, 1991) 441-453.