INFORMATION AND COMPUTATION **131**, 1–46 (1996)
ARTICLE NO. 0090

Symbol–Relation Grammars: A Formalism for Graphical Languages

F. Ferrucci

Dipartimento di Informatica ed Applicazioni, Università di Salerno, I-84081 Baronissi, Salerno, Italy
E-mail: jentor@udsab.dia.unisa.it

G. Pacini

Dipartimento di Matematica Applicata ed Informatica, Università di Venezia, Via Torino 155, 30170 Mestre (VE), Italy
E-mail: pacini@di.unipi.it

G. Satta

Dipartimento di Ingegneria Elettronica ed Informatica, Università di Padova, via Gradenigo 6/a, I-35131Padua, Italy
E-mail: satta@dei.unipd.it

and

M. I. Sessa, G. Tortora, M. Tucci, and G. Vitiello

Dipartimento di Informatica ed Applicazioni, Università di Salerno, I-84081 Baronissi, Salerno, Italy
E-mail: jentor@udsab.dia.unisa.it

A common approach to the formal description of pictorial and visual languages makes use of formal grammars and rewriting mechanisms. The present paper is concerned with the formalism of Symbol–Relation Grammars (SR grammars, for short). Each sentence in an SR language is composed of a set of symbol occurrences representing visual elementary objects, which are related through a set of binary relational items. The main feature of SR grammars is the uniform way they use context-free productions to rewrite symbol occurrences as well as relation items. The clearness and uniformity of the derivation process for SR grammars allow the extension of well-established techniques of syntactic and semantic analysis to the case of SR grammars. The paper provides an accurate analysis of the derivation mechanism and the expressive power of the SR formalism. This is necessary to fully exploit the capabilities of the model. The most meaningful features of SR grammars as well as their generative power are compared with those of well-known graph grammar families. In spite of their structural simplicity, variations of SR grammars have a generative power comparable with that of expressive classes of graph grammars, such as the edNCE and the N-edNCE classes.

0890-5401/96 $18.00

1. INTRODUCTION

Graphical and diagrammatic representations play a central role in several application fields, such as software specification and design (Petri nets, FSAs, statecharts), data flow analysis (data-flow diagrams), concurrency, pattern recognition, visual programming, and visual interfaces (see, e.g., Harel, 1987; Harel, 1988; Jacob, 1985). Indeed, they are widely recognized as powerful aids for the description and understanding of complex systems for which traditional textual descriptions are inadequate. In general, when suitably designed, diagrammatic representations can be easily understood by people regardless of their individual cultures, because the human visual system is more inclined to processing graphical information than textual information.

The above considerations and the decreasing cost of hardware and graphics software have caused the development of a wide variety of systems that make extensive use of icons and diagrams (see, e.g., Angelaccio *et al.*, 1990; Chang, 1990b; Costagliola *et al.*, 1995; Crimi *et al.*, 1990; Glinert and Tanimoto, 1984; Hirakawa *et al.*, 1990). As a consequence, formalisms for the specification of diagrammatic languages are currently of great interest. In particular, notable efforts are devoted to the field of visual languages, where a language is usually conceived as a collection of sentences given by pictorial components which are suitably arranged in two or more dimensions (see, e.g., Chang, 1990a).

A common approach to the description of visual and graphical languages makes use of formal grammars and rewriting mechanisms able to generate this kind of sentence (Crimi *et al.*, 1991; Golin and Reiss, 1990; Marriott, 1994; Wittenburg and Weitzman, 1990; Wittenburg, 1992). Such an approach allows those languages to be handled with general syntactic specification mechanisms. As was the case for traditional programming languages, such formal models provide appropriate methodological frameworks for graphical language design and applications. In particular, they should make tools available that allow for the automatic creation of lexical analyzers, parsers, semantic analyzers, syntax-driven visual programming environments, etc. Following this approach, in this paper we present the formalism of *Symbol–Relation Grammars* (SR grammars, for short) and some results concerning their derivation mechanism, their expressive power, and their practical applications.

In the SR formalism, a sentence is viewed as a set of *symbol occurrences* (s-items in the sequel) and a set of *relational items* over symbol occurrences (r-items in the sequel). The main feature of SR grammars is that the derivation of a sentence is performed by rewriting both symbol occurrences and relational items by means of very simple context-free styled rules. More precisely, during a derivation step a symbol occurrence X^0 in a sentence $S1$ is replaced by a sentence $S2$ according to a rewriting rule of the form $X^0 \rightarrow S2$; called s-item production (s-production). After X^0 has been rewritten, the replacement of the set of r-items involving X^0 is performed through r-item rewriting rules (r-productions) of the form $r(X^0, Y^1) \rightarrow R$, where R is a set of r-items relating Y^1 to s-items in $S2$. In other words, an r-production simply states that, once X^0 has been rewritten, the r-item $r(X^0, Y^1)$ can be replaced by the r-items in R.

Symbol–Relation grammars are proposed as a simple and quite general syntactic framework for graphical language analysis and development. As to their origin, SR grammars were initially devised as a multidimensional extension of string grammars, where several relation names were possible with suitable associated rewriting rules. Indeed, traditional string grammars may be conceived as SR grammars where only the relation *OnTheLeftOf* is present and a set of specific very simple r-item rewriting productions is predefined. The main idea is that the simple uniform style proposed both for s-item and r-item rewriting, together with the formal analogy between SR productions and context-free grammar productions, facilitates the comprehension and the practical use of the formalism. Indeed, such an analogy is also useful to extend well-established analysis methods for traditional context-free string grammars.

The paper shows through several examples, the suitability of the formalism for expressing grammars for nontrivial graphical languages, and extends to SR grammars the notion of syntactic trees, typical of context-free string languages. As a matter of fact, the generation of a sentence by an SR grammar can be described by means of a tree structure, called a *Symbol–Relation tree* (SR tree), where each node represents either a symbol occurrence or a relational item. The concept of an SR tree can be assumed as a quite natural basis for transferring traditional techniques of syntactic and semantic analysis, based on attributes definition and synthesis, to the case of visual and graphical languages. As a result, Symbol–Relation grammar ideas and techniques have been successfully exploited to construct parsers for visual languages (Ferrucci *et al*., 1991; Tucci *et al*., 1994; Ferrucci *et al*., 1994b).

An accurate study of the formalism is in general necessary to fully exploit the capabilities of a language grammar model. Thus, another important goal of the present paper is to analyze the expressive power and the derivation mechanism of the SR formalism. In order to analyze the SR derivation mechanism, we consider some structural aspects concerning SR productions and their application. In particular, we analyze *nondeterministic r-item rewriting* and *r-item collapsing*. The capability of nondeterministic r-item rewriting is a remarkable feature of SR grammars. It means that several alternative ways of rewriting the same r-item are possible during a derivation step. We prove that in general this feature does not increase the generative power of SR grammars, and an r-deterministic normal form exists. Nonetheless, nondeterminism in r-item rewriting can simplify the specification of grammars. Moreover, we show that the generative power of some interesting subclasses of SR grammars is increased by the presence of nondeterministic r-item rewriting. The phenomenon of r-item collapsing occurs when different occurrences of identical r-items, generated during the derivation process, are collapsed. Such a feature can give rise to derivation sequences where the number of relational items presents nonmonotonic behavior. However, we prove that a monotonic normal form exists for SR grammars.

The analysis of the SR derivation process also allows us to study the generative power of the formalism. Indeed, SR grammars are compared with grammar schemas, which are used in computer science for modelling various kinds of diagrams and are widely investigated in the literature from a theoretical viewpoint. It is intuitive that the concept of SR sentence has a natural correspondence with the

abstract concept of graph. On the basis of that correspondence, SR grammars have been compared with well known node rewriting graph grammars. In spite of the quite simple form of rewriting mechanism, the paper shows that variations of SR grammars have a generative power comparable with that of general classes of graph grammars such as the edNCE and the N-edNCE classes (Engelfriet, 1990; Engelfriet *et al.*, 1990).

The rest of this paper has the following organization. In Section 2, the formalism of Symbol–Relation grammar is presented. This section also describes the derivation mechanism of SR grammars, and some examples illustrate the use of the SR formalism for describing complex visual objects. In Section 3 some features of SR grammars are investigated, such as r-item collapsing and the capability of nondeterministic r-item rewriting. Such properties are then analyzed from the generative point of view in Section 4. As a result, some interesting normal forms for SR grammars are given. Section 5 shows how the notion of an attribute context-free grammar can be naturally extended to SR grammars in order to perform the semantic analysis of visual languages by exploiting the concept of an SR tree. The comparison of SR grammars and some well-known classes of node rewriting graph grammars is carried out in Section 6. Section 7 provides an overview of other grammatical approaches for specifying the syntax of visual languages that are related to SR grammars. Some final remarks conclude the paper.

2. SYMBOL–RELATION GRAMMARS

In this section, the formalism of *Symbol–Relation* grammars (SR grammars, for short) is presented. The approach is based on the notion of an *SR sentence*, which is conceived as a set of symbol occurrences together with a set of relationships among them. Multiple occurrences of the same symbol within a sentence are possible. Thus, they are distinguished by different index numbers. These concepts are formalized in the following definitions.

DEFINITION 2.1. Given an alphabet A, the set of *symbol occurrences* (*s-items*, for short) on A is defined as $C_A = A \times N$, where N is the set of natural numbers. For simplicity, the element (X, k) of C_A will be written as X^k.

In the sequel, symbol occurrences will be denoted by the different font X, Y, etc., whenever the occurrence number is inessential for the discussion. This notation is used in Definitions 2.2 and 2.3.

DEFINITION 2.2. Given a set B of relation symbols, an alphabet A, and a subset $\mathbf{M}$ of C_A, a *relational item* (*r-item*, for short) on B and $\mathbf{M}$ has the form $r(\mathsf{X}, \mathsf{Y})$, where $r \in B$, $\mathsf{X} \in \mathbf{M}$, $\mathsf{Y} \in \mathbf{M}$, and $\mathsf{X} \neq \mathsf{Y}$.

The notion of an SR sentence can be now formally defined.

DEFINITION 2.3. Given a set B of relation symbols and an alphabet A, an *SR sentence* on B and A is a pair $\langle \mathbf{M}, \mathbf{R} \rangle$, where $\mathbf{M}$ is a finite nonempty subset of C_A

and **R** is a set of r-items on B and **M**. More explicitly, the general form of a sentence is

$$\begin{aligned}\langle \mathbf{M}, \mathbf{R} \rangle &= \langle \{\mathsf{X}_1, ..., \mathsf{X}_\mu\}, \{r_1(\mathsf{Y}_1, \mathsf{Z}_1), ..., r_\sigma(\mathsf{Y}_\sigma, \mathsf{Z}_\sigma)\} \rangle \\ &= \langle \{X_1^{m_1}, ..., X_\mu^{m_\mu}\}, \{r_1(Y_1^{n_1}, Z_1^{p_1}), ..., r_\sigma(Y_\sigma^{n_\sigma}, Z_\sigma^{p_\sigma})\} \rangle,\end{aligned}$$

where

$$\mu \geqslant 1, \sigma \geqslant 0$$
$$\mathsf{X}_j \in C_A, \quad \text{for} \quad 1 \leqslant j \leqslant \mu$$
$$r_i \in B, \quad \text{for} \quad 1 \leqslant i \leqslant \sigma$$
$$\mathsf{Y}_i, \mathsf{Z}_i \in \mathbf{M}, \quad \text{for} \quad 1 \leqslant i \leqslant \sigma.$$

As an example, let us consider the following sentence:

$$\langle \{t^2, t^4, t^6, t^7, t^5, t^3\}, \{next(t^2, t^4), next(t^4, t^6), next(t^4, t^7), next(t^6, t^5), next(t^7, t^5), loop(t^5, t^4), next(t^5, t^3)\} \rangle.$$

The sentence can be viewed as the representation of the arrangement of blocks of Fig. 2.1. Essentially, the sentence represents a block diagram with sequentialization, cycles, and parallelism. In the set of symbol occurrences, the symbol "t" represents a block. It appears six times in the sentence, each time with a different superscript as occurrence number; the meaning of the relation symbols *next* and *loop* is evident.

It is worth noting that an SR sentence can be easily interpreted as a directed, labelled graph, where the s-items correspond to labelled nodes and the r-items correspond to labelled edges.

An SR grammar is specified by a set of productions that state how to rewrite s-items (*s-productions*) and r-items (*r-productions*). The right-hand side of each s-production consists of an SR sentence, i.e., a set of symbol occurrences and a set of relationships among them. The r-productions allow us to embed the right hand-side of the applied s-production into the host sentence. In other words, r-productions replace r-items containing the rewritten s-item with r-items relating the new s-items to existing ones in the host sentence.

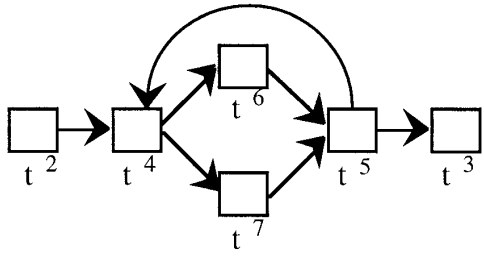

FIGURE 2.1

The above ideas are formalized in the following definition.

DEFINITION 2.4. A Symbol–Relation grammar is a 6-tuple $G = (V_N, V_T, V_R, S, P, R,)$ where:

— V_N is a finite nonempty set of *nonterminal* symbols.

— V_T is a finite nonempty set of *terminal* symbols.

— V_R is a finite set of *relation* symbols.

— $S \in V_N$ is the start symbol.

— P is a finite set of rewriting rules, called *s-item productions* (s-productions, for short), of the form

$$l\colon Y^0 \to \langle \mathbf{M}, \mathbf{R} \rangle$$

where

(a) l is an integer uniquely labelling the s-production,

(b) $\langle \mathbf{M}, \mathbf{R} \rangle$ is a sentence on V_R and $V_N \cup V_T$,

(c) $Y \in V_N$, $Y^0 \notin \mathbf{M}$.

— R is a finite set of rewriting rules, called *r-item productions* (r-productions, for short), of the form

$$s(Y^0, X^1) \to [l]\ \mathbf{Q} \qquad \text{or} \quad s(X^1, Y^0) \to [l]\ \mathbf{Q}$$

where:

(d) $s \in V_R$,

(e) l is the label of an s-production $Y^0 \to \langle \mathbf{M}, \mathbf{R} \rangle$,

(f) $X \in V_N \cup V_T$ and $X^1 \notin \mathbf{M}$,

(g) $\mathbf{Q} \neq \varnothing$ is a finite set of r-items of the form $r(\mathsf{Z}, X^1)$ or $r(X^1, \mathsf{Z})$, with $\mathsf{Z} \in \mathbf{M}$.

As a convention, the apex "0" will only be used to index the symbol on the left-hand side of every s-production and the apex "1" to index the other symbol on the left-hand side of every r-production.

The label l in the right-hand side of an r-production establishes an operative link among s-productions and r-productions; i.e., an r-production

$$p_r = s(Y^0, X^1) \to [l]\ \mathbf{Q}$$

can be applied only after the symbol Y has been rewritten using the s-production l. In the sequel, p_r is said to be *allowed* by the s-production l. As can be seen in several examples in the paper, the same r-production is frequently allowed by more than one s-production. In this case, it is convenient to use a concise notation of the form

$$s(Y^0, X^1) \to [l_1, l_2, ..., l_h]\ \mathbf{Q},$$

which summarizes the set of the h following r-productions:

$$s(Y^0, X^1) \to [l_i]\ \mathbf{Q}, \qquad \text{for} \quad 1 \leqslant i \leqslant h.$$

For the sake of conciseness, we will use such a notation in the examples throughout the paper.

2.1. SR Derivation Process

This section describes the derivation mechanism of SR grammars. Intuitively, given an SR grammar and a sentence $\langle \mathbf{M}, \mathbf{R} \rangle$, a derivation step is performed applying the following rewriting operations:

(1) rewrite an occurrence X of a nonterminal symbol in **M**, applying an s-production p_s;

(2) rewrite each r-item in **R** containing the rewritten symbol occurrence X by using an r-production that is allowed by p_s.

In order to help the intuition, let us consider the following grammar G generating a language of block diagram sentences with sequentialization, nested cycles, and parallelism; a sample sentence has been shown in Fig. 2.1.

EXAMPLE 2.1. Let $G = (\{D, B\}), \{t\}, \{next, loop\}, D, P, R,)$ be an SR grammar, where P contains the s-productions

$$1:\ D^0 \to \langle \{t^2, B^2, t^3\}, \{next(t^2, B^2), next(B^2, t^3)\} \rangle$$
$$2:\ B^0 \to \langle \{B^2, B^3\}, \varnothing \rangle$$
$$3:\ B^0 \to \langle \{B^2, B^3\}, \{next(B^2, B^3)\} \rangle$$
$$4:\ B^0 \to \langle \{t^2, B^2, t^3\}, \{next(t^2, B^2), next(B^2, t^3), loop(t^3, t^2)\} \rangle$$
$$5:\ B^0 \to \langle \{t^2\}, \varnothing \rangle$$

and R contains the r-productions

$$next(B^0, t^1) \to [2]\{next(B^2, t^1), next(B^3, t^1)\}$$
$$next(B^0, t^1) \to [3]\{next(B^3, t^1)\}$$
$$next(B^0, t^1) \to [4]\{next(t^3, t^1)\}$$
$$next(B^0, t^1) \to [5]\{next(t^2, t^1)\}$$

$$next(t^1, B^0) \to [2]\{next(t^1, B^2), next(t^1, B^3)\}$$
$$next(t^1, B^0) \to [3]\{next(t^1, B^2)\}$$
$$next(t^1, B^0) \to [4, 5]\{next(t^1, t^2)\}$$

$$next(B^1, B^0) \to [2]\{next(B^1, B^2), next(B^1, B^3)\}$$
$$next(B^1, B^0) \to [3]\{next(B^1, B^2)\}$$
$$next(B^1, B^0) \to [4, 5]\{next(B^1, t^2)\}.$$

Observe that in the previous example, an r-production with the list [4, 5] in the right-hand side is used to denote two r-productions, the former being allowed by s-production 4 and the latter by s-production 5.

With reference to the previous grammar, let us sketch how a derivation step is performed. Consider the sentence

$$\langle \{t^2, B^2, B^3, t^3\}, \{next(t^2, B^2), next(t^2, B^3), next(B^2, t^3), next(B^3, t^3)\} \rangle,$$

which corresponds to the arrangement of blocks in Fig. 2.2a. Suppose we want to rewrite the symbol occurrence B^2 according to the s-production with label 4. As will be formally stated by Definition 2.6, before a production is applied the occurrence numbers must be systematically changed in order to avoid conflicts with occurrence numbers already present in the original sentence and to establish matching between the rewritten symbol and the left-hand side of the chosen s-production and r-productions. For instance, in the previous case the following renumbered version of the s-production is applied:

$$4: \; B^2 \rightarrow \langle \{t^4, B^4, t^5\}, \{next(t^4, B^4), next(B^4, t^5), loop(t^5, t^4)\} \rangle.$$

The r-items $next(t^2, B^2)$ and $next(B^2, t^3)$, which contain the rewritten s-item B^2 of the original sentence, must be rewritten in turn. The r-productions for the given r-items that are allowed by s-production 4 are those having 4 in the list of labels, namely

$$next(t^1, B^0) \rightarrow [4, 5]\{next(t^1, t^2)\}$$
$$next(B^0, t^1) \rightarrow [4]\{next(t^3, t^1)\}.$$

Then, by suitably renumbering the symbol occurrences, we obtain

$$next(t^2, B^2) \rightarrow [4, 5]\{next(t^2, t^4)\}$$
$$next(B^2, t^3) \rightarrow [4]\{next(t^5, t^3)\}.$$

The derived sentence

$$\langle \{t^2, t^4, B^4, t^5, B^3, t^3\}, \{next(t^2, t^4), next(t^4, B^4), next(B^4, t^5), loop(t^5, t^4), next(t^2, B^3), next(t^5, t^3), next(B^3, t^3)\} \rangle$$

corresponds to the arrangement of blocks in Fig. 2.2b.

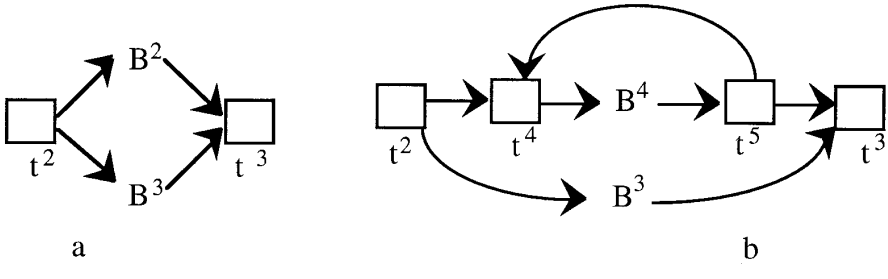

FIG. 2.2. The representation of a derivation step.

The above discussion uses the idea of renumbering of symbol occurrences. Before giving the formal specification of the SR derivation process, let us define the renumbering operation. Renumbering is essentially a function from symbol occurrences on $V_N \cup V_T$ to the set N of natural numbers. A renumbering can be defined as follows.

DEFINITION 2.5 (Renumbering). A renumbering is a function

$$\theta: (V_N \cup V_T) \times N \to N,$$

such that for any symbol $A \in V_N \cup V_T$, if $n \neq m$, then $\theta(A, n) \neq \theta(A, m)$. Given the symbol occurrence A^m, the symbol occurrence $A^{\theta(A, m)}$ is denoted by $A^m\theta$.

The renumbering can be naturally extended to r-items, sets of s-items, etc. as follows:

— given an r-item $r(\mathsf{X}, \mathsf{Y})$, then $r(\mathsf{X}, \mathsf{Y})\,\theta = r(\mathsf{X}\theta, \mathsf{Y}\theta)$;

— given a set $\mathbf{M} = \{\mathsf{X}_1, ..., \mathsf{X}_\mu\}$, then $\mathbf{M}\theta = \{\mathsf{X}_1\theta, ..., \mathsf{X}_\mu\theta\}$.

Now we are ready to give the definition of derivation step.

DEFINITION 2.6. Let G be an SR grammar and $\langle \mathbf{M}', \mathbf{R}' \rangle$, $\langle \mathbf{M}'', \mathbf{R}'' \rangle$ be two SR sentences. $\langle \mathbf{M}', \mathbf{R}' \rangle$ *directly derives* $\langle \mathbf{M}'', \mathbf{R}'' \rangle$, denoted by $\langle \mathbf{M}', \mathbf{R}' \rangle \Rightarrow \langle \mathbf{M}'', \mathbf{R}'' \rangle$, if and only if a symbol occurrence $A^n \in \mathbf{M}'$ and an s-production $l: A^0 \to \langle \mathbf{M}, \mathbf{R} \rangle$ in G exist such that the following conditions hold:

(1) There exists a renumbering θ such that:

(a) $A^0\theta = A^n$, $\mathbf{M}\theta \cap \mathbf{M}' = \varnothing$,

(b) $\mathbf{M}'' = (\mathbf{M}' - \{A^n\}) \cup \mathbf{M}\theta$.

(2) Let $\{s_i(\mathsf{X}_i, \mathsf{Y}_i) \mid 1 \leqslant i \leqslant m\}$ be the set of all r-items in $\mathbf{R}'$ such that either $\mathsf{X}_i = A^n$ or $\mathsf{Y}_i = A^n$. There exist r-productions $s_i(\mathsf{T}_i, \mathsf{V}_i) \to [l]\ \mathbf{Q}_i$ and renumberings θ_i, $1 \leqslant i \leqslant m$, such that:

(c) $\mathsf{B}\theta_i = \mathsf{B}\theta$ for every $\mathsf{B} \in \{A^0\} \cup \mathbf{M}$, $1 \leqslant i \leqslant m$,

(d) $s_i(\mathsf{T}_i, \mathsf{V}_i)\,\theta_i = s_i(\mathsf{X}_i, \mathsf{Y}_i)$, $1 \leqslant i \leqslant m$,

(e) $\mathbf{R}'' = (\mathbf{R}' - \{s_i(\mathsf{X}_i, \mathsf{Y}_i) \mid 1 \leqslant i \leqslant m\}) \cup \mathbf{R}\theta \cup (\bigcup_{1 \leqslant i \leqslant m} (\mathbf{Q}_i, \theta_i))$.

Let $G = (V_N, V_T, V_R, S, P, R)$ be an SR grammar. We say that an r-item $r(\mathsf{X}, \mathsf{Y})$ is *terminal* if $\mathsf{X}, \mathsf{Y} \in C_{V_T}$. A sentence $\langle \mathbf{M}, \mathbf{R} \rangle$ is called *terminal* if $\mathbf{M} \subseteq C_{V_T}$, i.e., all the symbols occurring in $\mathbf{M}$ are terminal symbols and $\mathbf{R}$ contains only terminal r-items. As usual, $\Rightarrow^*$ denotes the reflexive and transitive closure of the relation $\Rightarrow$. The *language* generated by G can then be defined as

$$L(G) = \{\langle \mathbf{M}, \mathbf{R} \rangle \mid \langle \{S^0\}, \varnothing \rangle \Rightarrow^* \langle \mathbf{M}, \mathbf{R} \rangle \text{ and } \langle \mathbf{M}, \mathbf{R} \rangle \text{ is a terminal sentence.}\}$$

EXAMPLE 2.2. The terminal sentence of Fig. 2.1 is generated according to the SR grammar of Example 2.1 through the application of the following sequence of s-productions and corresponding allowed r-productions:

$\langle \{D^0\}, \varnothing \rangle \Rightarrow$ applying s-production 1 to D^0
$\langle \{t^2, B^2, t^3\}, \{next(t^2, B^2), next(B^2, t^3)\} \rangle \Rightarrow$ applying s-production 4 to B^2
$\langle \{t^2, t^4, B^3, t^5, t^3\}, \{next(t^2, t^4), next(t^4, B^3), next(B^3, t^5),$
$loop(t^5, t^4), next(t^5, t^3)\} \rangle \Rightarrow$ applying s-production 2 to B^3
$\langle \{t^2, t^4, B^4, B^5, t^5, t^3\}, \{next(t^2, t^4), next(t^4, B^4), next(t^4, B^5), next(B^4, t^5),$
$next(B^5, t^5), loop(t^5, t^4), next(t^5, t^3)\} \rangle \Rightarrow$ applying s-production 5 to B^4
$\langle \{t^2, t^4, t^6, B^5, t^5, t^3\}, \{next(t^2, t^4), next(t^4, t^6), next(t^4, B^5), next(t^6, t^5),$
$next(B^5, t^5), loop(t^5, t^4), next(t^5, t^3)\} \rangle \Rightarrow$ applying s-production 5 to B^5
$\langle \{t^2, t^4, t^6, t^7, t^5, t^3\}, \{next(t^2, t^4), next(t^4, t^6), next(t^4, t^7), next(t^6, t^5),$
$next(t^7, t^5), loop(t^5, t^4), next(t^5, t^3)\} \rangle$.

2.2. Examples

The examples in this section illustrate the use of the SR formalism for describing complex visual objects. The first example deals with logic sequential networks. A sequential network is structured as a combinatorial network together with a set of memory components, the flip-flops. The combinatorial network is made of a set of combinatorial components, which are connected without any feedback. In a sequential network generated by the following grammar, feedbacks can only be established by the connection of output ports to input ports of the combinatorial network through flip-flops.

EXAMPLE 2.3. Sequential networks are modeled by the grammar $Seq\_Net = (\{S\}, \{\text{cc, i-port, o-port, ff}\}, \{conn\}, S, P, R)$.

The set P contains the s-productions

1: $S^0 \to \langle \{S^2, \text{i-port}^2, \text{ff}^2\}, \{conn(S^2, \text{ff}^2), conn(\text{i-port}^2, S^2), conn(\text{ff}^2, \text{i-port}^2)\} \rangle$
2: $S^0 \to \langle \{S^2, \text{i-port}^2\}, \{conn(\text{i-port}^2, S^2)\} \rangle$
3: $S^0 \to \langle \{S^2, \text{cc}^2\}, \{conn(\text{cc}^2, S^2)\} \rangle$
4: $S^0 \to \langle \{S^2, \text{cc}^2, \text{o-port}^2\}, \{conn(\text{cc}^2, \text{o-port}^2)\} \rangle$
5: $S^0 \to \langle \{S^2, \text{cc}^2, \text{o-port}^2\}, \{conn(\text{cc}^2, \text{o-port}^2), conn(\text{cc}^2, S^2)\} \rangle$
6: $S^0 \to \langle \{\text{o-port}^2, \text{cc}^2\}, \{conn(\text{cc}^2, \text{o-port}^2)\} \rangle$

and R contains the r-productions

$conn(S^0, \text{ff}^1) \to [4, 5, 6]\{conn(\text{o-port}^2, \text{ff}^1)\}$
$conn(S^0, \text{ff}^1) \to [1, 2, 3, 4, 5]\{conn(S^2, \text{ff}^1)\}$
$conn(\sigma^1, S^0) \to [1, 2, 3, 4, 5]\{conn(\sigma^1, S^2)\}$
$conn(\sigma^1, S^0) \to [3, 4, 5, 6]\{conn(\sigma^1, \text{cc}^2)\}$
$conn(\sigma^1, S^0) \to [3, 4, 5]\{conn(\sigma^1, S^2), conn(\sigma^1, \text{cc}^2)\}$ where $\sigma = \text{i-port, cc}$.

Note that the meta-symbol σ is used to denote any of the terminal symbols *i-port* or *cc* since the corresponding r-productions behave in the same fashion. The terminal symbols of the grammar *Seq_Net* have the following meanings: *i-port* is an input port, *o-port* is an output port, *cc* is a combinatorial component, *ff* is a flip-flop, and finally *conn* is a connection between components. Figure 2.3 gives a graphical representation of the s-productions of the grammar. Symbols in dotted line stand for the left hand sides of s-productions. Right-hand sides are given a

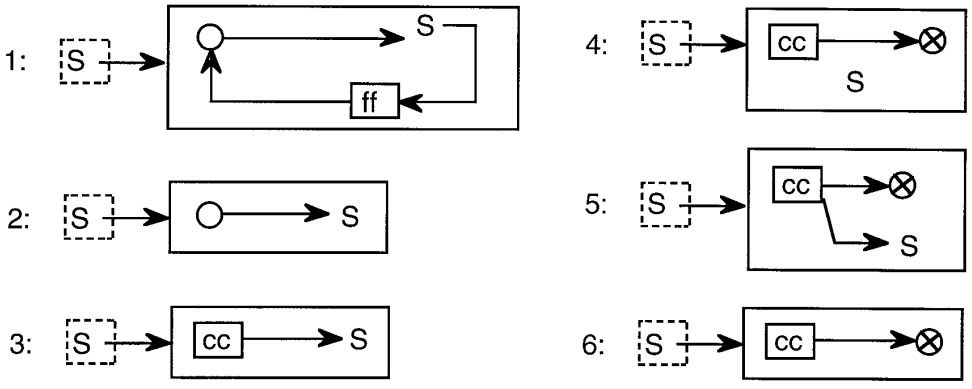

FIG. 2.3. A graphical representation for the s-productions of the grammar *Seq_Net*.

visual presentation. Input and output ports are drawn as circles and crossed circles, respectively. Since no explicit symbol is present in the grammar for ramification points of signals exiting combinatorial components, they are considered implicit in the fact that several r-items may exit the same symbol occurrence (see s-production 5). In any case, the introduction of a terminal symbol which explicitly represents a ramification point could be treated in analogy with the symbol i-port.

It is worth noting that, if the first s-production and the first two r-productions are excluded, the remaining rules define a grammar which generates combinatorial networks without feedbacks. In fact, the complete grammar can be used in two phases. In the first phase only s-productions (1) and (2) are exploited (with the corresponding allowed r-productions) in order to produce a sentential form where all the flip-flops and the input ports of the combinatorial network are already present. At this stage, the combinatorial network (apart from its input ports) is represented by a nonterminal symbol of type *S*. In the second phase, the internal structure of the combinatorial network is produced together with its output ports. Figure 2.4 shows the process of generating a sequential network according to the given grammar *Seq_Net*. For each derivation step, an apex number indicates which s-production is applied.

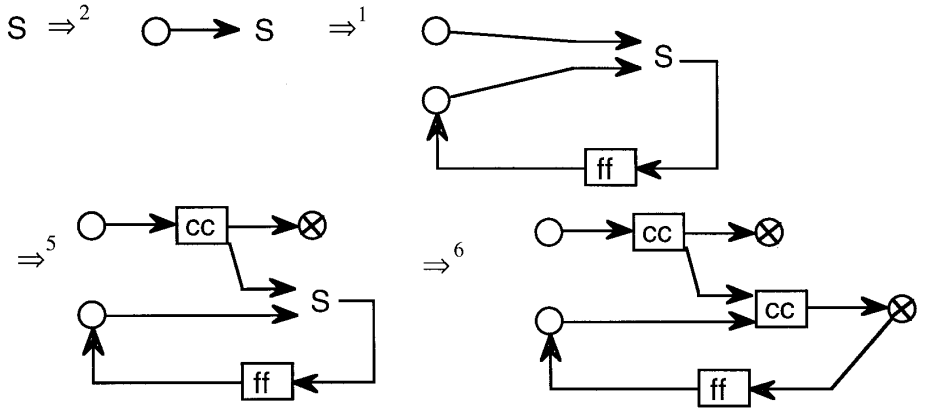

FIG. 2.4. A sequential network generated by the grammar *Seq_Net*.

The second grammar in this section is an enhanced version of the grammar given in Example 2.1. It generates structured flowcharts with any number of nested *if–then–else* and *while* structures for integrated visual programming environments (Costagliola *et al.*, 1995). The generated flowcharts can easily be translated into Pascal-like programs. In Section 5 we will show how well-known methods, based on attributes which are synthesized in agreement with the development of the syntactical analysis phase, can be applied to SR grammars in order to perform semantic analysis and code generation.

EXAMPLE 2.4. The following SR grammar *Fl_Ch* defines a class of structured flowcharts. The symbol *simple* is used to represent any elementary statement, while the symbol *cond* represents a condition. A *next* relation is used to indicate (control-flow) connections between nodes of a flowchart. *If–then–else* and *while* structures are produced by s-productions 4 and 5, respectively:

Fl_Ch $= (\{S, F\}, \{\text{start, halt, cond, simple}\}, \{next\}, S, P, R)$, where P contains the s-productions:

1: $S^0 \rightarrow \langle \{\text{start}^2, F^2, \text{halt}^2\}, \{\text{next}(\text{start}^2, F^2), \text{next}(F^2, \text{halt}^2)\} \rangle$

2: $F^0 \rightarrow \langle \{F^2, F^3\}, \{\text{next}(F^2, F^3)\} \rangle$

3: $F^0 \rightarrow \langle \{\text{simple}^2\}, \varnothing \rangle$

4: $F^0 \rightarrow \langle \{\text{cond}^2, F^2, F^3\}, \{\text{next}(\text{cond}^2, F^2), \text{next}(\text{cond}^2, F^3)\} \rangle$

5: $F^0 \rightarrow \langle \{\text{cond}^2, F^2\}, \{\text{next}(\text{cond}^2, F^2), \text{next}(F^2, \text{cond}^2)\} \rangle$.

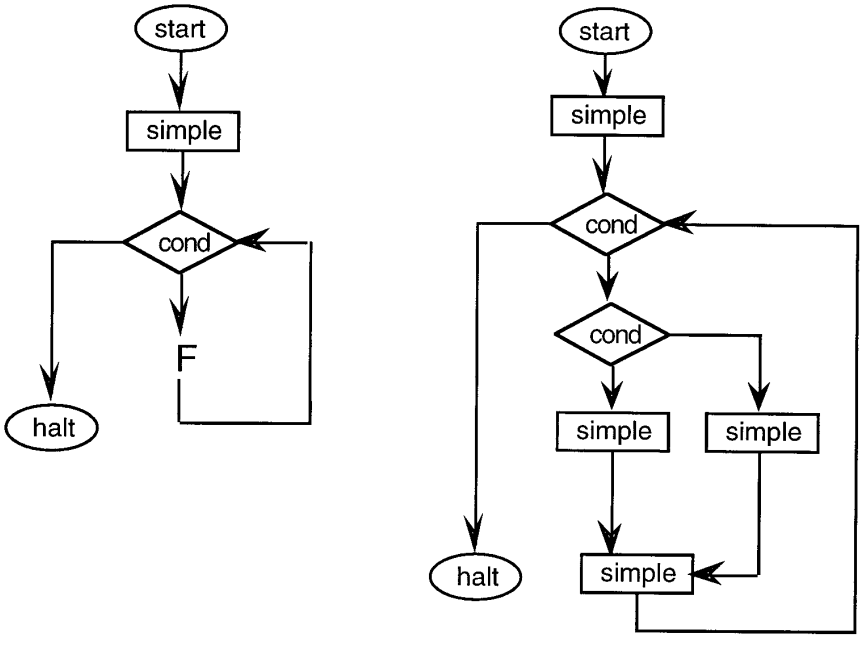

FIG. 2.5. (a) An intermediate sentence generated during a derivation and (b) the flowchart corresponding to the generated terminal sentence.

The set of r-productions R is composed of:

$\text{next}(\tau^1, F^0) \to [2]\{\text{next}(\tau^1, F^2)\}$
$\text{next}(\tau^1, F^0) \to [3]\{\text{next}(\tau^1, \text{simple}^2)\}$
$\text{next}(\tau^1, F^0) \to [4, 5]\{\text{next}(\tau^1, \text{cont}^2)\}$ where $\tau = \text{cond, simple, start, F}$
$\text{next}(F^0, \sigma^1) \to [2]\{\text{next}(F^3, \sigma^1)\}$
$\text{next}(F^0, \sigma^1) \to [3]\{\text{next}(\text{simple}^2, \sigma^1)\}$
$\text{next}(F^0, \sigma^1) \to [4]\{\text{next}(F^2, \sigma^1), \text{next}(F^3, \sigma^1)\}$
$\text{next}(F^0, \sigma^1) \to [5]\{\text{next}(\text{cond}^2, \sigma^1)\}$ where $\sigma = \text{cond, simple, halt, F.}$

The meta-symbol τ is used to denote any of the terminal symbols *cond*, *simple*, *start*, or F; analogously, the meta-symbol σ is used to denote any of the terminal symbols *cond*, *simple*, *halt*, or F in the remaining r-productions. A flowchart generated by *Fl_Ch* is shown in Fig. 2.5b. Elementary statements are denoted by rectangles, diamond boxes stand for conditions, and *next* relationships are indicated by arrows. Fig. 2.5a shows the intermediate sentence generated by applying s-productions 1, 2, 3, and 5. The terminal sentence of Fig. 2.5b is then obtained by the sequence of s-productions 2, 4, 3, 3, and 3.

3. SOME FEATURES OF SR GRAMMARS

In order to get insight into the structure of the SR derivation process, in this section some features of SR grammars are investigated. In particular, we analyze three aspects concerning the SR derivation process: *r-item rewriting failure*, the capability of *nondeterministic r-item rewriting*, and *r-item collapsing*. The results of such analysis are useful to determine the aspects that actually affect the generative power of SR grammars and allow us to fully exploit the capabilities of the formal model in the specification and analysis of visual languages.

3.1. r-Item Rewriting Failure

According to the definition of SR derivation step, when an s-production p_s is applied, all the r-items containing the replaced s-item must be rewritten using r-productions allowed by p_s. It may happen that no applicable r-productions exist for at least one of the r-items containing the rewritten s-item. Such an event is called r-item rewriting failure (*r-rewriting failure*, for short). If an r-item rewriting failure occurs, the derivation step cannot be brought to an end. Thus, the s-production itself is considered as not applicable. This feature can be used to generate "context-sensitive" languages, as discussed in this section.

In order to illustrate the effects of r-item rewriting failure on the generation process of SR grammars, let us consider a grammar generating the language consisting of all the SR sentences containing exactly 2^n s-items, $n \geqslant 0$, labelled by the same symbol and "chained" by a relation r.

EXAMPLE 3.1. Consider the SR grammar $G = (\{H\}, \{a\}, \{sl, nsl, r\}, H, P, R)$, where P consists of the s-productions

1: $H^0 \to \langle \{H^2, H^3\}, \{sl(H^2, H^3)\} \rangle$
2: $H^0 \to \langle \{a^2\}, \varnothing \rangle$

and R consists of the r-productions

$$sl(H^1, H^0) \to [1]\{nsl(H^1, H^2)\}$$
$$nsl(H^0, H^1) \to [1]\{sl(H^3, H^1)\}$$
$$sl(H^1, H^0) \to [2]\{nsl(H^1, a^2)\}$$
$$nsl(H^0, a^1) \to [2]\{r(a^2, a^1)\}.$$

It is not difficult to see that in every sentential form $\langle \mathbf{M}, \mathbf{R} \rangle$ generated by G we can order the elements of $\mathbf{M}$ in such a way that every two consecutive elements are related by exactly one r-item in $\mathbf{R}$. Note that $\langle \mathbf{M}, \mathbf{R} \rangle$ resembles a string structure. Consider now a sentential form of G involving only occurrences of the nonterminal symbol H and r-items of the form $sl(H^i, H^j)$. Observe that s-production 1 can only be applied to the last s-item w.r.t. our ordering, since an r-item of the form $sl(H^0, H^1)$ gives rise to an r-rewriting failure when s-production 1 is applied. After this first application of s-production 1, the only possible rewriting step is the application of the same s-production to the s-item of $\mathbf{M}$ which precedes the s-item rewritten at the previous step, and so forth. This process "doubles" all the s-items in $\mathbf{M}$. Note that the use of the r-rewriting failure feature is crucial here. A similar argument holds for the application of s-production 2 above. By using the previous observations and proceeding by induction, it is not difficult to show that $L(G) = \{\langle \mathbf{M}, \mathbf{R} \rangle \mid \mathbf{M} = \{a^{k_i} \mid 1 \leqslant i \leqslant 2^n, n \geqslant 0\}$ and $\mathbf{R} = \{r(a^{k_{i-1}}, a^{k_i}) \mid 2 \leqslant i \leqslant 2^n\}\}$.

The above example points out that the success of the r-item rewriting phase acts as a condition for the applicability of an s-production. As a result, the r-rewriting failure can be used to generate languages that are context-sensitive. In particular, any context-sensitive string language can be generated by an SR grammar, as stated below.

In what follows, a string $w = a_1 \cdots a_n$, $n \geqslant 1$, will be implicitly represented by any SR sentence $\langle \mathbf{M}, \mathbf{R} \rangle$ such that $\mathbf{M} = \{a_i^{k_i} \mid k_i \geqslant 1, k_i \neq k_j$ for $a_i = a_j, 1 \leqslant i < j \leqslant n\}$ and $\mathbf{R} = \{left(a_{i-1}^{k_{i-1}}, a_i^{k_i}) \mid 2 \leqslant i \leqslant n\}$, where the usual string concatenation relation is represented by the relation symbol *left*. Theorem 3.1 below states that SR grammars can generate any context-sensitive string language. The proof technique crucially exploits the r-rewriting failure feature in the following way. Each context-sensitive string production can be reduced to productions of the form $AB \to AC$, where A, B, and C are nonterminal symbols (see Penttonen, 1974). Such a production is then expressed as an s-production having the form $l: B^0 \to \langle \{C^2\}, \varnothing \rangle$, which corresponds to the context-free string production $B \to C$. In addition, a set of r-productions allowed by l is introduced, which are able to recover contextual restrictions using the r-rewriting failure. More precisely, these r-productions will cause an r-rewriting failure whenever the s-production l is applied to an s-item B which is not immediately to the right of an s-item A. The complete proof of the theorem is deferred to Appendix A.

THEOREM 3.1. *For any context-sensitive language L, an SR grammar G' exists such that $L = L(G')$.*

An analogous result has been established for NLC graph grammars by Janssens and Rozenberg (1980b). However, they use a different mechanism to define the generated language. Indeed, if G is an NLC grammar then the string language

generated by G is defined by intersecting the set of terminal graphs generated by G with the family of oriented chains.

As a consequence of the previous theorem, several undecidability results for SR grammars are inherited from context-sensitive string grammars. In particular:

COROLLARY 3.1. *The emptiness problem for SR grammars is undecidable.*

In the next definition a class of SR grammars is introduced, the *complete* SR grammars, for which no r-rewriting failure ever occurs during a derivation, independent of the chosen s-productions. The grammar of Example 2.1 is not complete since no r-production is given which has $next(B^0, B^1)$ as left-hand side, whereas the grammars *Seq_Net* and *Fl_Ch* of Examples 2.3 and 2.4, respectively, are complete.

DEFINITION 3.1. An SR grammar $G = (V_N, V_T, V_R, S, P, R)$ is a *complete SR grammar* (C-SR grammar, for short) if the following condition holds. For each s-production p_s in P, which rewrites some nonterminal A, and for each r-item $r(X^i, A^j)$ (resp. $r(A^j, X^i)$) appearing in the right-hand side of either an s-production or an r-production of G, there exists at least one r-production allowed by p_s with left-hand side $r(X^1, A^0)$ (resp. $r(A^0, X^1)$).

In Section 6 it will be shown that the condition expressed in the above definition reduces the generative power of SR grammars.

3.2. Nondeterministic r-Item Rewriting

A particular feature of SR grammars is the capability of nondeterministic r-item rewriting. Indeed, in an SR grammar, given an s-production, there might be more than one allowed r-production for the same r-item. As a consequence, several alternative ways of rewriting r-items are possible after an s-production is applied. For example, let us consider the following grammar, which generates the language of all SR sentences on $V_T = \{a\}$ and $V_R = \{r\}$. The grammar exhibits the nondeterministic r-item rewriting capability. Indeed, for r-items of the form $r(a^1, S^0)$ or $r(S^0, a^1)$, three rewriting rules are allowed by each of the s-productions 1, 2, 3, and 4.

EXAMPLE 3.2. Let $All-s = (\{S\}, \{a\}, \{r\}, S, P, R)$ be an SR grammar, where P is specified as follows:

$$
\begin{aligned}
&1:\ S^0 \to \langle \{a^2, S^2\}, \varnothing \rangle \\
&2:\ S^0 \to \langle \{a^2, S^2\}, \{r(a^2, S^2)\} \rangle \\
&3:\ S^0 \to \langle \{a^2, S^2\}, \{r(a^2, S^2), r(S^2, a^2)\} \rangle \\
&4:\ S^0 \to \langle \{a^2, S^2\}, \{r(S^2, a^2)\} \rangle \\
&5:\ S^0 \to \langle \{a^2\}, \varnothing \rangle
\end{aligned}
$$

and R is specified as follows:

$$
\begin{aligned}
&r(a^1, S^0) \to [1, 2, 3, 4, 5]\{r(a^1, a^2)\} \\
&r(a^1, S^0) \to [1, 2, 3, 4]\{r(a^1, a^2), r(a^1, S^2)\} \\
&r(a^1, S^0) \to [1, 2, 3, 4]\{r(a^1, S^2)\} \\
&r(S^0, a^1) \to [1, 2, 3, 4, 5]\{r(a^2, a^1)\} \\
&r(S^0, a^1) \to [1, 2, 3, 4]\{r(a^2, a^1), r(S^2, a^1)\} \\
&r(S^0, a^1) \to [1, 2, 3, 4]\{r(S^2, a^1)\}.
\end{aligned}
$$

Suppose that a symbol occurrence S^i in a sentence $\langle \mathbf{M}, \mathbf{R} \rangle$ has been rewritten by an s-production other than 5. Any r-item referring to S^i can be rewritten exploiting any of the three allowed r-productions. Note that, if two r-items $r(a^k, S^i)$ and $r(a^h, S^i)$ are in $\mathbf{R}$, different r-productions can be chosen freely for rewriting them. As a consequence, the new occurrence of S can be linked in all possible ways to the s-items which were related to the replaced occurrence S^i.

We now introduce the definition of an *r-deterministic SR grammar*. In an r-deterministic SR grammar, for any s-production p_s, all the r-productions allowed by p_s have distinct left-hand sides. If an SR grammar is r-deterministic, at each derivation step the replaced symbol occurrence and the exploited s-production are sufficient for determining the obtained sentence, since no alternative is possible in the r-item rewriting phase. The grammars of Examples 2.1 and 2.4 are instances of r-deterministic SR grammars, while the grammar *Seq_Net* exhibits nondeterministim in r-item rewriting (see for example the r-productions allowed by s-production 4 and having $conn(S^0, \text{ff}^1)$ as left-hand side).

DEFINITION 3.2. An SR grammar G is r-*deterministic* if, for any pair of r-productions p'_r and p''_r, both allowed by the same s-production, the left-hand side of p'_r is different from the left-hand side of p''_r.

Nondeterministic r-item rewriting can be important for a simple and concise description of languages for which equivalent r-deterministic grammars are less immediate. In Section 4, we prove that an r-deterministic normal form exists for SR grammars. As a matter of fact, we show that the elimination of nondeterministic r-rewriting can be done by means of unit s-productions, i.e., productions which replace a nonterminal symbol by another (single) nonterminal symbol. As a consequence, the transformed grammar may in general loose its original clearness. In Section 4 an r-deterministic grammar will be presented, which is equivalent to that of Example 3.2.

The grammar of the next example shows a combined use of nondeterministic rewriting and r-rewriting failure. The grammar generates all the SR sentences on $V_T = \{\text{white, red, green}\}$ and $V_R = \{r\}$ such that no relationships hold between two different occurrences of the same symbol. If we interpret symbols as colors associated with the countries of a map and r as an adjacency relation, the grammar generates three-colorable maps.

EXAMPLE 3.3 (Three-Colorable Maps). $3-c = (\{S, N\}, \{\text{white, red, green}\}, \{r\}, S, P, R)$ is an SR grammar, where P is specified as follows:

$$1:\ S^0 \to \langle \{S^2, N^2\}, \{r(N^2, S^2)\} \rangle$$
$$2:\ S^0 \to \langle \{S^2, N^2\}, \varnothing \rangle$$
$$3:\ S^0 \to \langle \{N^2\}, \varnothing \rangle$$
$$4:\ N^0 \to \langle \{\text{white}^2\}, \varnothing \rangle$$
$$5:\ N^0 \to \langle \{\text{red}^2\}, \varnothing \rangle$$
$$6:\ N^0 \to \langle \{\text{green}^2\}, \varnothing \rangle$$

and R is specified as follows:

$$
\begin{aligned}
r(N^1, S^0) &\to [1, 2, 3]\{r(N^1, N^2)\}\\
r(N^1, S^0) &\to [1, 2]\{r(N^1, N^2), r(N^1, S^2)\}\\
r(N^1, S^0) &\to [1, 2]\{r(N^1, S^2)\}\\
r(N^0, N^1) &\to [4]\{r(\text{white}^2, N^1)\}\\
r(N^0, N^1) &\to [5]\{r(\text{red}^2, N^1)\}\\
r(N^0, N^1) &\to [6]\{r(\text{green}^2, N^1)\}\\
r(\text{white}^1, N^0) &\to [5]\{r(\text{white}^1, \text{red}^2)\}\\
r(\text{white}^1, N^0) &\to [6]\{r(\text{white}^1, \text{green}^2)\}\\
r(\text{red}^1, N^0) &\to [4]\{r(\text{red}^1, \text{white}^2)\}\\
r(\text{red}^1, N^0) &\to [6]\{r(\text{red}^1, \text{green}^2)\}\\
r(\text{green}^1, N^0) &\to [4]\{r(\text{green}^1, \text{white}^2)\}\\
r(\text{green}^1, N^0) &\to [5]\{r(\text{green}^1, \text{red}^2)\}.
\end{aligned}
$$

If one attempts to specify a sentence where two adjacent regions have the same color, an r-rewriting failure occurs in the corresponding derivation. Consider, for instance, the derivation

$$
\begin{aligned}
\langle \{S^0\}, \varnothing \rangle &\Rightarrow \langle \{S^2, N^2\}, \{r(N^2, S^2)\} \rangle\\
&\Rightarrow \langle \{N^3, N^2\}, \{r(N^2, N^3)\} \rangle\\
&\Rightarrow \langle \{N^3, \text{red}^2\}, \{r(\text{red}^2, N^3)\} \rangle.
\end{aligned}
$$

The s-production 5 cannot be applied to the s-item N^3, since there is no r-production allowed by it that can rewrite $r(\text{red}^2, N^3)$.

3.3. *r-Item Collapsing and Monotonic Grammars*

Let us observe that no explicit deletion of s-items or r-items is allowed by SR productions during a derivation step. Nevertheless, the derivation process is not monotonic in the number of r-items in the SR sentence. Indeed, given a sentence $\langle \mathbf{M}', \mathbf{R}' \rangle$ which is directly derived from $\langle \mathbf{M}, \mathbf{R} \rangle$, it may happen that $\text{card}(\mathbf{R}') < \text{card}(\mathbf{R})$. This is due to the fact that, in a derivation step, identical r-items can be generated by rewriting different r-items. In this case, the identical r-items are "collapsed," i.e., only one of them appears in the resulting sentence $\langle \mathbf{M}', \mathbf{R}' \rangle$, since $\mathbf{R}'$ is a set of r-items. In the sequel such a phenomenon will be referred to as r-item collapsing.

Of course, two identical r-items can only be generated by rewriting two different r-items which relate the same couple of s-items, defined as competing r-items in the following.

DEFINITION 3.3. Two r-items $r(\mathsf{X}, \mathsf{Y})$ and $s(\mathsf{V}, \mathsf{Z})$ are said to be *competing* whenever they are not terminal and $(\mathsf{X}, \mathsf{Y}) \in \{(\mathsf{V}, \mathsf{Z}), (\mathsf{Z}, \mathsf{V})\}$.

For instance, in the sequence $\langle \mathbf{M}, \mathbf{R} \rangle = \langle \{A^1, B^1, A^2, a^1, b^1\}, \{r(B^1, A^2), s(B^1, A^2), s(A^1, A^2), r(B^1, a^1), r(a^1, b^1), s(a^1, B^1)\} \rangle$, $r(B^1, A^2)$ competes with $s(B^1, A^2)$ and $r(B^1, a^1)$ competes with $s(a^1, B^1)$. An s-production $l: A^0 \to \langle \mathbf{M}, \mathbf{R} \rangle$

is called *noncompeting* if **R** does not contain competing r-items. Similarly, an r-production $r(A^0, X^1) \to [l]\,\mathbf{Q}$ is called noncompeting if **Q** does not contain competing r-items.

DEFINITION 3.4. An SR grammar is called *monotonic* if all its s-productions and r-productions are noncompeting.

In Section 4.1. it will be shown that every SR grammar can be cast in monotonic form. Here, we show that for monotonic SR grammars the number of r-items never decreases during the derivation process.

PROPOSITION 3.1. *Let G be a monotonic SR grammar and let* $\langle \mathbf{M}, \mathbf{R} \rangle$ *be a sentence of G. If* $\langle \mathbf{M}', \mathbf{R}' \rangle$ *is derived from* $\langle \mathbf{M}, \mathbf{R} \rangle$, *then* $\text{card}(\mathbf{R}) \leqslant \text{card}(\mathbf{R}')$.

Proof. By definition, the right-hand side of any r-production is not empty; that is, no deletion can ever occur in a derivation step. Thus, the number of r-items can decrease only by rewriting two different r-items with two identical r-items. This happens only if the two r-items are derived from two competing r-items. One can easily verify that no competing r-items can be found in a sentence of *G*, since derivation steps in a monotonic SR grammar never introduce competing r-items. This concludes the proof. ▌

4. NORMAL FORMS FOR SR GRAMMARS

The properties of the SR derivation process which have been emphasized in the previous section are now analyzed from the generative power point of view. As a result, some interesting normal forms for SR grammars are given.

Concerning the nondeterminism of r-item rewriting, we have shown that it is useful for achieving a simple and concise description of languages. However, in this section we prove that this feature does not affect the generative power of SR grammars, since an r-deterministic normal form exists. This is also the case for the r-item collapsing phenomenon. Indeed, it is proved that every SR grammar can be cast into a monotonic form, where r-item collapsing never occurs.

The results about normal forms are useful not only to show the degree of flexibility of SR grammars but also to prove nice properties, such as the existence of a tree structure describing the derivation process (see Section 5) and the equivalence of SR grammars to significant classes of graph grammars (see Section 6).

4.1. Monotonic Normal Form

Let us recall that the static condition which defines monotonic SR grammars is the absence of competing productions. We have already shown in Proposition 3.1 that in the derivation of a monotonic SR grammar the number of r-items in the generated SR sentences cannot decrease. In this section, we prove that for any SR grammar there exists an equivalent SR grammar in monotonic form and such that the completeness property is preserved. The basic idea in the proof is to encode a set of competing r-items of an SR grammar *G* by means of a single r-item. To do

this, we introduce new relation symbols having the form $[\Delta, \Delta']$, where Δ and Δ' are sets of relation symbols of G. In this way, an r-item $[\Delta, \Delta'](X^i, Y^j)$ encodes the set composed of the r-items $r(X^i, Y^j)$ and $r'(Y^j, X^i)$, for every $r \in \Delta$ and $r' \in \Delta'$. Note that $[\Delta, \Delta'](X^i, Y^j)$ and $[\Delta', \Delta](Y^j, X^i)$ encode the same set. This technical problem is overcome by fixing an arbitrary order on the set of all s-items, as will be specified below.

The monotonic grammar is obtained by modifying the s-productions of G using the above encoding. Thus, the r-productions must be transformed accordingly. More precisely, r-productions for r-items $[\Delta, \Delta']$ are constructed exploiting all possible combinations of r-productions of G for relation symbols in Δ and Δ'. The new relation symbols are unfolded into the original ones, only when terminal s-items are generated.

THEOREM 4.1. *Let L be a language generated by an SR grammar G. Then L can be generated by a monotonic SR grammar G'. Moreover, if G is complete then G' is also complete.*

Proof. Let $G = (V_N, V_T, V_R, S, P, R)$. In order to specify G', we introduce some additional notation. We define $V'_R = \{[\Delta, \Delta'] \mid \Delta \cup \Delta' \subseteq V_R, \Delta \cup \Delta' \neq \varnothing\}$. We also fix an arbitrary order $<$ on $C_{V_N \cup V_T}$; this is used below to uniquely encode sets of competing r-items of G. Let $\mathbf{M}$ be a set of symbol occurrences in $C_{V_N \cup V_T}$ and let $\mathbf{R}$ be a set of r-items on V_R and $\mathbf{M}$. We write $\sigma(\mathbf{M}, \mathbf{R})$ to denote the set of r-items on V'_R and $\mathbf{M}$ specified as

$$\sigma(\mathbf{M}, \mathbf{R}) = \{[\Delta, \Delta'](\mathsf{X}, \mathsf{Y}) \mid [\Delta, \Delta'] \in V'_R, \mathsf{X}, \mathsf{Y} \in \mathbf{M}, \mathsf{X} < \mathsf{Y}, \mathsf{X} \in C_{V_N} \text{ or } \mathsf{Y} \in C_{V_N},$$
$$r \in \Delta \Leftrightarrow r(\mathsf{X}, \mathsf{Y}) \in \mathbf{R}, r \in \Delta' \Leftrightarrow r(\mathsf{Y}, \mathsf{X}) \in \mathbf{R}\}$$
$$\cup \{r(\mathsf{X}, \mathsf{Y}) \mid r(\mathsf{X}, \mathsf{Y}) \in \mathbf{R}, \mathsf{X} \in C_{V_T} \text{ and } \mathsf{Y} \in C_{V_T}\}.$$

As an example, assuming $A^1 < B^1 < a^1 < b^1$, for an *SR* sentence $\langle \mathbf{M}, \mathbf{R} \rangle = \langle \{A^1, B^1, a^1, b^1\}, \{r(A^1, B^1), s(B^1, A^1), r(a^1, A^1), s(b^1, a^1)\} \rangle$ we have $\sigma(\mathbf{M}, \mathbf{R}) = \{[\{r\}, \{s\}](A^1, B^1), [\varnothing, \{r\}](A^1, a^1), s(b^1, a^1)\}$. From the definition of $\sigma(\mathbf{M}, \mathbf{R})$, it directly follows that:

(i) no competing r-items are found in the set $\sigma(\mathbf{M}, \mathbf{R})$; and

(ii) the terminal r-items in $\sigma(\mathbf{M}, \mathbf{R})$ are all and only the terminal r-items in $\mathbf{R}$.

We can now specify a grammar $G' = (V_N, V_T, V_R \cup V'_R, S, P', R')$ which is monotonic and such that $L(G') = L(G)$. Sets P' and R' are constructed as follows. Initially, let $P' = R' = \varnothing$. For each s-production $p_s = l: (A^0 \to \langle \mathbf{M}, \mathbf{R} \rangle) \in P$, perform the following three steps:

(1) add to P' the s-production $p_s{}' = l': A^0 \to \langle \mathbf{M}, \sigma(\mathbf{M}, \mathbf{R}) \rangle$;

(2) for every possible choice of

- $Y \in V_N \cup V_T$,
- $[\Delta, \Delta'] \in V'_R$, with $\Delta = \{r_1, ..., r_p\}$, $p \geqslant 0$, $\Delta' = \{r'_1, ..., r'_q\}$, $q \geqslant 0$, and

- $p+q$ r-productions in R having the form

$$p_{r_i} = r_i(A^0, Y^1) \to [l]\ \mathbf{Q}_i, \qquad 1 \leqslant i \leqslant p, \qquad \text{and}$$
$$p_{r'j} = r'_j(Y^1, A^0) \to [l]\ \mathbf{Q}'_j, \qquad 1 \leqslant j \leqslant q,$$

add to R' the r-production $[\Delta, \Delta'](A^0, Y^1) \to [l']\ \sigma(\mathbf{M} \cup \{Y^1\}, \mathbf{S})$, where $\mathbf{S}$ is the union of all $\mathbf{Q}_i$ and $\mathbf{Q}'_j$, $1 \leqslant i \leqslant p$ and $1 \leqslant j \leqslant q$;

(3) in a way completely symmetrical to step (2), add r-productions with left-hand side $[\Delta, \Delta'](Y^1, A^0)$ to R' for every possible choice of symbol Y, relation $[\Delta, \Delta']$ and $\text{card}(\Delta) + \text{card}(\Delta')$ r-productions.

We remark that step (3) above is necessary. In fact, even if relation $X^i < Y^j$ always holds for r-items of the form $[\Delta, \Delta'](X^i, Y^j)$ in the productions of G', this might not be the case in a derived SR sentence, due to the renumbering of s-items at the rewriting steps.

From property (i) above, we have that G' is monotonic. Observe that, at steps (2) and (3) in the given construction, r-productions are added to G' to exploit all possible choices of r-productions for any $[\Delta, \Delta'] \in V'_R$. Hence no r-rewriting failure is introduced in G' that was not already present in G. We can conclude that our construction preserves the completeness property. Finally, using property (ii) above, it is not difficult to verify that $L(G') = L(G)$. ▌

In the next section, the existence of a monotonic normal form will be used to simplify the proof of the r-deterministic normal form. In Section 5, the monotonic normal form result allows the definition of derivation tree structures describing the generation of SR sentences by an SR grammar. Moreover, as shown below, the existence of a monotonic normal form provides an immediate proof of the following upper bound on the computational complexity of the recognition problem for languages generated by an SR grammar. In Theorem 4.2, we denote as NSPACE(n) the class of all languages whose membership problem can be solved by a nondeterministic Turing machine using an amount of space equal to the length of the input, where n is the length of the string representing the set of s-items and the set of r-items of an SR sentence.

THEOREM 4.2. *Let L be a language generated by an SR grammar. Then $L \in$* NSPACE(n).

Proof. Let G be an SR grammar generating L. Without loss of generality, we assume that G is monotonic. Any derivation in G is a sequence of SR sentences where the number of s-items is nondecreasing, by Definition 2.4, and the number of r-items is nondecreasing, by Proposition 3.2. As a consequence, a nondeterministic Turing machine can easily be specified that decides whether a string of length n, representing a terminal SR sentence, belongs to L, using an amount of space bounded by n. ▌

Finally, let us remark that, although in the general case the derivation process for an SR grammar is not monotonic in the number of r-items, it is not difficult to establish an upper bound on the size of the generated intermediate SR sentences.

Since s-items cannot be deleted in the rewriting process, the only possible way in which the number of r-items can decrease is through the collapsing of several competing r-items into (at least) one new r-item. Now, it is evident that the cardinality of a set of competing r-items can be at most $2 * \text{card}(V_R)$. On this basis, it is easy to show that the number of r-items of any intermediate SR sentence is at most $2 * \text{card}(V_R)$ times the number of r-items of the terminal sentence. We conclude that the size of an intermediate SR sentence is bounded by the number of s-items in the terminal SR sentence, plus the number of r-items of the terminal sentence times the constant $2 * \text{card}(V_R)$. This upper bound could be used to offer an alternative proof of Theorem 4.2 above.

4.2. r-Deterministic Normal Form

In an SR derivation step the rewriting of an r-item can be nondeterministic, in the sense that several alternative r-productions are applicable once an r-item is selected for rewriting after a given s-production has been applied. In Section 3.2 the definition of the subclass of r-deterministic SR grammars was given, for which no more than one choice is possible in the r-rewriting phase. In the next theorem it is shown that any SR language can be generated by an r-deterministic SR grammar. The basic idea underlying the proof is to relabel the neighbors of an s-item before it is rewritten. Thus, the nondeterminism in r-item rewriting is transformed into nondeterminism in s-item rewriting. If the original grammar is complete, the completeness is preserved by the transformation.

For the proof of the theorem we start from an SR grammar in *terminal r-deterministic form*, where r-items containing terminal symbols have only deterministic rewriting rules. More precisely,

DEFINITION 4.1. An SR grammar $G = (V_N, V_T, V_R, S, P, R)$ is in *terminal r-deterministic form* (TrDF for short) if, for any s-production $p_s = l\colon A^0 \to \langle \mathbf{M}, \mathbf{R} \rangle$ and for any r-item of the form $r(b^1, A^0)$ (resp. $r(A^0, b^1)$) with $b \in V_T$, $r(b^1, A^0)$ (resp. $r(A^0, b^1)$) is the left-hand side of at most one r-production allowed by p_s. ▌

The following lemma states that the assumption used in the proof of the r-deterministic normal form does dot cause any loss of generality, since for any SR grammar there exists an equivalent SR grammar in Terminal r-Deterministic form. An SR grammar can be cast in TrDF normal form by replacing any terminal symbol of the source grammar with a new corresponding nonterminal one. Then suitable unit productions and (r-deterministic) r-productions are introduced to ensure the generation of the original language, as shown in detail below.

LEMMA 4.1. *Let L be a language generated by an SR grammar G. Then L can be generated by an SR grammar G′ which is in TrDF. Moreover, if G is complete then G′ is also complete.*

Proof. Given $G = (V_N, V_T, V_R, S, P, R)$, we construct an SR grammar $G' = (V'_N, V_T, V_R, S, P', R')$ such that G' is in TrDF and $L(G) = L(G')$. The set V'_N is equal to $V_N \cup W$, where W is a set of new nonterminal symbols, defined as $W = \{\bar{a} \mid a \in V_T\}$.

The set of s-productions P' is composed of

— the set $\bar{P}$ of the s-productions which are obtained by substituting each occurrence of a^n in P with $\bar{a}^n$, for each $a \in V_T$ and $n \geqslant 0$,

— an s-production of the form

$$\bar{a}^0 \to \langle \{a^2\}, \varnothing \rangle$$

for any $\bar{a} \in W$.

The set of r-productions R' is composed of

— the set $\bar{R}$ of the r-productions which are obtained by replacing each occurrence of a^n in R with $\bar{a}^n$, for each $a \in V_T$ and $n \geqslant 0$;

— the set $\bar{\bar{R}}$ of the r-productions

$$r(X^1, \bar{a}^0) \to [l]\{r(X^1, a^2)\} \qquad (\text{resp. } r(\bar{a}^0, X^1) \to [l]\{r(a^2, X^1)\})$$

for any s-production in P' of the form $l: \bar{a}^0 \to \langle \{a^2\}, \varnothing \rangle$, and for any r-item of the form $r(X^i, \bar{a}^j)$ (resp. $r(\bar{a}^j, X^i)$), with $X \in V'_N$, which appears in the right-hand side of a production in $\bar{P}$ or in $\bar{R}$;

— the r-productions

$$r(X^1, \bar{a}^0) \to [l]\{r(X^1, a^2)\} \qquad (\text{resp. } r(\bar{a}^0, X^1) \to [l]\{r(a^2, X^1)\})$$

for any s-production in P' of the form $l: \bar{a}^0 \to \langle \{a^2\}, \varnothing \rangle$, and for any r-item of the form $r(X^i, \bar{a}^j)$ (resp. $r(\bar{a}^j, X^i)$), with $X \in V_T$, which appears on the right-hand side of a production in $\bar{\bar{R}}$.

The obtained grammar is in TrDF by construction, and it is easy to verify that the grammar G' is equivalent to G. However, G' is not complete, since no r-productions are provided for r-items of the form $r(A^i, a^j)$, where $A \in V_N$ and $a \in V_T$. To solve the problem, for each r-item such as $r(A^i, a^j)$, and for each s-production $l: (A^0 \to \langle \mathbf{M}, \mathbf{R} \rangle)$ belonging to P we arbitrarily extract from G one r-production allowed by l with left-hand side $r(A^0, a^1)$ and add it to R' (if some such r-production exists in G). The generated language is not affected and the above mentioned source of r-rewriting failure is eliminated, unless it was already present in G. ▮

Let us now provide the intuition for the construction which allows us to remove from a source grammar the nondeterminism in r-item rewriting. Nondeterministic r-item rewriting means that several alternative r-productions may be allowed for an r-item of the form $r(X^j, A^i)$ by an s-production that rewrites A^i or X^j. As a consequence, if two different occurrences X^h and X^k of the symbol X are related to A^i through the same relation symbol r, different choices may be made in order to rewrite the r-items $r(X^h, A^i)$ and $r(X^k, A^i)$. By definition, this is impossible if the grammar is r-deterministic. In order to solve the problem, for any symbol of the original grammar we introduce a set of new nonterminal symbols. The idea is to rewrite X^h and X^k as different elements of the associated set of new symbols by

suitable unit s-productions before A^i is actually rewritten. Thus, alternative choices of r-item rewriting can be done when A^i is rewritten, even if the grammar is r-deterministic. Then, the new symbols produced from X^h and X^k can be replaced by occurrences of the symbol X by other suitable unit s-productions.

We can now present the construction in detail.

THEOREM 4.3. *Let L be a language generated by an SR grammar G. Then L can be generated by an r-deterministic SR grammar G'. Moreover, if G is complete then G' is also complete.*

Proof. Without loss of generality, we assume that $G=(V_N, V_T, V_R, S, P, R)$ is in TrDF and monotonic normal forms. It is easy to see that starting from a TrDF SR grammar the construction given in Theorem 4.1 preserves the TrDF normal form.

The construction of the r-deterministic SR grammar $G'=(V'_N, V_T, V_R, S, P', R')$ is specified as follows.

For each s-production $p_s = l\colon (A^0 \to \langle \mathbf{M}, \mathbf{R} \rangle) \in P$, each $r \in V_R$, and each $B \in V_N$, we define the sets of r-productions

$$\Delta(p_s, r, B) = \{p_r \mid p_r = (r(A^0, B^1) \to [l]\ \mathbf{Q}) \in R\}, \qquad \text{and}$$

$$\Delta(p_s, B, r) = \{p_r \mid p_r = (r(B^1, A^0) \to [l]\ \mathbf{Q}) \in R\}.$$

Let $d = \max\{\text{card}(\Delta(p_s, r, B)),\ \text{card}(\Delta(p_s, B, r)) \mid p_s \in P,\ r \in V_R,\ B \in V_N\}$. We define

$$V'_N = V_N \cup \{[B, i] \mid B \in V_N, 1 \leqslant i \leqslant d\}.$$

Sets P' and R' are constructed in three steps. Initially, let $P' = P$ and $R' = \{r(\mathsf{X}, \mathsf{Y}) \to [l]\ Q \in R \mid \mathsf{X} \text{ or } \mathsf{Y} \in C_{V_T}\}$.

(1) For each $B \in V_N$ and each i, $1 \leqslant i \leqslant d$, add to P' the unit s-productions $p_{B,i}$ and $p'_{B,i}$, and add to R' the sets of r-productions $R_{B,i}$ and $R'_{B,i}$, specified as

$$p_{B,i} = l_{B,i}\colon B^0 \to \langle \{[B, i]^2\}, \varnothing \rangle;$$

$$p'_{B,i} = l'_{B,i}\colon [B, i]^0 \to \langle \{B^2\}, \varnothing \rangle;$$

$$R_{B,i} = \{v(B^0, Z^1) \to [l_{B,i}]\{v([B, i]^2, Z^1)\} \mid v \in V_R, Z \in (V'_N \cup V_T)\}$$
$$\cup \{v(Z^1, B^0) \to [l_{B,i}]\{v(Z^1, [B, i]^2)\} \mid v \in V_R, Z \in (V'_N \cup V_T)\};$$

$$R'_{B,i} = \{v([B, i]^0, Z^1) \to [l'_{B,i}]\{v(B^2, Z^1)\} \mid v \in V_R, Z \in (V'_N \cup V_T)\}$$
$$\cup \{v(Z^1, [B, i]^0) \to [l'_{B,i}]\{v(Z^1, B^2)\} \mid v \in V_R, Z \in (V'_N \cup V_T)\}.$$

(2) For each $p_s = l\colon (A^0 \to \langle \mathbf{M}, \mathbf{R} \rangle) \in P$, $r \in V_R$, and $B \in V_N$ such that $\Delta(p_s, r, B) \neq \varnothing$, perform the following actions. Assume an arbitrary ordering on $\Delta(p_s, r, B)$ and denote as $r(A^0, B^1) \to [l]\ \mathbf{Q}_i$ the ith element in such an ordering.

In what follows, we write $\mathbf{Q}_i[B^1/[B, i]^1]$ to denote the set of r-items obtained from $\mathbf{Q}_i$ by replacing B^1 with $[B, i]^1$. Add to R' the set of r-productions $\Delta'(p_s, r, B)$ specified as

$$\begin{aligned}\Delta'(p_s, r, B) = &\{r(A^0, [B, i]^1) \to [l]\ \mathbf{Q}_i[B^1/[B, i]^1] \mid 1 \leqslant i \leqslant \mathrm{card}(\Delta(p_s, r, B))\} \\ &\cup \{r(A^0, [B, i]^1) \to [l]\ \mathbf{Q}_1[B^1/[B, i]^1] \mid \mathrm{card}(\Delta(p_s, r, B)) < i \leqslant d\} \\ &\cup \{r(A^0, B^1) \to [l]\ \mathbf{Q}_1\}.\end{aligned}$$

(3) For each $p_s \in P$, $r \in V_R$ and $B \in V_N$ such that $\Delta(p_s, B, r) \neq \varnothing$, define sets of r-productions $\Delta'(p_s, B, r)$ starting from sets $\Delta(p_s, B, r)$ and in a way completely symmetrical to step (2). Add these sets to R'.

G' is r-deterministic, since sets $R_{B,i}$, $R'_{B,i}$, $\Delta'(p_s, r, B)$, and $\Delta'(p_s, B, r)$ do not introduce nondeterminism in r-item rewriting. Neither do they introduce r-rewriting failure. Therefore G' is complete whenever G is complete. Note that the second and the third sets in the definition of $\Delta'(p_s, r, B)$ above are needed just to maintain completeness. Finally, it is not difficult to see that the construction preserves the language L, since the unit s-productions introduced at step (1) only add "ineffective" extra steps to the derivations in G'. It is worth noting that the different alternative ways of r-item rewriting are completely reproduced through the set of new symbols V'_N only if the original grammar is monotonic. In the nonmonotonic case, a larger set of new symbols should be introduced, where a new symbol is present for each possible combination of competing r-items. ▌

The proof of the above theorem shows that the price to pay for the elimination of nondeterminism in r-items rewriting is the introduction of unit productions. It is interesting to note that this fact is in agreement with a result of Welzl (1984) about the expressiveness of NLC-like graph grammar formalisms. These formalisms do not provide facilities corresponding to nondeterministic r-item rewriting. As a matter of fact Welzl showed that the expression of a grammar generating all graphs labelled by an arbitrary and fixed set of labels is impossible in NLC formalism without unit productions. Let us observe that in an SR sentence each symbol occurrence can be seen as a labelled node and each r-item as a directed edge. With this interpretation, the grammar of Example 3.2 generates all the directed graphs with nodes labelled by a and edges labelled by r. Thus, the SR formalism makes it possible to express the language of "all graphs" in a quite immediate way without unit productions through nondeterministic r-item rewriting.

Table 4.1 exemplifies how we can renounce nondeterministic r-item rewriting if unit s-productions are used. The example refers to a variant of the grammar in

TABLE 4.1.a

P_a:	R_a:
1: $S^0 \to \langle \{a^2, S^2\}, \varnothing \rangle$	$r(a^1, S^0) \to [1, 2, 3]\{r(a^1, a^2), r(a^2, a^1)\}$
2: $S^0 \to \langle \{a^2, S^2\}, \{r(a^2, S^2)\rangle\}$	$r(a^1, S^0) \to [1,2]\{r(a^1, a^2), r(a^2, a^1), r(a^1, S^2)\}$
3: $S^0 \to \langle \{a^2\}, \varnothing \rangle$	$r(a^1, S^0) \to [1, 2]\{r(a^1, S^2)\}$

TABLE 4.1.b

P_c:	R_c:
1: $S^0 \to \langle \{\bar{a}^2, S^2\}, \varnothing \rangle$	$r([a,1]^1, S^0) \to [1,2,3]\{r([a,1]^1, \bar{a}^2)\}$
2: $S^0 \to \langle \{\bar{a}^2, S^2\}, \{r(\bar{a}^2, S^2)\} \rangle$	$r([a,2]^2, S^0) \to [1,2]\{r([a,2]^1, \bar{a}^2), r([a,2]^1, S^2)\}$
3: $S^0 \to \langle \{\bar{a}^2\}, \varnothing \rangle$	$r([a,3]^1, S^0) \to [1,2]\{r([a,3]^1, S^2)$
	$r([a,h]^1, S^0) \to [3]\{r([a,h]^1, \bar{a}^2)\}$
	$r(\rho^1, S^0) \to [1,2,3]\{r(\rho^1, \bar{a}^2)\}$
l_k: $\bar{a}^0 \to \langle \{[a,k]^2\}, \varnothing \rangle$	$r(\bar{a}^0, \sigma^1) \to [l_k]\{r([a,k]^2, \sigma^1)\}$
l'_k: $[a,k]^0 \to \langle \{\bar{a}^2\}, \varnothing \rangle$	$r([a,k]^0, \sigma^1) \to [l'_k]\{r(\bar{a}^2, \sigma^1)\}$
	$r(\tau^1, \bar{a}^0) \to [l_k]\{r(\tau^1, [a,k]^2)\}$
	$r(\tau^1, [a,k]^0) \to [l'_k]\{r(\tau^1, \bar{a}^2)\}$
4: $\bar{a}^0 \to \langle \{a^2\}, \varnothing \rangle$	$r(\phi^1, \bar{a}^0) \to [4]\{r(\phi^1, a^2)\}$
	$r(\bar{a}^0, \gamma^1) \to [4]\{r(a^2, \gamma^1)\}$
	$r(\bar{a}^0, a^1) \to [4]\{r(a^2, a^1), r(a^1, a^2)\}$
	$r(a^1, \bar{a}^0) \to [4]\{r(a^1, a^2), r(a^2, a^1)\}$
	where: $h = 2, 3$ $k = 1, 2, 3$ $\rho = \bar{a}, a$
	$\sigma = \bar{a}, S, a, [a,1], [a,2], [a,3]$ $\gamma = \bar{a}, S, [a,1], [a,2], [a,3]$
	$\tau = \bar{a}, a, [a,1], [a,2], [a,3]$ $\phi = \bar{a}, [a,1], [a,2], [a,3]$

Example 3.2, which generates all possible directed graphs where the orientation of edges is not effective, in the sense that an edge can appear only if the reversed edge is also present. Tables 4.1a and 4.1b show the "natural" version of the grammar and the r-deterministic grammar, which is obtained by applying the construction given in the proof of Theorem 4.3, respectively. Note that the resulting r-deterministic version of the grammar presents a notable loss of compactness and readability with respect to the original one.

5. SEMANTIC ANALYSIS OF VISUAL LANGUAGES

The idea underlying the derivation of SR grammars is to handle s-items and r-items in a quite uniform fashion, and rewrite any item by means of context-free styled rules (s-productions or r-productions, accordingly). This turned out to be particularly useful for extending well-established techniques developed for string grammars. In particular, in this section we show that the notion of attribute context-free grammar can be extended naturally to SR grammars in order to perform semantic analysis of visual languages. The technique benefits from the existence of a derivation structure that is analogous to the usual syntactic tree of context-free string languages.

5.1. Derivation Trees

The concept of derivation structure for SR grammars is introduced to describe the SR sentence generation process. The formalism of SR grammars uses a mechanism that rewrites both r-items and s-items uniformly through context-free

styled rules. This viewpoint suggests that derivations of SR sentences can be described by means of a tree-shaped derivation structure, called the SR tree (Symbol–Relation tree). In an SR tree each node corresponds either to an s-item or to an r-item. The children of a nonterminal node are the occurrences of the items obtained by applying an s-production or an r-production, respectively. The idea of an SR tree and its construction during an SR derivation for the grammar of Example 3.1 are illustrated below.

EXAMPLE 5.1. Figure 5.1 shows the construction of the SR tree associated with the derivation of the sentence $\langle \{a^2, a^3\}, \{r(a^3, a^2)\} \rangle$ in agreement with the grammar of Example 3.1.

The root of the SR tree of Fig. 5.1a is an occurrence of the initial symbol H. The root has as many children as the number of items appearing in the right-hand side of the production which is applied to H in the first derivation step. Each child is either an s-item or an r-item. The effects of the second derivation step are shown in Fig. 5.1b, where s-production 2 has been applied to the nonterminal s-item H^3 and $sl(H^2, H^3)$ has been rewritten by the r-production $sl(H^1, H^0) \rightarrow [2]\{nsl(H^1, a^2)\}$. A terminal sentence is finally obtained by applying s-production 2 to H^2 and then the r-production $nsl(H^0, a^1) \rightarrow [2]\{r(a^2, a^1)\}$ to $nsl(H^2, a^2)$. The yield of the SR tree in Fig. 5.1c is the terminal SR-sentence $\langle \{a^2, a^3\}, \{r(a^3, a^2)\} \rangle$. In general, an SR tree corresponds to a terminal sentence if all the nodes of its frontier are occurrences of terminal items.

It should be clear that each derivation of an SR grammar can be represented as a tree structure whenever there is no collapsing of r-items (see Section 3). The collapsing of two r-items could be represented by a node having two different parents, but in that case a graph structure would be necessary to represent the

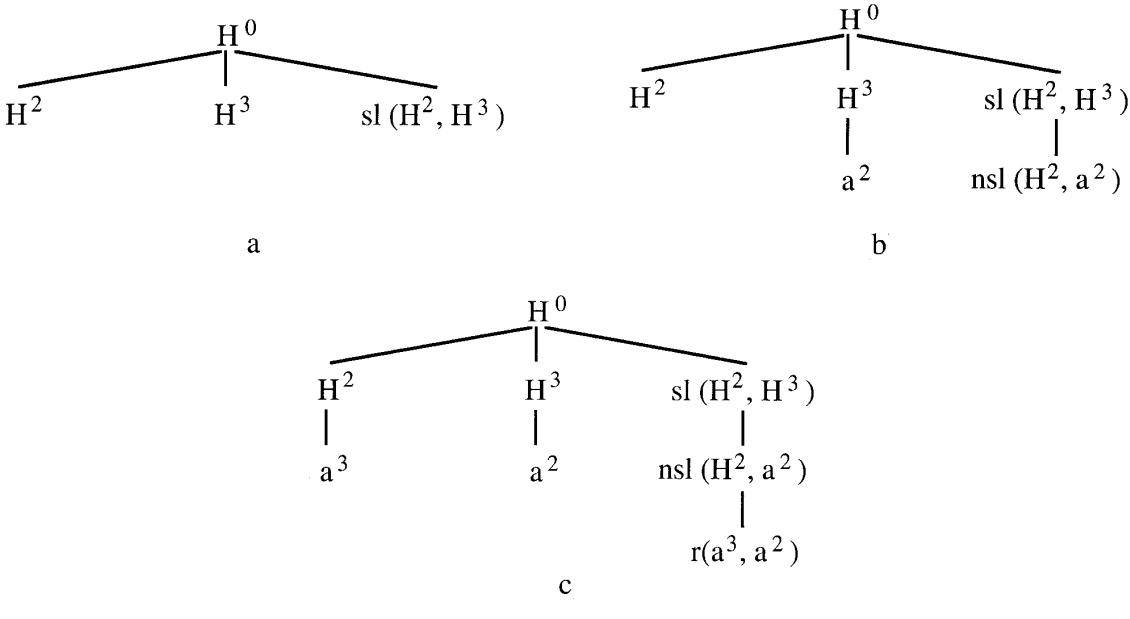

FIG. 5.1. Construction of the SR tree of the sentence $\langle \{a^2, a^3\}, \{r(a^3 a^2)\} \rangle$.

derivation. On the other hand, Theorem 4.4 shows that each SR language can be generated by a monotonic SR grammar, whose derivations do not present any collapsing of r-items. In this sense, derivation structures are assumed to be defined as trees without loss of generality. As shown in the above example (see Fig. 5.1), SR trees are also able to describe context-sensitive aspects of derivations.

The use of derivation structures to determine properties of grammars is very common for context-free grammars. Also in the domain of graph grammars, suitable derivation structures have been exploited by Yamazaki and Yaku (1993) to demonstrate a pumping lemma for Boundary NLC grammars and by Engelfriet (1990) to provide a characterization of confluent edNCE graph grammars in terms of regular tree languages and regular string languages. An SR tree can be useful to study the properties of SR derivations and to extend techniques and results typical of context-free grammars. A meaningful example of application of SR trees is given in the next section, concerning the semantic analysis of visual languages specified by SR grammars.

To get a grasp of how SR trees can be exploited to prove properties of SR grammars, consider the class of complete SR grammars, defined in Section 3.1. It can be shown that the decidability of the emptiness problem for the languages generated by C-SR grammars can be obtained by a straightforward extension of the proof for context-free languages in Hopcroft and Ullman (1979). This proof is based on the fact that if the language generated by a context-free grammar contains at least one word, then it must contain at least one word whose derivation tree has depth not greater than the number of nonterminals of the grammar. To decide whether the language is empty it is sufficient to exhaustively generate all such derivation trees. The same argument can be used to solve the emptiness problem for C-SR grammars. Indeed, in a C-SR grammar the application of s-productions cannot be affected by any r-rewriting failure.

5.2. Attribute SR Grammars

Let us now show that the concept of an SR tree can be useful for extending well-established techniques of semantic analysis developed for string grammars. In particular, let us consider the well-known concept of Attribute Grammars, which are a widely used tool for the description and implementation of the semantic analysis in programming language compilers, initially introduced by Knuth (1968). An attribute grammar is a context-free grammar with synthesized and inherited attributes associated with symbols, whose values are computed by semantic rules associated with grammar productions. An attribute grammar performs a translation by assigning a meaning to each parse tree through the evaluation of the designated attribute of its root.

An Attribute SR grammar (ASR grammar) can be defined as an SR grammar $G=(V_N, V_T, V_R, S, P, R)$ such that synthesized and inherited attributes are associated with nonterminal symbols in V_N and relation symbols in V_R (Ferrucci *et al.* 1994a). The values of the attributes are computed by means of semantic rules associated with each s-production and r-production. Similar approaches exist in the literature to defining attribute graph grammars (Bunke, 1982; Gottler, 1982;

Kaplan *et al.*, 1987; Gottler, 1989) and attribute tree grammars (Barbar, 1993). SR trees can be used as a very natural base to accomplish attribute evaluation. In essence, the attribute evaluation for a given derivation can be performed as a (single or multiple) visit of the corresponding SR tree. The translation performed by an ASR grammar G, denoted by $T(G)$, is the mapping from SR trees of G to the set of values of a synthesized attribute of the initial symbol of G, such that if t is an SR tree, then $T(G)(t)$ is the value of the designated attribute of the root of t (after the attribute evaluation).

In the following we illustrate the use of ASR grammars for the semantic analysis, by means of two examples. In particular, the first example shows the capability of ASR grammars to synthesize structural information of the sentences. The second example is meant to show how the evaluation of the attributes carried out by a semantic analysis can be used to translate a flowchart of a program into the corresponding programming language code.

EXAMPLE 5.2. Let us consider the grammar *All-s* of Example 3.2 that generates all the graphs. As we will show below, by associating suitable attributes to the items of the grammar *All-s* we can define an ASR grammar which is able to collect information about the structure of the graph. In particular, the value of one of the synthesized attributes of the starting s-item is the number of connected components of the graph.

In the following, we list the synthesized attributes associated with the terminal s-item a^h and the nonterminal items of type S^k, $r(a^h, S^k)$ and $r(S^k, a^h)$. The corresponding semantic rules are devised in order to obtain a bottom-up flowing of all information necessary to construct the connected components, as the value of a synthesized attribute of the initial symbol S.

(a) An s-item a^h has the attribute T, where:

- $a^h.T$ is the node a^h

(b) An s-item S^k has the attributes CC and N, where:

- $S^k.CC$ is a set of elements, where each element describes a connected component of the (sub)graph generated by S^k. Any component is described by its set of nodes, without further information.
- $S^k.N$ is the number of connected components of the (sub)graph generated by S^k. It is the designated attribute of the start symbol.

(c) An r-item $r(a^h, S^k)(r(S^k, a^h))$ has the attribute D, where:

- $r(a^h, S^k).D$ is the set of nodes which the edges generated by $r(a^h, S^k)$ (resp. $r(S^k, a^h)$) are incident with.

For any terminal r-item $r(a^h, b^k)$, $r(a^h, b^k).D$ is the set $\{a^h.T, b^k.T\}$.

The semantic rules associated with s-productions and r-productions are defined in agreement with the above intended meaning for synthesized attributes. The semantic rules are given in detail in Tables 5.1a and 5.1b below.

EXAMPLE 5.3. In Section 2.2 we have introduced the grammar Fl_Ch generating structured flowcharts with any number of nested *if–then–else* and *while* structures

TABLE 5.1.a.

Semantic Rules for s-Productions

(1) $S^0.CC := S^2.CC \cup \{\{a^2.T\}\}$
$S^0.N := \text{card}(S^0.CC)$

(2) $S^0.N := \text{card}(S^0.CC)$
$S^0.CC := (S^2.CC - F) \cup \{F1 \cup \{a^2.T\}\}$;
where
$F := \{c \in S^2.CC \mid c \cap r(a^2, S^2).D \neq \varnothing\}$
$F1 := \bigcup c$ such that $c \in F$

(3) $S^0.N := \text{card}(S^0.CC)$
$S^0.CC := (S^2.CC - F) \cup \{F1 \cup \{a^2.T\}\}$;
where
$F := \{c \in S^2.CC \mid c \cap (r(S^2, a^2).D \cup r(a^2, S^2).D) \neq \varnothing\}$
$F1 := \bigcup c$ such that $c \in F$

(4) $S^0.N := \text{card}(S^0.CC)$
$S^0.CC := (S^2.CC - F) \cup \{F1 \cup \{a^2.T\}\}$;
where
$F := \{c \in S^2.CC \mid c \cap r(a^2, S^2).D \neq \varnothing\}$
$F1 := \bigcup c$ such that $c \in F$

(5) $S^0.N := \text{card}(S^0.CC)$
$S^0.CC := \{\{a^2.T\}\}$;

and we have suggested their use in a visual programming environment. In particular, we have pointed out that the evaluation of the attributes carried out by a semantic analysis can be used to translate a typical flowchart of a program into the corresponding code in a programming language, such as Pascal, C, etc. To do that we define an ASR grammar with Fl_Ch as underlying SR grammar. In the following, we list the synthesized attributes associated with the terminal s-items and the nonterminal s-items. The value of a synthesized attribute of the initial symbol S is the corresponding program code.

The nonterminal s-items F^i and S^i are assigned the attribute *code*, whose value is the code associated to the portions of flowchart that they generate. The same attribute *code* is assigned to the terminal s-items $cond^i$ and $simple^i$, where $cond^i.code$ and $simple^i.code$ are respectively the textual description of the conditions and the textual description of the statements, which are explicitly associated with blocks by the user of the visual environment. Moreover, a boolean attribute *seqn* is associated with any s-item F^i. If F^i generates a compound statement, then F^i.seqn is True, otherwise it is False.

TABLE 5.1.b.

Semantic Rules for r-Productions

$r(a^1, S^0).D := r(a^1, a^2).D$
$r(a^1, S^0).D := r(a^1, S^2).D \cup r(a^1, a^2).D$
$r(a^1, S^0).D := r(a^1, S^2).D$
$r(S^0, a^1).D := r(a^2, a^1).D$
$r(S^0, a^1).D := r(S^2, a^1).D \cup r(a^2, a^1).D$
$r(S^0, a^1).D := r(S^2, a^1).D$

The semantic rules associated with s-productions are defined in agreement with the above intended meaning for synthesized attributes. The semantic rules are given in details in Table 5.2, where the symbol ‖ denotes the string concatenation function.

A relevant property of the ASR grammars of the previous examples is that the translation of their SR trees can be always performed in a single bottom-up visit. In general, the evaluation of the attributes can be very complex and expensive, due to circularity problems typical of attribute grammars. The order of attribute evaluation follows the dependencies induced by the semantic functions and by the form of the SR tree. Since we are especially interested in efficient techniques for semantic analysis of visual languages, it is important to consider special cases in which the evaluation of attributes can be performed efficiently. As usual, an attribute SR grammar is called "*one-visit*" if the attributes of every SR tree can be evaluated by walking through the SR tree visiting any subtree exactly once.

The one-visit property for string grammars has a static characterization that can be easily verified by an analysis of the semantic rules (Engelfriet and Filè, 1981). Ferrucci *et al.* (1994a) have shown how this characterization can be extended to ASR grammars. An attribute a_j is said to *depend on* on attribute a_i in p iff there exists a semantic rule associated with p that defines the value of a_j as a function of the value of a_i. The one-visit property for ASR grammars can be statically verified by considering how the symbols of the right-hand side of each s-production (or r-production) p depend on each other through their attributes. An ASR grammar is said to be *Statically One-Visit* if all its productions do not present circular dependencies of attributes. As shown in Ferrucci *et al.* (1994a), an ASR G is one-visit if and only if it is statically one-visit.

TABLE 5.2.

Semantic Rules for s-Productions

(1) S^0.code := "program" ‖ ";" ‖ F^2.code ‖ "end."

(2) F^0.code := F^2.code ‖ ";" ‖ F^3.code
F^0.seqn := True

(3) F^0.code := simple^2.code
F^0.seqn := False

(4) F^0.code := "if" ‖ cond^2.code ‖ "then" ‖ body$_1$ ‖ "else" ‖ body$_2$
F^0.seqn := False

(5) F^0.code := "while" ‖ cond^2.code ‖ "do" ‖ body$_1$
F^0.seqn := False

where the values of body$_1$ and body$_2$ are determined as follows:

if (F^0.seqn = True)
then body$_i$ = "begin" ‖ F^{i+1}.code ‖ "end"
else body$_i$ = F^{i+1}.code
for $i = 1, 2$

6. SR VERSUS GRAPH GRAMMARS

In this section SR grammars are compared with well known graph rewriting formalisms. In particular, we consider the family of node rewriting graph grammars (Janssens and Rozenberg, 1980a), whose derivation process is also characterized by the distinction of two phases. A node rewriting phase, which generates a *daughter graph*, is followed by a second phase, the embedding phase, which embeds the daughter graph into the *host graph*, as described by Kreowski and Rozenberg (1990).

In the following, the analogies and the differences between SR grammars and node rewriting graph grammars are pointed out and some equivalences between classes of grammars are established. The equivalence results clarify the influence that certain features have on the generative power. Moreover, they are useful to prove that the generative power of some subclasses of SR grammars is increased by the presence of nondeterministic r-item rewriting.

6.1. edNCE Graph Grammars

The family of NLC-like grammars has been widely investigated (see, e.g., Janssens and Rozenberg, 1980a, 1980b; Ehrenfeucht and Rozenberg, 1984). These grammars are characterized by a *node label controlled* (NLC) rewriting mechanism. The NLC derivation process is based on transformations, which are expressed by a finite number of rules, each one consisting of a production and of a finite number of embedding rules. NLC productions are context-free style node rewriting rules which replace a mother node with a daughter graph (Janssens and Rozenberg, 1983). The embedding rules are specified as a set of pairs of the form (λ, μ), where λ and μ are (terminal or nonterminal) node labels. If a pair (λ, μ) belongs to an embedding rule, an edge should be established between each node labeled λ in the daughter graph and each node labelled μ in the neighborhood of the mother node. Thus, embedding rules are described using a formalism quite different from the one employed in SR grammars, where r-items rewriting rules are used instead. Nonetheless, several results can be established relating the generative power of SR grammars to the one of NLC-like grammars. In particular, we consider edNCE grammars (edge-labelled directed Neighborhood Controlled Embedding grammars; Engelfriet *et al.*, 1990; Engelfriet, 1990), which generate graphs with labelled nodes and labelled directed edges.

According to the derivation process of edNCE grammars, a production can always be applied to a node independently of the context; that is, the embedding phase is always successful. There are two main consequences of this fact. First, a mechanism directly analogous to the r-rewriting failure feature of SR grammars is not available in edNCE grammars. Second, if an edge connecting a rewritten node and a node in its neighborhood is not considered by the embedding rules, this edge is just deleted from the resulting graph. We recall that, on the other hand, r-item deletion is not possible in SR grammars, since r-items for which rewriting rules are not available cause r-rewriting failure and the right-hand sides of r-productions are nonempty. Nevertheless, edNCE grammars have a particular capability, called

blocking edges, which has an effect similar to the r-rewriting failure of SR grammars, as will be discussed below. More precisely, edNCE grammars introduce the distinction between *final* and *nonfinal* edges, where only graphs with final edges belong to the languages generated by the grammar. A blocking edge is then defined as a nonfinal edge connecting two terminal nodes. Since terminal nodes cannot be further rewritten by the grammar and a graph containing a blocking edge is not in the language generated by the grammar, blocking edges act as filters for the generated language. Finally, we must note that edNCE grammars, and all NLC-like grammars as well, do not admit alternatives in the way a daughter graph can be embedded is a host graph. Following the SR terminology, we may say that the NLC-like grammars are r-deterministic. In essence, edNCE grammars exploit both edge deletion and a mechanism that can block the derivation process, that is, the blocking edge, but not nondeterministic embedding, whereas SR grammars exploit both nondeterminism in r-item rewriting and a mechanism that can block the derivation process, that is the r-rewriting failure, but not r-item deletion.

In what follows we formally investigate the consequences of the above-mentioned differences, under both a generative and a computational perspective. In order to do so, we need to introduce some additional definitions. First, we have to make the assumption that SR sentences and graphs with labelled nodes and labelled directed edges can be compared. This is done in the obvious way, by implicitly reinterpreting s-items as labelled nodes and r-items as labelled directed edges. Second, some extensions of the SR formalism must be defined. We introduce the class of SRd grammars, in which r-item deletion is permitted by allowing r-productions of the form $s(\mathsf{A}, \mathsf{B}) \rightarrow [l]\varnothing$. If during a derivation step an r-production of this form is applied to rewrite an r-item $s(A^i, B^j)$, the r-item is deleted from the resulting SR sentence. Furthermore, we introduce a subclass of SRd grammars which is characterized by the lack or r-rewriting failure capability, the Complete SRd grammars.

DEFINITION 6.1. The class of *SRd* (resp. *C-SRd*) grammars is obtained from the class of SR (resp. C-SR) grammars by allowing r-productions of the form

$$s(\mathsf{A}, \mathsf{B}) \rightarrow [l]\varnothing$$

where $\varnothing$ is the empty set.

Finally, in this section we use the symbol Δ to denote the r-deterministic variants of all the above classes. For example, ΔC-SRd denotes the class of r-deterministic complete SR grammars with r-item deletion.

The class of *N-edNCE* graph grammars (Engelfriet *et al.*, 1990; Engelfriet, 1990) is defined as the subclass of all edNCE grammars which cannot generate blocking edges, i.e., when an edge is established between two terminal nodes, it must have a final label. In this way, in N-edNCe graph grammars the distinction between final and nonfinal edge labels is uneffective. Thus, given an N-edNCE grammar we can assume that it has only final edge labels.

Our first result concerning the comparison between SRd grammars and graph grammars states that the class of languages generated by C-SRd grammars (denoted by $\mathscr{L}$(C-SRd)) is equal to the one generated by N-edNCE graph grammars

(denoted by $\mathscr{L}$(N-edNCE)). Thus, the elimination of blocking edges has an effect analogous to the one of completion of SRd grammars.

THEOREM 6.1. *The class of C-SRd grammars is equivalent to the class of N-edNCE graph grammars. More precisely, the following equivalences hold*:

$$\mathscr{L}(\text{C-SRd}) = \mathscr{L}(\Delta\text{C-SRd}) = \mathscr{L}(\text{N-edNCE}).$$

Proof. We observe that the r-deterministic normal form result also holds for SRd grammars. Indeed, the construction used in the proof of Theorem 4.3 can be applied in order to prove that $\mathscr{L}(\Delta\text{-SRd}) = \mathscr{L}(\text{SRd})$. Moreover, it can be verified that the construction of the proof preserves the completeness of the original grammar, thus we have $\mathscr{L}(\Delta\text{C-SRd}) = \mathscr{L}(\text{C-SRd})$. The proof of the second equivalence above requires the formal definition of edNCE and N-edNCE grammars and has been reported in Appendix B. ▮

Let us recall that the only difference between edNCE and N-edNCE graph grammars lies in the fact that the former has the blocking edge capability. It should then be intuitive that one can reformulate any edNCE grammar G as a pair $\langle G_0, V_{\text{F}}\rangle$, where G_0 is an N-edNCE grammar and V_{F} is a finite alphabet which plays the role of the set of final edge labels in edNCE grammars. The language degenerated by G, denoted by $L(G)$, consists of all the graphs in $L(G_0)$ whose edge labels are in V_{F}. With this reformulation, it is easy to prove the following theorem, which states the equivalence of the classes edNCE and SRd.

THEOREM 6.2. *The following equivalences hold*:

$$\mathscr{L}(\text{SRd}) = \mathscr{L}(\Delta\text{SRd}) = \mathscr{L}(\text{edNCE}).$$

Proof. The first equivalence can be shown by applying the construction used in the proof of Theorem 4.2, as stated in the proof of Theorem 6.1. In order to prove the second equivalence, consider an edNCE grammar reformulated as a pair $G = \langle G_0, V_F\rangle$, where G_0 is an N-edNCE grammar and V_F is some alphabet. From Theorem 6.1 we have that there exists a ΔC-SRd grammar $G_1 = (V_N, V_T, V_R, S, P, R)$ such that $L(G_1) = L(G_0)$. Consider then the ΔSRd grammar $G' = (V_N, V_T, V_R, S, P, R')$, where R' includes all and only those r-productions in R whose right-hand side does not contain a terminal r-item with relation symbol not in V_F. It is easy to see that $L(G) = L(G')$, since now an r-rewriting failure occurs in a derivation of G' if and only if a blocking edge is generated in the corresponding derivation of G.

To prove the converse, let $G = (V_N, V_T, V_R, S, P, R)$ be a ΔSRd grammar, and let \$ be a new relation symbol. For each $X \in V_N \cup V_T$, each $r \in V_R \cup \{\$\}$, and each s-production $l: A^0 \to \langle \mathbf{M}, \mathbf{R}\rangle$ in P such that no r-production of the form $r(X^1, A^0) \to [l]\ \mathbf{Q}$ exists in R, choose some symbol occurrence in $\mathbf{M}$, say Y^j, and add to an initially empty set R' the r-production $r(X^1, A^0) \to [l]\{\$(X^1, Y^j)\}$. A symmetrical step is then performed for each X, r and s-production in P specified as above and for each r-production of the form $r(A^0, X^1) \to [l]\ \mathbf{Q}$, adding to an initially empty set R'' r-productions of the form $r(A^0, X^1) \to [l]\{\$(Y^j, X^1)\}$. In

essence, any r-item for which no r-production is allowed in G is rewritten with an r-item of the form $\$(\mathsf{X}, \mathsf{Y})$, and any such r-item produces in turn another r-item of the same type until a terminal r-item is obtained. It is easy to see that the resulting grammar $G^* = (V_N, V_T, V_R \cup \{\$\}, S, P, R \cup R' \cup R'')$ is a ΔC-SRd grammar. Furthermore, we have that $L(G)$ includes all and only the SR sentences in $L(G^*)$ that do not contain r-items with the relation symbol \$. By Theorem 6.1, there exists an N-edNCE grammar G_0 such that $L(G_0) = L(G^*)$, and by the above observation on $L(G)$ we have that the reformulated edNCE grammar $G' = \langle G_0, V_R \rangle$ is such that $L(G') = L(G)$. ▌

EXAMPLE 6.1. Let us consider the language of star-shaped graphs with a b-node in the middle and with 2^n a-nodes attached to this b-node through the relation symbol $*$, for some $n \geqslant 0$. An edNCE graph grammar generating such a language is described by Engelfriet *et al.* (1990). Here we provide an equivalent ΔSR grammar. In essence, the grammar is obtained from the one by Engelfriet *et al.* (1990) by ignoring all the embedding rules which give rise to blocking edges.

Let $G = (\Sigma, \Delta, \Gamma, S, P, R)$ be an SRd grammar, where

$$\Sigma = \{S, A, C\}$$
$$\Delta = \{a, b\}$$
$$\Gamma = \{\lambda, \mu, *\},$$

P contains the s-productions

$$1:\ S^0 \to \langle \{C^2, A^3\}, \{\mu(C^2, A^3)\} \rangle$$
$$2:\ C^0 \to \langle \{C^2\}, \varnothing \rangle$$
$$3:\ C^0 \to \langle \{b^2\}, \varnothing \rangle$$
$$4:\ A^0 \to \langle \{A^2, A^3\}, \varnothing \rangle$$
$$5:\ A^0 \to \langle \{a^2\}, \varnothing \rangle,$$

and R contains the r-productions

$$\mu(C^0, A^1) \to [2]\{\lambda(C^2, A^1)\}$$
$$\mu(C^0, A^1) \to [3]\{\lambda(b^2, A^1)\}$$
$$\lambda(C^1, A^0) \to [4]\{\mu(C^1, A^2), \mu(C^1, A^3)\}$$
$$\lambda(b^1, A^0) \to [5]\{*(b^1, a^2)\}.$$

Observe that, as an effect of the r-rewriting failure, s-productions 1 and 2 cannot be applied when the edges λ are incident with the node C, since no r-production is provided for $\lambda(\mathsf{C}, \mathsf{A})$. Similarly, s-production 3 can never be applied when the edges μ connect the node C or the edges λ are incident with the node b. Finally, s-production 4 is the only rule that can be applied after the node C has been replaced by a node b.

The language of the previous example has been used in Engelfriet *et al.* (1990) to prove that the blocking edge capability affects the generative power of edNCE grammars. Indeed, the class of languages generated by N-edNCE grammars is strictly included in the class of edNCE languages. It is worth noting that the edge

deletion capability is not exploited in the generation of the language of Example 6.1. Thus, as a consequence of Theorems 6.1 and 6.2, the following proposition holds.

PROPOSITION 6.1. (i) *The class of C-SR languages is strictly included in the class of SR languages*; *i.e.*, $\mathscr{L}(C\text{-}SR) \subset \mathscr{L}(SR)$.

(ii) *The class of C-SRd languages is strictly included in the class of SRd languages*; *i.e.*, $\mathscr{L}(C\text{-}SRd) \subset \mathscr{L}(SRd)$.

The following theorem shows that the r-item deletion feature increases the generative power of the SR grammars.

THEOREM 6.3. $\mathscr{L}(SR) \subset \mathscr{L}(SRd)$.

Proof. Consider the language of discrete SR sentences, i.e., SR sentences where the set of r-items is empty, having 2^n s-items of the kind a^j, with $1 \leqslant j \leqslant 2^n$ and $n \geqslant 0$. An SRd grammar generating this language is obtained from the grammar G of Example 3.1, substituting in R the r-production

$$nsl(H^0, a^1) \rightarrow [2]\{r(a^2, a^1)\}$$

with the r-production

$$nsl(H^0, a^1) \rightarrow [2]\,\varnothing.$$

An SR grammar generating this language cannot contain r-items (because there is no r-item deletion). Thus it would be a C-SR grammar and (by Theorem 6.1) the language would be in N-edNCE, but it is not (by Lemma 16 of Engelfriet *et al.* 1990). ∎

In the absence of r-rewriting failure, we have no result corresponding to Theorem 6.3 above. In other words, it is an open problem whether the r-item deletion feature affects the generative power also in the absence of r-rewriting failure, namely whether $\mathscr{L}(\text{C-SRd}) \neq \mathscr{L}(\text{C-SR})$. This amounts to say that it is an open problem whether the edge deletion capability of N-edNCE, implicitly stated in the definition of the embedding mechanism, affects the generative power of this class.

Combining the results of Theorem 6.2 and 6.3, we have as an immediate consequence that the class of languages generated by SR grammars is strictly included in the class of languages generated by edNCE grammars:

THEOREM 6.4. $\mathscr{L}(SR) \subset \mathscr{L}(edNCE)$.

In addition to the above generative result, a second issue regarding the relationships between classes SR and edNCE is worth pointing out here. We have already seen that the class $\mathscr{L}(\text{SR})$ is included in NSPACE(n) (Theorem 4.2). Crucial to this result is the absence of the r-item deletion capability in SR grammars. From general results presented in Brandenburg (1983), it follows that the class $\mathscr{L}(\text{edNCE})$ is included in NSPACE(n^2). We are not aware of any inclusion result in NSPACE(n) for $\mathscr{L}(\text{edNCE})$.

6.2. Boundary Grammars

As already stated in Section 4, every SR language can be generated by an r-deterministic grammar, although examples can be given of grammars which are considerably simpler if nondeterminism in r-item rewriting is available. However, on the basis of the comparison developed in Theorem 6.1, it can be easily proven that, for certain interesting subclasses of SR grammars, the availability of nondeterministic rewriting for r-items increases the generative power (see Proposition 6.2 below). The next definition is given in analogy to the definition of Boundary node rewriting graph grammars (Engelfriet *et al.*, 1990; Rozenberg and Welzl, 1986).

DEFINITION 6.2. An SR (resp. SRd) grammar $G = (V_N, V_T, V_R, S, P, R)$ is a *Boundary SR* (resp. *Boundary SRd*) grammar if, for every r-item $r(X, Y)$ appearing in the productions of P and R, we have either $X \in C_{V_T}$ or $Y \in V_{V_T}$.

Boundary SR grammars satisfy the condition that no r-item linking two nonterminal s-items is ever generated during a derivation. As a consequence, the order in which s-items are rewritten does not affect the result of a derivation; i.e., the Church–Rosser property, also known as the *confluence* property (see Courcelle, 1987; Slisenko, 1982), holds for Boundary SR grammars. For Boundary SR grammars we prove that nondeterminism in r-item rewriting increases the generative power.

In Engelfriet *et al.* (1990) it is shown that, given a Boundary edNCE grammar, an equivalent Boundary edNCE grammar exists without blocking edges; that is $\mathscr{L}(\text{Boundary N-edNCE}) = \mathscr{L}(\text{Boundary edNCE})$. Thanks to the already established analogies between NCE-like and SR-like grammars (see Theorems 6.1 and 6.2), we can transfer the above result to the domain of Boundary ΔSRd grammars. Indeed, by analogy, the following proposition can be stated, whose proof is omitted.

PROPOSITION 6.2. *The following equivalences hold*:

$$\begin{aligned}\mathscr{L}(\text{Boundary N-edNCE}) &= \mathscr{L}(\text{Boundary } \Delta\text{C-SRd})\\ &= \mathscr{L}(\text{Boundary edNCE})\\ &= \mathscr{L}(\text{Boundary } \Delta\text{SRd}).\end{aligned}$$

The previous proposition is used to prove that nondeterministic r-rewriting affects the generative power of Boundary SRd grammars, as stated in the following theorem.

THEOREM 6.5. *The class of languages generated by Boundary* SRd *grammars properly includes the class of languages generated by Boundary* ΔSRd *grammars*; *i.e.*,

$$\mathscr{L}(\text{Boundary } \Delta\text{SRd}) \subset \mathscr{L}(\text{Boundary SRd}).$$

Proof. Engelfriet *et al.* (1990) have shown that no Boundary edNCE grammar can generate the language of all graphs on a set $\{a\}$ of node labels and a set $\{r\}$ of edge labels. Hence, by Proposition 6.2, no Boundary ΔSRd grammar can

generate such a language. On the other hand, the above language is generated by the SR grammar of Example 3.2, which is a Boundary SRd grammar. ▌

We conclude this section with a note about complexity issues. SR grammars can be fruitfully employed in the generation and in the syntactic analysis of visual languages (see, e.g., Ferrucci *et al.*, 1991). In the implementation of parsing techniques, efficiency issues can be faced taking into account theoretical results obtained in the domain of graph grammars. In particular, according to a general result due to Brandenburg (1988), three specific properties of graph grammars are essential in establishing a borderline between deterministic polynomial time and NP-hardness of the membership problem. These properties are the confluence of the grammar, the connectivity and the bounded degree of the generated graphs. Ferrucci *et al.* (1994b) exploited these results to design a predictive parsing algorithm for Boundary SR grammars by using the context-free style of s-productions and r-productions. The algorithm is an extension of Earley's algorithm for recognition of context-free (string) languages (1970), and has a polynomial time behavior when applied to confluent grammars which generate SR sentences having the additional properties of connectivity and degree-boundedness.

7. RELATED WORKS

In this section some grammatical approaches to specifying the syntax of visual languages are discussed. The literature offers a wide variety of visual language grammatical formalisms, which differ one from the other under several aspects. This is mainly due to the fact that there is no widely accepted agreement about the primitive elements of a visual language. In most cases, visual sentences are basically conceived as multisets of symbols. Some grammar models make use of attributes in order to handle information about spatial layout of symbols, which in general reflect the graphical interface nature. The attributes are an integral part of the parsing of an input sentence, in the sense that a production is applicable only if given constraints hold among attribute values. Other visual formalisms do not use attributes. They specify the relationships among the symbols in the multisets at a higher level of abstraction, which is less dependent on the underlying implementation of a graphical interface. Finally, it is intuitive that a multiset of related objects is a concept which can be associated to the abstract idea of graph. Indeed, graph grammar principles, with the corresponding rewriting approaches, have also been taken as a basis for visual and graphical grammar proposals.

In the formalism of Picture Layout Grammars (PLG's), proposed by Golin and Reiss (1990), a visual sentence is an unordered collection, namely a multiset of visual symbols with attributes containing positional information about symbols. Each production of a PLG is associated with a set of semantic functions and constraints. The semantic functions specify how the values of the attributes of the symbol on the left-hand side of the production are synthesized in terms of the attribute values of the symbols on the right-hand side. The constraints are predicates over the attribute values of the right-hand side indicating when a production can be applied. The efficient parsing algorithm proposed by the authors

works under certain restrictions. The main problem with the algorithm is that such restrictions can only be checked at runtime, giving rise to possible nonterminating computations.

Marriott (1994) has introduced a constraint-based formalism, called Constraint Multiset Grammars (CMGs), which is closely related to Picture Layout Grammars. This is another example of formalism based on multiset rewriting. A nonterminal symbol in a multiset can be rewritten by a production in the grammar whenever the attributes of the symbols in the multiset satisfy a given constraint, which describes relationships between pictures. The main difference between PLGs and CMGs is that the latter formalism supports the specification of constraints over existentially quantified symbols. In other words, a production can specify constraints over the attributes of *any* symbol in the current sentential form. Another difference is that *negative* constraints are allowed in CMGs (Chok and Marriott, 1995). Such constraints consist of tests for nonexistence of certain elements in the input sentence and they are useful in specifying visual languages by deterministic CMGs, resulting in efficient parsing.

The formalism of Positional Grammars (PGs) works on symbols that are associated with attributes (Costagliola *et al.*, 1995). Each production has the form $A \to x_1 R_1 x_2 R_2 \cdots R_{m-1} x_m, \Delta$, where A is a nonterminal, x_i is a terminal or a nonterminal, R_i is a relation (defined over the attributes of $x_1 x_2 \cdots x_{i+1}$), and Δ defines the attribute values of A depending on the attribute $x_1 x_2 \cdots x_m$. The string-like form of a PG production induces an order on the generated symbols that is used by the parsing algorithm. This feature causes a reduction in the generative power of Positional grammars with respect to PLGs and CMGs, but on the other hand, it allows efficient scanning of the parsed sentence. The proposed parsing algorithm is based on the well-known LR parsing technique. When a positional grammar allows the construction of a pLR parsing table without conflicts, the parser for such a grammar is deterministic and thus efficient.

A common feature of the formalisms considered above is the use of attributes, which are an integral part of the parsing of an input sentence. Little analysis has been made of the expressive power of these formalisms. A first reason is probably that the authors are more concerned with the implementation of parsers, and therefore with the recognition of visual languages, than with their generation. A second reason could be recognized in the possible difficulties deriving from the integration of not purely syntactic information in the parsing, in the sense that attribute values are seen as crucial parsing information. As we have shown in Section 5, manipulation of attributes can be easily introduced into the SR formalism, on the basis of the SR-trees resulting from the syntactic analysis. This way, the handling of attributes can be combined with syntactic procedures, while maintaining their semantic nature.

Wittenburg (1992) proposes a relational grammar class (Fringe Grammars) which does not exploit attributes. Wittenburg requires that relations impose strict partial orders on multisets of symbols. Restated in SR grammar terms, the idea is that, for each relation symbol α, an α-min symbol and an α-max symbol can be recognized in the right hand side of any s-production. Given that a symbol occurrence A^0 has previously been rewritten by an s-production $A^0 \to \langle \mathbf{M}, \mathbf{R} \rangle$, an r-item of the form

$\alpha(A^0, B^1)$ or $\alpha(B^1, A^0)$ is rewritten as $\alpha(\alpha\text{-max}(\mathbf{M}), B^1)$ or $\alpha(B^1, \alpha\text{-min}(\mathbf{M}))$ respectively. Actually, Wittenburg's productions have a form like $A^0 \rightarrow \langle \mathbf{M}, \mathbf{R}, \mathbf{F} \rangle$. The added third component **F** is used to explicitly specify the α-min and α-max symbols, whenever the strict partial order deriving from the r-items for the relation α in *R*, does not define them univocally. It is not difficult to interpret Fringe Grammars as a subclass of SR grammars, though r-productions are not explicitly present in Wittenburg's proposal. Indeed, rules for rewriting relation items are indirectly stated exploiting the α-min and α-max symbols. Considered as SR grammars, Fringe Grammars are complete and r-deterministic. Wittenburg shows that the given restrictions make it possible to design a predictive Earley-style parser for his class of grammars.

In the last years, graph grammars have also been used to specify the syntax of visual languages. A recent graph grammar schema for visual applications was proposed by Rekers and Shürr (1995a). It is a context-sensitive graph grammar model where the left-right hand sides of productions are extended with context-elements (Rekers and Shürr, 1995b). The context-elements are not modified by production applications but may be used as sources or targets for new relationships. More precisely, a production is defined as a pair of graphs (L, R), where L and R can have a common subgraph K, called the interface graph. This graph identifies all those context-elements in a host graph which have to be present, but are not deleted by the application of the production. Thus, no explicit rules are provided to specify the embedding phase. The presence of context-elements requires the use of quite complex parsing algorithms. Although Rekers and Shürr argue that the requirements can be dramatically reduced for real world examples, the general parsing algorithm proposed by the authors has an exponential time and space complexity.

8. FINAL REMARKS

The formalism of SR grammars is suited both for the description of pictorial scenes and for the extension of traditional syntactic and semantic techniques. Indeed, on one hand, the formalism has the ability to describe and generate sets of objects related by (spatial or not) relations, using a natural specification mechanism that rewrites relations among nonterminal objects in a context-free style until relations among terminal objects are generated. As an immediate consequence, high flexibility is achieved, since we can deal with the sentences of the specified language with different abstraction levels. On the other hand, the SR grammars are characterized by a natural and uniform derivation process, which makes use of context-free styled rewriting rules. This turned out to be particularly useful for extending well-established techniques developed for string grammars.

Structural properties of Symbol–Relation grammars have been analyzed in the paper and their influence on the generative power of the grammars has been investigated. The formalism was introduced to provide a general framework for specifying visual languages. The interpretation of such languages was addressed, and special attention was paid to the syntactic and semantic analysis issues.

A general parsing algorithm for SR grammars was presented in (Ferrucci *et al.*, 1991) and more efficient techniques were later proposed for restricted SR grammars, which maintain the ability to model practical visual languages (Tucci *et al.*, 1994; Ferrucci *et al.*, 1994b). In Ferrucci *et al.* (1994a) attributed SR grammars were introduced as a tool for the description and implementation of the semantic analysis.

The further improvement of the analysis techniques proposed so far to attain a full grasp of the power of the formalism has motivated the investigation carried out in this paper. The theoretical results derived from the analyzed properties have provided useful knowledge for the effective use of SR grammars in the specification of complex languages. Indeed, SR grammars have been proved to have a generative power comparable with that of the well known graph grammar approaches, still providing a natural and uniform rewriting mechanism more suited to implementing analysis tools for visual languages.

APPENDIX A

In this Appendix we prove that, as already stated in Theorem 3.1, SR grammars can generate any context-sensitive string language, exploiting the r-rewriting failure feature. Any string $w = a_1 \cdots a_n$, $n \geqslant 1$, can be represented by an SR sentence $\langle \mathbf{M}, \mathbf{R} \rangle$ such that $\mathbf{M} = \{a_i^{k_i} \mid k_i \geqslant 1, k_i \neq k_j \text{ for } a_i = a_j, 1 \leqslant i, j \leqslant n\}$ and $\mathbf{R} = \{left(a_{i-1}^{k_{i-1}}, a_i^{k_i}) \mid 2 \leqslant i \leqslant n\}$, where *left* is some relation symbol representing the usual string concatenation.

THEOREM 3.1. *For any context-sensitive language L an SR grammar G' exists such that $L = L(G')$.*

Proof. Let L be a context-sensitive language. Without loss of generality, we assume that L is generated by a grammar $G = (V_N, V_T, P, S)$ where each production in P has one of the forms $AB \to AC$, $A \to BC$, or $A \to a$, with $A, B, C \in V_N$ and $a \in V_T$, and that the start symbol S never occurs in the right hand side of any production in P (Penttonen, 1974). Starting from G, construct an SR grammar $G' = (V_N, V_T, V_R, P', R, S)$, where $V_R = \{leftmost, left\}$. Sets P' and R are specified as follows:

(i) for each production $S \to AB$ in P, add to P' the s-production

$$l_1 : S^0 \to \langle \{A^2, B^3\}, \{leftmost(A^2, B^3)\} \rangle;$$

(ii) for each production $A \to a$ in P, add to P' the s-production

$$l_2 : A^0 \to \langle \{a^2\}, \varnothing \rangle$$

and add to R the r-productions

$$left(A^0, X^1) \to [l_2]\{left(a^2, X^1)\}$$
$$left(X^1, A^0) \to [l_2]\{left(X^1, a^2)\}$$
$$leftmost(A^0, X^1) \to [l_2]\{left(a^2, X^1)\}$$
$$leftmost(N^1, A^0) \to [l_2]\{leftmost(N^1, a^2)\},$$

for each $X \in (V_N \cup V_T)$ and $N \in V_N$;

(iii) for each production $A \to BC$ in P, add to P' the s-productions

$$l_3: A^0 \to \langle \{B^2, C^3\}, \{leftmost(B^2, C^3)\} \rangle$$
$$l_4: A^0 \to \langle \{B^2, C^3\}, \{left(B^2, C^3)\} \rangle,$$

and add to R the r-productions

$$leftmost(A^0, X^1) \to [l_3]\{left(C^3, X^1)\}$$
$$leftmost(N^1, A^0) \to [l_4]\{leftmost(N^1, B^2)\}$$
$$left(A^0, X^1) \to [l_4]\{left(C^3, X^1)\}$$
$$left(X^1, A^0) \to [l_4]\{left(X^1, B^2)\},$$

for each $X \in (V_N \cup V_T)$ and $N \in V_N$;

(iv) for each production $AB \to AC$ in P, add to P' the s-production

$$l_5: B^0 \to \langle \{C^2\}, \varnothing \rangle$$

and add to R the r-productions

$$left(A^1, B^0) \to [l_5]\{left(A^1, C^2)\}$$
$$leftmost(A^1, B^0) \to [l_5]\{leftmost(A^1, C^2)\}$$
$$left(B^0, X^1) \to [l_5]\{left(C^2, X^1)\},$$

for each $X \in (V_N \cup V_T)$.

It is easy to see that every derivation in G has an equivalent derivation in G'. This entails that $L(G) \subseteq L(G')$. To show the converse, we focus on the s-productions introduced in (iv) above. Consider an SR sentence $\langle \mathbf{M}, \mathbf{R} \rangle$ generated by G' from the start symbol. It is not difficult to see that $\mathbf{R}$ has one of the forms

(a) $leftmost\ (\mathsf{N}_1, \mathsf{X}_2)$, $left(\mathsf{X}_2, \mathsf{X}_3)$, ..., $left(\mathsf{X}_{k-1}, \mathsf{X}_1)$, or

(b) $left\ (a_1, \mathsf{X}_2)$, $left(\mathsf{X}_2, \mathsf{X}_3)$, ..., $left(\mathsf{X}_{k-1}, \mathsf{X}_k)$,

where $N_1 \in C_{V_N}$, $X_i \in C_{V_N \cup V_T}$, and $a_1 \in C_{V_T}$, which is the assumed SR representation of strings. Thus the application of an s-production of type (iv) is feasible only when an occurrence of A is left adjacent to the occurrence of B to be rewritten. Indeed, the s-production $\langle B^0 \to \{C^2\}, \varnothing \rangle$ cannot be applied if the symbol "B" occurs in the leftmost position of the sentence. ▮

APPENDIX B

In this appendix the equivalence between C-SRd grammars and N-edNCE grammars is proved in detail. We first recall the definition of directed graphs with labelled nodes and edges. Then, the definitions of edNCE and N-edNCE graph grammars and of their derivation processes are given. The definition of edNCE is taken from Engelfriet (1990). The definition of derivation step is an equivalent variation of the one in the same paper. In edNCE, "e" refers to the presence of labels on edges, "d" stands for directed, and "NCE" for Neighborhood Controlled Embedding.

DEFINITION B.1. A directed *node- and edge-labelled graph* (*EDG-graph*) over Σ and Γ is a quintuple $D = (V, E, \Sigma, \Gamma, \phi)$, where V is the finite, nonempty set of nodes, Σ is the finite, nonempty set of node labels, Γ is the finite set of edge labels, E is the set of edges of the form (v, λ, w), where $v, w \in V$, $\lambda \in \Gamma$, and $\phi: V \to \Sigma$ is the *node labelling function*. The class of all EDG-graphs over Σ and Γ is denoted by $EDG_{\Sigma, \Gamma}$.

In what follows, the symbols V_D, E_D, and ϕ_D will denote the node set, the edge set, and the labelling function of an EDG-graph D.

DEFINITION B.2. An *edNCE graph grammar* is a system $G = (\Sigma, \Theta, \Gamma, \Omega, P, S)$, where Σ is the alphabet of node labels, $\Theta \subseteq \Sigma$ is the alphabet of *terminal* node labels, Γ is the alphabet of edge labels, $\Omega \subseteq \Gamma$ is the alphabet of *final* edge labels, P is the finite set of productions, and $S \in \Sigma - \Theta$ is the initial nonterminal. A production $\pi \in P$ has the form $\pi = (A, D, B_{\text{in}}, B_{\text{out}})$, where $A \in \Sigma - \Theta$, $D \in EDG_{\Sigma, \Gamma}$, and both B_{in} and B_{out} are subsets of $V_D \times \Gamma \times \Gamma \times \Sigma \times \{in, out\}$.

Now we informally describe how a derivation step is performed. A production $\pi = (A, D, B_{\text{in}}, B_{\text{out}})$ is applied to a graph $H \in EDG_{\Sigma, \Gamma}$ in order to replace a node v such that $\phi(v) = A$ (the *mother node*) with the graph D (the *daughter graph*). First, v is removed from H, together with all the edges that are incident with v, obtaining a graph H'. Then D is embedded in H' adding edges between nodes of D and nodes of H', according to the *embedding rules* contained in B_{in} and B_{out}. More precisely:

— if the edge (v, r, x) (resp. (x, r, v)) is in H, with $\phi_H(x) = \rho$, and $(y, r, s, \rho, \text{in}) \in B_{\text{out}}$ (resp. B_{in}), then the edge (x, s, y) is added to the resulting graph,

— if the edge (v, r, x) (resp. (x, r, v)) is in H, with $\phi_H(x) = \rho$, and $(y, r, s, \rho, \text{out}) \in B_{\text{out}}$ (resp. B_{in}), then the edge (y, s, x) is added to the resulting graph.

A node is said to be terminal if it has a terminal label, and an edge is said to be final if it has a final label. The language generated by an edNCE grammar is the set of all graphs that can be produced by a chain of derivation steps and contain terminal nodes and final edges only. In other words, graphs with nonfinal edges between terminal nodes are not accepted in the generated language. Such edges are called *blocking edges*.

The class of N-edNCE grammars can be defined as the subset of edNCE grammars that cannot generate blocking edges. It is easy to see that in the class of N-edNCE grammars the distinction between final and nonfinal edge labels is ineffective. In other words, given an N-edNCE grammar, we can always assume that the set of nonfinal edge labels is empty.

An evident correspondence can be established between $EDG_{\Sigma, \Gamma}$ graphs and SR sentences on Γ and Σ. We can indeed associate a symbol occurrence to a labelled node and an r-item to a labelled directed edge. Thus, a correspondence relation can be established between graphs and SR sentences, which will be denoted $\cong_{\Sigma\Gamma}$ in the following. Given an SR sentence $\sigma = \langle \mathbf{M}, \mathbf{R} \rangle$ and a graph D such that $\sigma \cong_{\Sigma\Gamma} D$, a one-to-one mapping exists between symbol occurrences in $\mathbf{M}$ and nodes in D, which will be denoted by $\Phi_{\sigma, D}$ in the following. Finally, let us recall that the notion of r-deterministic SR grammar, introduced in Section 3, can be extended to SRd

and C-SRd grammars in the obvious way. We are now ready to prove that the class of r-deterministic C-SRd (ΔC-SRd, for short) languages equals the class of N-edNCE languages.

THEOREM 6.1. *The class of* C-SRd *grammars is equivalent to the class of* N-edNCE *graph grammars. More precisely, the following equivalences hold*:

$$\mathscr{L}(\text{C-SRd}) = \mathscr{L}(\Delta\text{C-SRd}) = \mathscr{L}(\text{N-edNCE}).$$

Proof. Let us prove that $\mathscr{L}(\text{N-edNCE}) \subseteq \mathscr{L}(\Delta\text{C-SRd})$. We consider an N-edNCE graph grammar $G = (\Sigma, \Theta, \Gamma, \Omega, P, S)$ and construct an equivalent ΔC-SRd grammar $G' = (V_N, V_T, V_R, S, P', R')$, where $V_N = (\Sigma - \Theta)$, $V_T = \Theta$, and $V_R = \Gamma$. The sets P' and R' are specified as follows. For each production $(A, D, B_{\text{in}}, B_{\text{out}})$ in P:

(i) add to P' the s-production $l\colon A^0 \to \sigma$, where $\sigma \cong_{\Sigma,\Gamma} D$ (for a suitable l);

(ii) for each $r \in \Gamma$ and $X \in \Sigma$:

(a) add to R' the r-production $r(X^1, A^0) \to [l]\ \mathbf{Q}$, where

$$\mathbf{Q} = \{s(X^1, Y^j) \mid (\Phi_{\sigma, D}(Y^j), r, s, X, \text{in}) \in B_{\text{in}}\} \cup \{s(Y^j, X^1) \mid (\Phi_{\sigma, D}(Y^j), r, s, X, \text{out} \in B_{\text{in}})\},$$

(b) add to R' the r-production $r(A^0, X^1) \to [l]\ \mathbf{Q}$ where

$$\mathbf{Q} = \{s(Y^j, X^1) \mid (\Phi_{\sigma, D}(Y^j), r, s, X, \text{out}) \in B_{\text{out}}\} \cup \{s(X^1, Y^j) \mid (\Phi_{\sigma, D}(Y^j), r, s, X, \text{in}) \in B_{\text{out}}\}.$$

To prove that $\mathscr{L}(\Delta\text{C-SRd}) \subseteq \mathscr{L}(\text{N-edNCE})$, let us consider a ΔC-SRd grammar $G = (V_N, V_T, V_R, S, P, R)$ and construct an equivalent N-edNCE graph grammar $G' = (\Sigma, \Theta, \Gamma, \Omega, P', S)$, where $\Sigma = V_N \cup V_T$, $\Theta = V_T$, and $\Gamma = \Omega = V_R$. The set of productions P' is obtained as follows. For each s-production $l\colon A^0 \to \sigma$ in P, add to P' the production $(A, D, B_{\text{in}}, B_{\text{out}})$, where

$$D \cong_{\Sigma\Gamma} \sigma,$$

$$B_{\text{in}} = \{(\Phi_{\sigma, D}(Y^j), r, s, X, \text{in}) \mid r(X^1, A^0) \to [l]\ \mathbf{Q} \in R, s(X^1, Y^j) \in \mathbf{Q}\} \cup \{(\Phi_{\sigma, D}(Y^j), r, s, X, \text{out}) \mid r(X^1, A^0) \to [l]\ \mathbf{Q} \in R, s(Y^j, X^1) \in \mathbf{Q}\},$$

$$B_{\text{out}} = \{(\Phi_{\sigma, D}(Y^j), r, s, X, \text{out}) \mid r(A^0, X^1) \to [l]\ \mathbf{Q} \in R, s(Y^j, X^1) \in \mathbf{Q}\} \cup \{(\Phi_{\sigma, D}(Y^j), r, s, X, \text{in}) \mid r(A^0, X^1) \to [l]\ \mathbf{Q} \in R, s(X^1, Y^j) \in \mathbf{Q}\}.$$ ∎

ACKNOWLEDGMENT

The authors thank the referees for their helpful comments and constructive criticisms.

Received March 28, 1995; accepted in revised form August 1, 1996

REFERENCES

Aho, A. V., Sethi, R., and Ullman, J. D. (1986), "Compilers—Principles, Techniques and Tools," Addison–Wesley, Reading, MA.

Angelaccio, M., Catarci, T., and Santucci, G. (1990), QBD*: A graphical query language with recursion, *IEEE Trans. Software Enging*. **16**, No. 10, 1150–1163.

Barbar, K. (1993), Attributed tree grammars, *Theoret. Comput. Sci.* **119**, 3–22.

Brandenburg, F. J. (1983), The computational complexity of certain graph grammars, *in* "Lecture Notes in Computer Science," Vol. 145, pp. 91–99, Springer-Verlag, Berlin/New York.

Brandenburg, F. J. (1988), On polynomial time graph grammars, *in* "Lecture Notes in Computer Science," Vol. 294, pp. 227–236, Springer-Verlag, Berlin/New York.

Bunke, H. (1982), Attributed programmed graph grammars and their applications to schematic diagram interpretation, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-4**, **6**, No. 6, 574–582.

Chang, S. K., Ed. (1990a), "Visual Languages and Visual Programming," Plenum, New York.

Chang, S. K. (1990b), A visual language compiler for information retrieval by visual reasoning, *IEEE Trans. Software Eng*. **16**, No. 10, 1136–1149.

Chok, S. S., and Marriott, K. (1995), Parsing visual languages, *in* "18th Australasian Computer Science Conference, Glenolg."

Costagliola, G., De Lucia, A., Orefice, S., and Tortora, G. (1995), Automatic generation of visual programming environments, *IEEE Comput*. **28**, 56–66.

Courcelle, B. (1987), An axiomatic definition of context-free rewriting and its application to NLC graph grammars, *Theoret. Comput. Sci.* **55**, 141–181.

Crimi, C. Guercio, A., Nota, G., Pacini, G., Tortora, G., and Tucci, M. (1991), Relation grammars and their application to multi-dimensional languages, *J. Visual Languages Comput*. **2**, 333–346.

Crimi, C. Guercio, A., Pacini, G., Tortora, G., and Tucci, M. (1990), Automating visual language generation, *IEEE Trans. Software Enging*. **16**, No. 10, 1122–1135.

Earley, J. (1970), An efficient context-free parsing algorithm, *Comm. ACM* **13**, 94–102.

Ehrenfeucht, A., and Rozenberg, G. (1984), Restrictions on NLC graph grammars, *Theoret. Comput. Sci.* **31**, 211–223.

Engelfriet, J. (1990), Context-free NCE graph grammars, *in* "Lecture Notes in Computer Science," Vol. 380, pp. 148–161, Springer-Verlag, Berlin/New York.

Engelfriet, J., and Filè, G. (1981), The formal power of one-visit attribute grammars, *Acta Informat*. **16**, 275–302.

Engelfriet, J., and Leih, G. (1989), Linear graph grammars: Power and complexity, *Inform. and Comput*. **81**, 88–121.

Engelfriet, J., Leih G., and Welzl, E. (1990), Boundary graph grammars with dynamic edge relabelling, *J. Comput. System Sci*. **40**, 307–345.

Ferrucci, F., Pacini, G., Tortota, G., Tucci, M., and Vitiello, G. (1991), Efficient parsing of multi-dimensional structures, *in* "Proceedings, IEEE Workshop on Visual Languages, Kobe," pp. 105–110.

Ferrucci, F., Tortota, G., and Tucci, M. (1994a), Semantics of visual languages, *in* "Proceedings of International Workshop on Advanced Visual Interfaces, Bari," pp. 219–221.

Ferrucci, F., Tortota, G., Tucci, M., and Vitiello, G. (1994b), A predictive parser for visual languages specified by relation grammars, *in* "Proceedings, IEEE Symposium on Visual Languages, St. Louis, Missouri," pp. 245–252.

Glinert, E. P., and Tanimoto, S. L. (1984), Pict: An interactive graphical programming environment, *IEEE Comput*. **17**, 7–25.

Golin, E. J., and Reiss, S. P. (1990), The specification of visual language syntax, *J. Visual Languages Comput*. **1**, 141–157.

Gottler, H. (1982), Attributed graph grammars for graphics, *in* "Proceedings, 2nd Int. Workshop on Graph Grammars and their Applications to Computer Science" (H. Ehrig, M. Nagl, and

G. Rozenberg, Eds.), Lecture Notes in Computer Science, Vol. 153, pp. 130–142, Springer-Verlag, Berlin/New York.

Gottler, H. (1989), Graph grammars, a new paradigm for implementing visual languages, *in* "Proceedings, 2nd European Software Engineering Conference" (C. Ghezzi and J. A. McDermid, Eds.), Lecture Notes in Computer Science, Vol. 387, pp. 336–350, Springer-Verlag, Berlin/New York.

Harel, D. (1987), StateCharts: A visual formalism for complex systems, *Sci. Comput. Programming* **8**, 231–274.

Harel, D. (1988), On visual formalisms, *Comm. ACM* **31**, 514–530.

Hirakawa, M., Tanaka, M., and Ichikawa, T. (1990), An iconic programming system, HI-VISUAL, *IEEE Trans. Software Enging.* **16**, No. 10, 1178–1184.

Hopcroft, J. E., and Ullman, J. D. (1979), "Introduction to Automata Theory, Languages, and Computation," Addison–Wesley, Reading, MA.

Jacob, R. J. K. (1985), A state transition diagram language for visual programming, *IEEE Comput.* **18**, No. 8, 51–59.

Janssens, D., and Rozenberg, G. (1980a), On the structure of node-label controlled graph languages, *Inform. Sci.* **20**, 191–216.

Janssens, D., and Rozenberg, G. (1980b), Restrictions, extensions and variations of NLC grammars, *Inform. Sci.* **20**, 217–244.

Janssens, D., and Rozenberg, G. (1983), Graph grammars with node-label controlled rewriting and embedding, *in* "Proceedings of the 2nd International Workshop on Graph-Grammars and Their Application to Computer Science" (H. Ehrig, M. Nagl, and G. Rozenberg, Eds.), Lecture Notes in Computer Science, Vol. 153, pp. 186–205, Springer-Verlag, Berlin.

Janssens, D., Rozenberg, G., and Verraedt, R. (1982), On sequential and parallel node-rewriting graph grammars, *Comput. Graphics Image Process.* **18**, 279–304.

Kaplan, S. M., Goering, S. K., and Campbell, R. H. (1987), Supporting the software development process with attributed NLC grammars, *in* "Proceedings, 3rd International Workshop on Graph Grammars and Their Applications to Computer Science" (H. Ehrig, M. Nagl, and G. Rozenberg, Eds.), Lecture Notes in Computer Science, Vol. 291, pp. 309–325, Springer-Verlag, Berlin/New York.

Knuth, D. E. (1968), Semantics of context-free languages, *Math. Systems Theory* **2**, 127–145.

Kreowski, H. J., and Rozenberg, G. (1990a), On structured graph grammars. I, *Inform. Sci.* **52**, 185–210.

Kreowski, H. J., and Rozenberg, G. (1990b), On structured graph grammars. II, *Inform. Sci.* **52**, 221–246.

Marriott, K. (1994), Constraint multiset grammars, *in* "Proceedings, IEEE Symposium on Visual Languages, St. Louis, Missouri," pp. 118–125.

Penttonen, M. (1974), One-sided and two-sided context in formal grammars, *Inform. and Control* **25**, 371–392.

Pfeiffer, J. J. (1992), Parsing graphs representing two dimensional figures, *in* "Proceedings of the 1992 IEEE Workshop on Visual Languages, Seattle," pp. 200–206.

Rekers, J., and Schürr, A. (1995a), "A Parsing Algorithm for Context-Sensitive Graph Grammars" (long version), Technical Report 95-05, Leiden University, The Netherlands.

Rekers, J., and Schürr, A. (1995b), A graph grammar approach to graphical parsing, *in* "Proceedings of the 1995 IEEE Symposium on Visual Languages, Darmstadt."

Rozenberg, G. (1987), An introduction to the NLC way of rewriting graphs, *in* "Lecture Notes in Computer Science," Vol. 291, pp. 55–66, Springer-Verlag, Berlin/New York.

Rozenberg, G., and Welzl, E. (1986), Boundary NLC graph grammars—Basic definitions, normal forms, and complexity, *Inform. and Control* **69**, 136–167.

Shi, Q. Y., and Fu, K. S. (1983), Parsing and translation of attribute expansive graph languages for scene analysis, *IEEE Trans. Pattern Anal. Mach. Intell.* **5**, 472–485.

Slisenko, A. O. (1982), Context-free grammars as a tool for describing polynomial-time subclasses of hard problems, *Inform. Process. Lett.* **14**, 52–56.

Tucci, M., Vitiello, G., and Costagliola, G. (1994), Parsing nonlinear languages, *IEEE Trans. Software Enging*. **20**, No. 9, 720–739.

Welzl, E., (1984), Encoding graphs by derivations and implications for the theory of graph grammars, *in* "Lecture Notes in Computer Science," Vol. 172, pp. 503–513, Springer-Verlag, Berlin/New York.

Wittenburg, K. and Weitzman, L. (1990), Visual grammars and incremental parsing for interface languages, *in* "Proceedings, IEEE Workshop on Visual Languages, Skokie, Illinois," pp. 111–118.

Wittenburg, K. (1992), Earley-style parsing for relational grammars, *in* "Proceedings, IEEE Workshop on Visual Languages, Seattle," pp. 192–199.

Yamazaki, K., and Yaku, T. (1993), A pumping lemma and the structure of derivations in the Boundary NLC graph languages, *Inform. Sci.* **75**, 81–97.