JOURNAL OF COMPUTER AND SYSTEM SCIENCES 40, 307–345 (1990)

Boundary Graph Grammars with Dynamic Edge Relabeling

JOOST ENGELFRIET AND GEORGE LEIH*

Department of Computer Science, University of Leiden, P.O. BOX 9512, 2300 RA Leiden, The Netherlands

AND

EMO WELZL†

F. B. Mathematik, Freie Universität Berlin, Arnimallee 2–6, 1000 Berlin 33, West-Germany

Received April 21, 1988; revised December 20, 1988

Most NLC-like graph grammars generate node-labeled graphs. As one of the exceptions, eNCE graph grammars generate graphs with edge labels as well. We investigate this type of graph grammar and show that the use of edge labels (together with the NCE feature) is responsible for some new properties. Especially boundary eNCE (B-eNCE) grammars are considered. First, although eNCE grammars have the context-sensitive feature of "blocking edges," we show that B-eNCE grammars do not. Second, we show the existence of a Chomsky normal form and a Greibach normal form for B-eNCE grammars. Third, the boundary eNCE languages are characterized in terms of regular tree and string languages. Fourth, we prove that the class of (boundary) eNCE languages properly contains the closure of the class of (boundary) NLC languages under node relabelings. Analogous results are shown for linear eNCE grammars.

INTRODUCTION

Graph grammars provide a mechanism in which local transformations on graphs can be modelled in a mathematically precise way. This is done by specifying a set of productions; a production is usually a triple (H_1, H_2, Emb), where H_1 and H_2 are graphs, and Emb is some embedding mechanism. Such a production can be applied to a graph H whenever there is an occurrence of H_1 in H, i.e., a subgraph of H isomorphic with H_1. It is applied by removing this occurrence (and the

* The work of this author was conducted as part of the PRISMA project, a joint effort with Philips Research, partially supported by the Dutch "Stimuleringsprojectteam informaticaonderzoek" (SPIN).

† Research associate of: IIG, Institutes for Information Processing, Technical University of Graz, Austria.

0022-0000/90 $3.00

"dangling" edges) from H, replacing it by H_2, and finally using Emb to connect H_2 to the remainder of H. Moreover, an application condition can be associated to the production which determines whether the occurrence of H_1 in H may be replaced, depending on the context of this occurrence. For an overview of different types of graph grammar, see [Nag, EhrNagRoz, EhrNagRozRos].

We are interested in graph grammars which are context-free in the sense that (1) H_1 is a single-node graph without edges, and (2) productions have no application conditions. Moreover, we restrict ourselves to the natural case that (3) the embedding mechanism only establishes edges between nodes in H_2 and former neighbours of H_1. Graph grammars of this type belong to the NLC family op graph grammars: NLC, dNLC, eNLC, NCE, etc., where NLC stands for node label controlled, NCE for neighbourhood controlled embedding, the *d* for directed, and the *e* for edge labeled (see, e.g., [JanRoz1–JanRoz5], and [JanRozVer]). Here we study an interesting variant that has not got much attention before: the eNCE graph grammars (see [Kau1, Kau2, Bra, EngLei1, Englei3]). The two essential features of this variant are the following. First (the "*e*"), eNCE grammars generate undirected graphs with labeled edges and nodes (in contrast with ordinary NLC grammars which have no edge labels). These edge labels are manipulated by the grammar in an essential way, which is why we call it a grammar with dynamic edge relabeling. Second (the "NCE"), each node in H_2 can be treated separately by Emb in eNCE grammars (whereas all nodes in H_2 with the same label are treated identically in NLC grammars). These two features are formally expressed by the fact that Emb consists of tuples (x, λ, μ, b), where x is a node in H_2, λ and μ are edge labels, and b is a node label. Such a tuple causes Emb to establish a μ-labeled edge between x and y if there was a λ-labeled edge between H_1 and y, and y has label b.

In this paper and in [EngLei1], a companion paper, we claim that the eNCE grammar has many advantages over other grammars of the NLC family. Results which one would naturally expect of a type of graph grammar satisfying (1)–(3) often hold for eNCE and can be proved in a natural way, using the special features of the model (both the dynamic edge relabeling and the neighbourhood controlled embedding). In particular, the eNCE grammar is rather insensible to small changes in the model that one would expect to be harmless. Altogether, eNCE grammars are easy to work with.

Unfortunately, the eNCE grammar has two properties which make it not as context-free as we would like it to be. First, like the NLC grammar, it is not confluent (cf. [Cou]): when two (neighbouring) nodes in a graph can be replaced, then the order in which they are replaced may influence the result. Second, it has the property of "blocking edges" (cf. [Nag]). To explain this, we have to give some more details of the eNCE grammar. As usual it has terminal and nonterminal node labels, but additionally it has final and nonfinal edge labels. Nonterminal nodes may be replaced, whereas nonfinal edge labels are meant to contain information to be used by the embedding relation. A generated graph is in the language of the grammar if it has terminal nodes and final edges only. A blocking edge is now a nonfinal edge that connects two terminal nodes. Thus, graphs containing such an

edge, and all graphs derivable from it, are not in the language. These blocking edges therefore function as a kind of application condition: if a blocking edge will be generated by applying a production π, then π should in fact not be applied. This property can be used to generate "context-sensitive" languages, such as a graph language consisting of graphs with 2^n nodes. The analogous situation in which no production is applicable to a nonterminal node does not give problems: an eNCE grammar can easily be reduced (cf. [Jef]), just like a context-free string grammar.

Our main object of investigation in this paper is the class of boundary eNCE (or B-eNCE) grammars: in such a grammar nonterminal nodes are not allowed to be adjacent. This boundary restriction guarantees context-freeness in addition to (1)–(3), in the sense that the two problems mentioned above for arbitrary eNCE grammars are avoided: boundary grammars are confluent and blocking edges need not occur. Moreover, it guarantees that only "tree-like" graphs are generated, which are often easier to handle than arbitrary graphs. The boundary restriction on NLC grammars is studied in [RozWel1, RozWel2, RozWel3], resulting in some nice properties with respect to, e.g., normal forms, decidability, closure properties, and complexity of recognition. In our experience the B-eNCE grammar has all these nice properties too; we show, moreover, some additional results in this paper which cannot be shown for the boundary NLC grammar. These results suggest that boundary eNCE grammars are even nicer than boundary NLC grammars, a class of grammars which is commonly considered to be one of the most promising types of NLC-like graph grammars.

One of the simplest natural restrictions on graph grammars is linearity: in the graphs generated by a linear grammar there is at most one nonterminal node. Thus linear eNCE (or LIN-eNCE) grammars form a natural subclass of the B-eNCE grammars. Since linear grammars generate "chain-like" graphs, they are even easier to handle than boundary grammars. In [EngLei1] LIN-eNCE grammars form the main object of investigation. In this paper we treat linear grammars along with boundary grammars. As another, less important, special case we consider apex grammars, first studied in [EngLeiRoz1, EngLeiRoz2].

Before discussing the main results of this paper, we want to stress that they strongly depend on the special features of our model: analogous results are difficult to obtain (or cannot be obtained) for other graph grammar models. Hence, a general setup, as in [Jef], cannot be given. We now discuss the three main results of this paper.

First, there exists a Chomsky normal form and a Greibach normal form for B-eNCE grammars (and for LIN-eNCE grammars). As far as we know, there is no other graph grammar model for which such normal forms have been shown. Only in [Nag], a Chomsky-like normal form has been found, with chain-productions still allowed, but, in the proof, blocking edges are used. For many other models, in particular for the boundary NLC grammar, the non-existence of a Chomsky and a Greibach normal form has been proved ([Wel1, HabKre, RozWel1, EhrMaiRoz]).

Second, B-eNCE languages can be characterized in an elegant way by means of regular tree and string languages. Each graph in a B-eNCE language can be

obtained from a tree in a certain regular tree language by adding edges as follows: an edge between a node x in the tree and one of its descendants is established whenever the string of node labels on the path from x to y through the tree is in a certain regular language. The edges of the tree itself disappear, and a node relabeling is applied to the nodes. Vice versa, each set of graphs obtained from regular tree and string languages as described above is a B-eNCE language. This shows that a B-eNCE graph language is "tree-like": it consists in fact of "transitively closed" trees from which some of the edges are deleted. LIN-eNCE languages can be characterized in terms of regular string languages only, in a similar way.

Third, B-eNCE grammars (generating graphs without edge labels) are more powerful than B-NLC grammars: we show that B-eNCE properly contains the closure of B-NLC under node relabelings. The same holds for linear and arbitrary eNCE and NLC grammars. Apex grammars form an exception here: A-eNCE grammars have the same power as A-NLC grammars with a node relabeling.

These major results are based on several minor facts which one would "naturally expect" from a reasonable type of graph grammar, e.g., B-eNCE is closed under node (and edge) relabelings, chain and Λ-productions can be removed from B-eNCE grammars, and (as mentioned before) B-eNCE grammars can do without blocking edges. These handy results strengthen our opinion that the B-eNCE grammar is nice to work with.

The paper is organized as follows. The preliminaries can be found in Section 1. In Section 2, the basic definitions are given, together with some examples. Furthermore, some elementary results are proved which will be used through the whole paper. In Section 3 we investigate the blocking edge problem. It is shown that boundary eNCE grammars can do without this feature, whereas general eNCE grammars cannot. Sections 4, 5, and 6 contain our main results. The Chomsky and the Greibach normal form results for boundary and linear grammars are established in Section 4. In Section 5 the Greibach normal form result is used to derive the characterization results for boundary and linear grammars. A direct consequence is that LIN-eNCE is closed under edge-complement (but B-eNCE is not). Section 6 shows that (boundary, linear) eNCE grammars are more powerful than (boundary, linear) NLC grammars with a relabeling. In the apex case, eNCE grammars and NLC grammars with a relabeling have the same power.

Finally we wish to mention that the report version of this paper ([EngLeiWel]) contains two additional sections. Section 7 of [EngLeiWel] is concerned with another generalization of the B-NLC grammars: the partition controlled (PC) grammars of [Wel2]. It is shown that PC grammars have the same generating power as B-eNCE grammars that generate graphs without edge labels. In Section 8 of [EngLeiWel] directed eNCE graph grammars are defined (see also [EngLei Roz2]), and the existence of a Chomsky-like normal form is shown for directed eNCE grammars without blocking edges. It is not difficult to see that the obvious analogous of all results of this paper can also be obtained for directed graphs.

1. PRELIMINARIES

In this section we discuss some notation and terminology used in this paper. We assume the reader to be familiar with elementary concepts from formal language theory [Ber, Sal], and from graph theory [Har]. For a set A, $\#A$ denotes the cardinality of A, and $\mathcal{P}(A)$ is the set of all subsets of A.

An *alphabet* is a finite set. If Σ is an alphabet, then Σ^* denotes the set of all strings over Σ. A *context-free (string) grammar* will be denoted here as a four-tuple $G=(\Sigma, \Delta, P, S)$, where Σ is the alphabet of symbols, $\Delta \subseteq \Sigma$ is the alphabet of terminals ($\Sigma - \Delta$ is the alphabet of nonterminals), P is the set of productions, and $S \in \Sigma - \Delta$ is the initial nonterminal. Productions are of the form $X \to \xi$, with $X \in \Sigma - \Delta$, and $\xi \in \Sigma^*$. G is called a *right-linear grammar* if each production in P is of the form $X \to aY$ or $X \to a$, with $X, Y \in \Sigma - \Delta$ and $a \in \Delta$. A *regular language* is a set of strings that can be generated by a right-linear grammar.

A *node- and edge-labeled graph*, or just *graph*, is a system $H=(V, E, \Sigma, \Gamma, \varphi)$, where V is the finite set of *nodes*, Σ is the alphabet of *node labels*, Γ is the alphabet of *edge labels*, $E \subseteq \{(\{v, w\}, \lambda) \mid v, w \in V,\ v \neq w,\ \lambda \in \Gamma\}$ is the set of *edges*, and $\varphi: V \to \Sigma$ is the *node labeling function*. Thus we consider undirected graphs without loops; multiple edges between the same pair of nodes are allowed, but they should be labeled differently. Whenever a graph H is considered, its set of nodes, set of edges, set of node labels, set of edge labels, and node labeling function will be denoted by V_H, E_H, Σ_H, Γ_H, and φ_H, respectively. For better readability, an edge $(\{v, w\}, \lambda)$ will be denoted (v, λ, w) or (w, λ, v) in the sequel, so (v, λ, w) and (w, λ, v) denote the same edge; λ is said to be the label of (v, λ, w).

Let H be a graph. If $V_H = \varnothing$ then H is called the *empty graph* and is denoted Λ. If V_H is a singleton then H is called a *singleton graph*. A singleton graph H with, say, $V_H = \{v\}$ and $\varphi_H(v) = X$ for some $X \in \Sigma_H$, will also be denoted X. H is called *discrete* if $E_H = \varnothing$. If $E_H = \{(v, \lambda, w) \mid v, w \in V_H,\ v \neq w,\ \lambda \in \Gamma_H\}$, then H is called *complete*. H is called a graph *without edge labels* in case $\Gamma_H = \{*\}$, where $*$ is a reserved symbol.

A graph $H=(V, E, \Sigma, \Gamma, \varphi)$ is called a graph over Σ and Γ. For alphabets Σ and Γ, the set of all graphs over Σ and Γ is denoted $GR_{\Sigma, \Gamma}$. A subset of $GR_{\Sigma, \Gamma}$ is called a *graph language*. We consider graph languages to be the same if they only differ with respect to the empty graph Λ.

Let H and K be graphs over Σ and Γ. H and K are *isomorphic* if there is a bijection $h: V_H \to V_K$ such that $E_K = \{(h(v), \lambda, h(w)) \mid (v, \lambda, w) \in E_H\}$ and, for all $v \in V_H$, $\varphi_K(h(v)) = \varphi_H(v)$. As usual, isomorphic graphs are often identified, in particular, in graph languages. It should be clear from the context when isomorphic graphs are considered the same.

Let H be a graph. For $(v, \lambda, w) \in E_H$, we say that the edge (v, λ, w) is *incident with* (or *connects*) the nodes v and w (and vice versa, v and w are said to be incident with (v, λ, w)); moreover, we say that v and w are *neighbours*. Furthermore, for $v \in V_H$, the *context* of v in H is $\text{context}_H(v) = \{(a, \lambda) \in \Sigma_H \times \Gamma_H \mid \exists w \in V_H: (w, \lambda, v) \in E_H, \varphi_H(w) = a\}$, and the *neighbourhood* of v in H is $\text{neigh}_H(v) = \{w \in V_H \mid w \text{ and } v \text{ are}$

neighbours}. H is called *connected* if there is a path from each node in H to each other node. A graph language L is *connected* if each graph in L is connected. A graph language L is of *bounded degree* if the number of edges incident with any node in any graph in L is bounded.

Let H be a graph, and let $V \subseteq V_H$. Then the *subgraph of H induced by V* is the graph $(V, \bar{E}, \Sigma_H, \Gamma_H, \bar{\varphi})$, where $\bar{E} = \{(v, \lambda, w) \in E_H \mid v, w \in V, \lambda \in \Gamma_H\}$, and $\bar{\varphi}$ equals φ restricted to V. If $v \in V_H$, then $H - \{v\}$ is the subgraph of H induced by $V_H - \{v\}$.

Let Σ and Δ be alphabets. A mapping $\rho: \Sigma \to \Delta$ is called a *node relabeling*. For a graph $H = (V, E, \Sigma, \Gamma, \varphi)$, $\rho(H) = (V, E, \Delta, \Gamma, \rho \circ \varphi)$. For a graph language $L \subseteq GR_{\Sigma,\Gamma}$, $\rho(L) = \{\rho(H) \mid H \in L\}$. The class of all node relabelings is denoted R. For a class of graph languages X, $R(X) = \{\rho(L) \mid \rho \in R, L \in X\}$. X is *closed under node relabeling* if $R(X) \subseteq X$.

Let Σ be an alphabet, and let $a \in \Sigma$. For a string w over Σ, $\#_a(w)$ denotes the number of occurrences of a in w. For a graph H with node labels from Σ, $\#_a(H)$ denotes the number of nodes in H with label a.

This paper is concerned with undirected graphs, as defined above. However, in Section 5 we also use directed graphs. A *directed graph* is a system $H = (V, E, \Sigma, \Gamma, \varphi)$, where V, Σ, Γ, and φ are as defined above for a graph, and E, the set of edges, is a subset of $\{(v, \lambda, w) \mid v, w \in V, v \neq w, \lambda \in \Gamma\}$. Note that we use (v, λ, w) to stand for a directed edge leading from v to w, whereas earlier in this section we used (v, λ, w) as shorthand for $(\{v, w\}, \lambda)$. We hope that this ambiguity will introduce no confusion.

Let H be a directed graph. For $v, w \in V_H$, a (directed) *path* from v to w is a sequence of nodes $v_1, \ldots, v_n$ with $n \geqslant 1$ such that $v_1 = v$, $v_n = w$, and there is an edge from v_i to v_{i+1} for $1 \leqslant i \leqslant n-1$.

2. Basic Definitions and Results

In this section the definition of the type of graph grammar we consider in this paper is given. The definition coincides with the one in [EngLei1]. As discussed in the Introduction, these so-called eNCE grammars generate undirected labeled graphs, and the left-hand sides of the productions consist of a single node. They have neighbourhood controlled embedding, which means that newly generated nodes can get connected only to neighbours of the replaced node, and they use dynamic edge relabeling in the sense that edge labels can be changed during the embedding process.

As noted also in [EngLei1], these graph grammars have evolved from web grammars ([RosMil]) in the following way. First, in [JanRoz1–JanRoz4], NLC (i.e., node-label controlled) grammars were introduced. Second, in [JanRoz5], NCE grammars were defined as a generalization of NLC grammars. In NLC grammars only node labels may be used in the embedding process, whereas in NCE grammars the nodes themselves are allowed too. There are no edge labels in NLC

and NCE grammars. Edge labels, together with dynamic edge relabeling, were introduced in [JanRozVer] for NLC grammars, resulting in eNLC grammars. In this paper, as in [EngLei1], NCE and eNLC grammars are combined in an obvious way into the eNCE grammars.

In [Kau1, Kau2, Bra], (directed) graph grammars are studied which are essentially the same as the eNCE grammars of this paper. These graph grammars are also descended from web grammars, but this time via the graph grammars of [Nag].

We will consider several interesting special cases: the boundary, linear, and apex eNCE grammars. Boundary NLC grammars are introduced and investigated extensively in [RozWel1–RozWel3]. There it turns out that they have a lot of nice properties. We will show that boundary eNCE grammars can certainly compete with the boundary NLC grammars in that respect. Linear graph grammars are not very popular in the graph grammar world. Still, we believe that they are interesting since they are so simple and yet have a quite strong generating power. In [EngLei1] several results on linear eNCE grammars can be found, to which we will add some more in this paper. Apex graph grammars were introduced in [EngLeiRoz1] for directed NLC and NCE grammars. Omitting the direction and adding edge labels results in apex eNCE grammars.

The boundary grammars and the linear grammars are the main object of our attention. We will try, bit by bit, to convince the reader that boundary and linear eNCE grammars are really nice grammars to work with.

But first, we give the definition of eNCE graph grammars (cf. [EngLei1]). Note that, as opposed to [JanRoz5], all our grammars have singleton left-hand sides in their productions.

DEFINITION 1. A *graph grammar with neighbourhood controlled embedding and dynamic edge relabeling*, for short eNCE grammar, is a system $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$, where Σ is the *total alphabet of node labels*, $\Delta \subseteq \Sigma$ is the *alphabet of terminal node labels*, Γ is the *total alphabet of edge labels*, $\Omega \subseteq \Gamma$ is the *alphabet of final edge labels*, P is the finite set of *productions*, and $S \in \Sigma - \Delta$ is the *initial nonterminal*. A production $\pi \in P$ is of the form $\pi = (X, D, B)$, with $X \in \Sigma - \Delta$, $D \in GR_{\Sigma,\Gamma}$, and $B \subseteq V_D \times \Gamma \times \Gamma \times \Sigma$. B is called the *embedding relation* of π.

TERMINOLOGY. Elements from Δ are called terminals, elements from $\Sigma - \Delta$ are called nonterminals. For a graph $H \in GR_{\Sigma,\Gamma}$, a node $v \in V_H$ is called terminal if $\varphi_H(v) \in \Delta$, and nonterminal otherwise. An edge $(v, \lambda, w) \in E_H$ (recall that (v, λ, w) is standing for $(\{v, w\}, \lambda)$) is called final if $\lambda \in \Omega$, and nonfinal if $\lambda \in \Gamma - \Omega$. For a production $\pi = (X, D, B)$, X is called the left-hand side of π and D is called the right-hand side of π. We write $\mathrm{lhs}(\pi) = X$, $\mathrm{rhs}(\pi) = D$, and $B(\pi) = B$. If $D = \Lambda$, then π is called a Λ-production. If $V_D = \{v\}$ and $\varphi_D(v) \in \Sigma - \Delta$, then π is called a chain production.

We now discuss informally how a production $\pi = (X, D, B)$ is applied to a non-

terminal node v in a graph $H \in GR_{\Sigma,\Gamma}$, where $\varphi_H(v) = X$. First, v is removed from H, together with all edges incident with v. Next, D is added to the remainder of H, in place of v. Finally, D is embedded in the remainder of H by adding edges between nodes of D and nodes of $H - \{v\}$ as follows. If $x \in V_D$ and $y \in V_H - \{v\}$, then an edge is added between x and y labeled μ if and only if there was an edge between v and y labeled λ in H, and $(x, \lambda, \mu, \varphi_H(y))$ is in B. Thus, x inherits some of the edges that connect v to its neighbours, with possibly a different label. Formally, all of this is defined as follows.

DEFINITION 2. Let $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ be an eNCE grammar. Let H and K be graphs over Σ and Γ, let $v \in V_H$, and let $\pi = (X, D, B) \in P$. We may assume that $V_H \cap V_D = \varnothing$ (otherwise, replace H by an isomorphic copy). Then we write $H \Rightarrow_{(v,\pi)} K$, or just $H \Rightarrow K$, if $\varphi_H(v) = X$ and K is (isomorphic to) the following graph:

$$
\begin{aligned}
&V_K = (V_H - \{v\}) \cup V_D,\\
&E_K = \{(x, \mu, y) \in E_H \mid x \neq v,\ y \neq v\}\\
&\qquad \cup E_D\\
&\qquad \cup \{(x, \mu, y) \mid x \in V_D,\ y \in V_H - \{v\},\ \mu \in \Gamma,\ \exists \lambda \in \Gamma\colon (v, \lambda, y) \in E_H,\\
&\qquad\qquad (x, \lambda, \mu, \varphi_H(y)) \in B\},\\
&\Sigma_K = \Sigma,\\
&\Gamma_K = \Gamma,\\
&\varphi_K(x) = \varphi_H(x) \text{ if } x \in V_H - \{v\}, \text{ and } \varphi_K(x) = \varphi_D(x) \text{ if } x \in V_D.
\end{aligned}
$$

$H \Rightarrow K$ is called a *derivation step*, and a sequence of such derivation steps is called a *derivation*. As usual, $\overset{*}{\Rightarrow}$ is the transitive–reflexive closure of $\Rightarrow$. A graph $H \in GR_{\Sigma,\Gamma}$ such that $S \overset{*}{\Rightarrow} H$ is called a *sentential form* of G. $S(G)$ denotes the set of all sentential forms of G. The *language generated by* G is $L(G) = \{H \in GR_{\Delta,\Omega} \mid S \overset{*}{\Rightarrow} H\}$.

The class of all languages generated by eNCE grammars is denoted eNCE. Two examples are now given of eNCE grammars. In the first, a language is generated that is taken from [JanRozVer]. As in that paper (in [JanRozVer, Theorem 5] it is shown that this language is in eNLC, but not in NLC), the example demonstrates the use of (dynamically changing) edge labels. In this example we also discuss an elegant graphical specification of the productions, introduced in [Kau1], which we will use in most of our examples. The second example is taken from [Jan], where it is shown that the language that is generated is an NCE language. Hence, this example is not typically eNCE.

EXAMPLE 3. Consider the eNCE grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ with $\Sigma = \{S, a\}$, $\Delta = \{a\}$, $\Gamma = \{\lambda, *\}$, $\Omega = \{*\}$, $P = \{\pi_1, \pi_2\}$, where $\pi_1 = (S, D, B)$ and

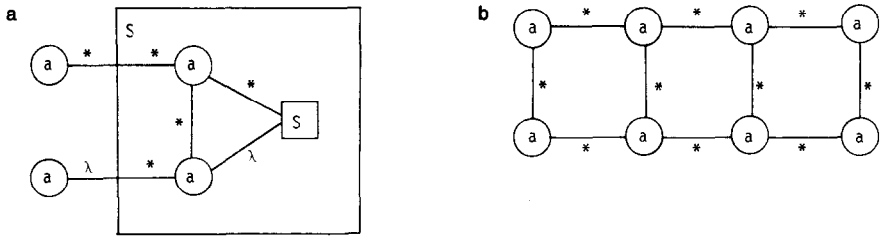

FIGURE 1

$\pi_2 = (S, A, \varnothing)$. The right-hand side of π_1 is defined as follows: $V_D = \{x, y, z\}$, $E_D = \{(x, *, y), (x, *, z), (y, \lambda, z)\}$, $\Sigma_D = \Sigma$, $\Gamma_D = \Gamma$, $\varphi_D(x) = \varphi_D(y) = a$, and $\varphi_D(z) = S$. Furthermore, $B = \{(x, *, *, a), (y, \lambda, *, a)\}$. In Fig. 1a the graphical specification of π_1 is given (since the one for π_2 is trivial, we do not draw it). Terminal nodes are circles, nonterminal nodes are boxes. In the upper left corner of the big box (representing the node that is to be rewritten) the left-hand side of the production is drawn, and the graph inside the box is the right-hand side of π_1. The edges that connect nodes inside the box with nodes outside the box represent the embedding relation. They have two labels: the one outside the box is the "old" label, and the one inside the box is the "new" label. The fourth component of a tuple in the embedding relation of π_1 is placed as label on a node outside the big box. It is now easy to see that $L(G)$ consists of all "ladders" of the form given in Fig.1b, of arbitrary length.

EXAMPLE 4. Consider the eNCE grammar $G = (\{A, B, a\}, \{a\}, \{*\}, \{*\}, P, A)$, where P consists of the four productions presented in Fig. 2. G generates the set of all graphs "without edge labels" and with node label a. In fact, G first applies production π_1 any number of times, resulting in a complete graph with A-labeled nodes only (note that the embedding relation of π_1 contains two tuples). Next, an edge between two nodes can be removed by applying first π_2 to both nodes, and then π_3. In this way, any edge can be removed from the graph. Finally π_4 is used to generate a terminal graph.

Notice that, as mentioned in the Introduction, the order of rewriting is important in eNCE grammars, just as in NLC and NCE grammars. If we first apply, for example, production π_4 of Example 4 to the graph in Fig. 3a, and next π_3, then the graph in Fig. 3b is obtained. If, on the other hand, we apply first π_3 and next π_4 to the same nodes in Fig. 3a, then the graph in Fig. 3c results. Hence, the order of rewriting influences the result, and so eNCE grammars are not confluent (cf. [Cou]), a severe disadvantage of eNCE grammars. In NLC grammars, where the same situation can occur, one way of "solving" this problem is to restrict attention to the subclass of boundary NLC grammars [RozWel1–RozWel3]. In these grammars, nonterminal nodes are never connected by an edge. Therefore, the application

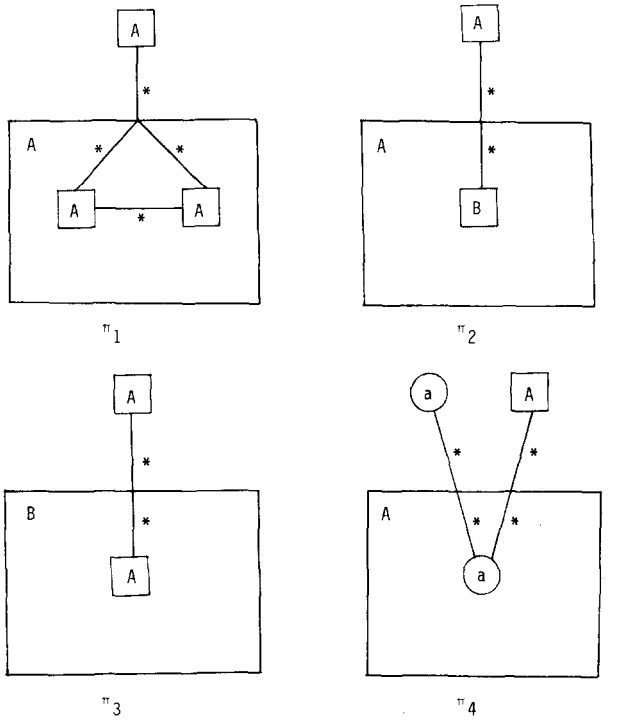

FIGURE 2

of a production to a nonterminal node has no influence on the neighbourhood of other nonterminal nodes. From this it can easily be deduced (cf. [RozWell, Lemma 2.3]) that boundary grammars are confluent. In fact, this is a crucial property of boundary grammars, which enables us to show a number of results for boundary grammars which cannot be derived for arbitrary eNCE grammars.

Boundary eNCE grammars are defined next, together with linear and apex grammars.

DEFINITION 5. Let $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ be an eNCE grammar.

(i) G is a *boundary eNCE* grammar, for short B-eNCE gramar, if, for every production $(X, D, B) \in P$, D does not contain edges between nonterminal nodes, i.e., if $(v, \lambda, w) \in E_D$, then $\varphi_D(v) \in \Delta$ or $\varphi_D(w) \in \Delta$.

FIGURE 3

(ii) G is a *linear eNCE grammar*, for short LIN-eNCE grammar, if for all $\pi \in P$, rhs(π) contains at most one nonterminal node.

(iii) G is an *apex eNCE grammar*, for short A-eNCE grammar, if for every production $\pi = (X, D, B)$ the embedding relation B is a subset of $\{(x, \lambda, \mu, a) \in V_D \times \Gamma \times \Gamma \times \Sigma \mid \varphi_D(x) \in \Delta \text{ and } a \in \Delta\}$.

For $X \in \{\text{B, LIN, A}\}$, the class of all languages that can be generated by an X-eNCE grammar is denoted X-eNCE.

Note that there are no edges between nonterminal nodes in all the sentential forms of B-eNCE grammars. This property will be used several times in the paper.

The eNCE grammar of Example 3 is clearly linear and apex. Another linear grammar is given in the following example. Example 7 gives an apex grammar generating the set of all binary trees. All these grammars are clearly boundary, whereas the grammar of Example 4 is not.

EXAMPLE 6. Let $G = (\{S, a\}, \{a\}, \{\lambda, *\}, \{*\}, P, S)$ be an eNCE grammar, where P is defined in Fig. 4. $L(G)$ contains the graphs $H(n) \in GR_{\{a\},\{*\}}$ with $V_{H(n)} = \{x_1, ..., x_n\}$ and $E_{H(n)} = \{(x_i, *, x_j) \mid 1 \leqslant i, j \leqslant n: |j-i| \geqslant 2\}$, for all $n \geqslant 0$. This example also illustrates the use of edge labels; we will show later that $L(G)$ is not in NCE. Clearly, G is a LIN-eNCE grammar.

EXAMPLE 7. Consider the A-eNCE grammar $G = (\{S, a\}, \{a\}, \{*\}, \{*\}, P, S)$ with P defined in Fig. 5. $L(G)$ is the set of all binary trees.

In Section 6 eNCE grammars will be compared to NCE graph grammars. Since these generate graphs without edge labels, we now define eNCE grammars which only generate such graphs. The reserved edge label * will be used to indicate the absence of an edge label (cf. Section 1).

DEFINITION 8. Let $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ be an eNCE grammar.

(i) G is a *one-final* eNCE, for short e_1NCE, grammar if $\Omega = \{*\}$.

(ii) G is an NCE grammar if $\Gamma = \Omega = \{*\}$.

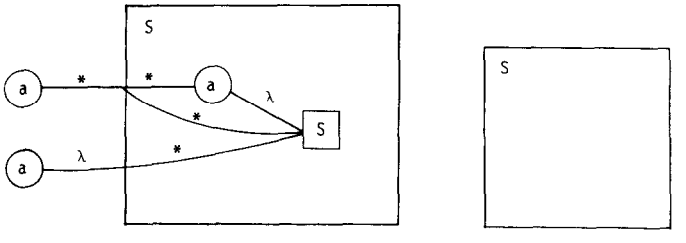

FIGURE 4

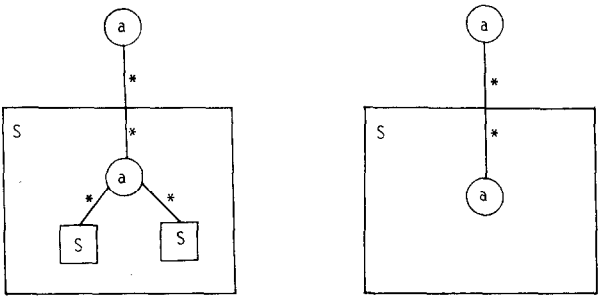

FIGURE 5

We use e_1NCE and NCE to denote the classes of languages that can be generated by the corresponding grammars. Moreover, it should be clear which classes we denote by the abbreviations LIN-e_1NCE, B-NCE, etc.

Note that e_1NCE just consists of all eNCE languages that contain graphs without edge labels only. Note also that the definition of NCE grammars coincides with the one from [JanRoz5]. Hence, as shown there, NCE = NLC: the class of NLC languages investigated in [JanRoz1–JanRoz4]. Note finally that the grammars of Example 4 and Example 7 are NCE grammars, whereas the grammars in Example 3 and Example 6 are e_1NCE.

Now we turn to some basic results on eNCE grammars. From the definitions it follows that LIN-eNCE $\subseteq$ B-eNCE. It can furthermore easily be shown that A-eNCE $\subseteq$ B-eNCE: just remove the edges between nonterminal nodes in the right-hand sides of the productions of an A-eNCE grammar. Both inclusions are proper. First, in [EngLeiRoz1] it is shown that apex languages are always of bounded degree, whereas the linear language of Example 6 is not. Second, in [EngLei1, Theorem 16] it is shown that the set of binary trees of Example 7 cannot be generated by a linear grammar. These two counterexamples show that LIN-eNCE and A-eNCE are incomparable subclasses of B-eNCE. In order to prove the correctness of the inclusion diagram in Fig. 6 (except for N-eNCE, a class of

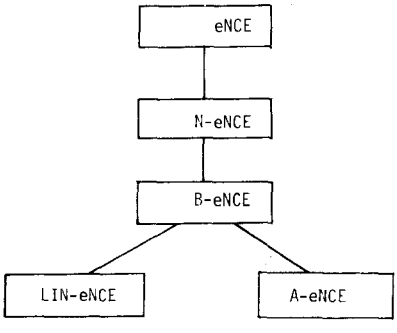

FIGURE 6

languages introduced in the next section), it suffices to show that there is an eNCE language that is not in B-eNCE. Later on in this section we will see that the language of Example 4 forms such a language.

Since the result is needed in the rest of this paper, we now prove that eNCE is closed under node relabeling (the same can easily be achieved for edge relabeling, but is not needed here). This is the first example of the advantage of grammars with dynamic edge relabeling over similar models: the class of NCE languages is not closed under node relabeling ([RozWel2, Theorem 4.1], see also [EngLeiRoz1, JanRozVer]).

THEOREM 9. *eNCE is closed under node relabeling.*

Proof. Let $G=(\Sigma, \Delta, \Gamma, \Omega, P, S)$ be an eNCE grammar, and let $\rho: \Delta \to \bar{\Delta}$ be a node relabeling. It may be assumed that $(\Sigma-\Delta)\cap\bar{\Delta}=\varnothing$. We will construct an eNCE grammar $\bar{G}=(\bar{\Sigma}, \bar{\Delta}, \bar{\Gamma}, \Omega, \bar{P}, S)$ such that $L(\bar{G})=\rho(L(G))$ as follows. First, let $\bar{\Sigma}=(\Sigma-\Delta)\cup\bar{\Delta}$, and let $\bar{\Gamma}=\Gamma\cup\Gamma\times\Delta$. Second, in order to define the productions in $\bar{P}$, we define the function $\sigma: \Sigma\times\Gamma\times\Sigma\to\bar{\Gamma}$ by $\sigma(a, \lambda, b)=\lambda$ if $a, b\in\Delta$ or $a, b\in\Sigma-\Delta$, and $\sigma(a, \lambda, b)=\sigma(b, \lambda, a)=\langle\lambda, b\rangle$ if $a\in\Sigma-\Delta$ and $b\in\Delta$. We extend ρ to Σ by defining $\rho(a)=a$ for each $a\in\Sigma-\Delta$. If now P contains the production (X, D, B), then $\bar{P}$ contains the production $(X, \bar{D}, \bar{B})$, where $V_{\bar{D}}=V_D$, $E_{\bar{D}}=\{(v, \sigma(\varphi_D(v), \lambda, \varphi_D(w)), w) \mid (v, \lambda, w)\in E_D\}$, and $\varphi_{\bar{D}}(v)=\rho(\varphi_D(v))$. Furthermore, $\bar{B}=\{(v, \sigma(X, \lambda, a), \sigma(\varphi_D(v), \mu, a), \rho(a)) \mid (v, \lambda, \mu, a)\in B\}$.

$\bar{G}$ has almost the same productions as G, only the labels differ. Instead of terminal label $a\in\Delta$, $\rho(a)$ is placed on the nodes in the productions of $\bar{G}$. To keep track of the original node label a, $\bar{G}$ places this a on all the edges incident with this terminal node and a nonterminal node. The labels of the rest of the edges are not changed. Now, it will not be difficult to see that $L(\bar{G})=\rho(L(G))$. ∎

It is easily seen that this theorem still holds in the boundary, the apex, and the linear case. (It also holds in the "nonblocking" case, to be discussed in the next section.)

The second result we prove in this section concerns chain and Λ-productions (see the terminology following Definition 1). The result tells us that B-eNCE grammars can do without chain and Λ-productions. In [RozWel1] the same has been proved for B-NLC grammars, but there the proof had to be lengthy. For B-eNCE grammars the proof is quite simple.

THEOREM 10. *Let $G=(\Sigma, \Delta, \Gamma, \Omega, P, S)$ be a B-eNCE grammar. Then there exists a B-eNCE grammar $\bar{G}=(\Sigma, \Delta, \Gamma, \Omega, \bar{P}, S)$ without chain or Λ-productions such that $L(\bar{G})=L(G)$.*

Proof. The proof for Λ-productions is fully analogous to the proof for context-free string grammars (see, e.g., [Sal, Part I, Theorem 6.2]). So assume that G has no Λ-productions. The proof for chain productions is also the same as the one for context-free string grammars, except that in addition we have to compute the

correct embedding relations. This can be done as follows. If $(X_1, D_1, B_1), \ldots, (X_n, D_n, B_n)$ are chain productions of P, with $n \geqslant 0$, and if $(X_{n+1}, D_{n+1}, B_{n+1})$ is not a chain production of P, such that X_{i+1} is the label of the node in D_i for $1 \leqslant i \leqslant n$, then $\bar{P}$ contains the production (X_1, D_{n+1}, B) with $B = \{(x_{n+1}, \lambda, \mu, a) \mid x_{n+1} \in V_{D_{n+1}}$, for every $0 \leqslant i \leqslant n+1$ there is a $\lambda_i \in \Gamma$ such that for all $1 \leqslant i \leqslant n+1$, $(x_i, \lambda_{i-1}, \lambda_i, a) \in B_i$ and $\lambda = \lambda_0$, $\mu = \lambda_{n+1}\}$, where x_i denotes the node of D_i $(1 \leqslant i \leqslant n)$. These are all the productions that $\bar{G}$ contains. It is not difficult to see that $L(\bar{G}) = L(G)$, that $\bar{G}$ contains no chain productions, and that no new Λ-productions are introduced. The construction used is not constructive, but, as in the string case, it can easily be made so. ∎

This result also holds for linear and apex eNCE grammars. The result on Λ-productions even holds for arbitrary eNCE grammars. Hence, it may be assumed that the number of nodes in the sentential forms of a derivation of an eNCE grammar never decreases (though the number of edges may decrease). From this it can now easily be deduced that eNCE is in NSPACE(n^2), where n is the number of nodes of the considered graph.

The construction in the proof above works because the order in which productions are applied to a sentential form does not influence the result (see the discussion on confluence just before Definition 5). Therefore, a number of consecutive chain productions may be simulated in one step. In arbitrary eNCE grammars, to the contrary, we may not simulate a number of chain productions applied to a nonterminal node x in one big step, since a production applied in-between to a nonterminal neighbour of x might influence the result. In fact, by the general result in [Wel1], eNCE grammars do need chain productions to generate certain eNCE languages. The set of all graphs without edge labels over $\{a\}$ is an example of such a language (see Example 4). Thus, there exists no boundary grammar generating this language, and hence B-eNCE is a proper subclass of eNCE (see Fig. 6).

3. Final and Nonfinal Edge Labels

In this section we take a closer look at the definition of eNCE grammars. Since it differs from the one of NCE grammars [JanRoz5] only with respect to the edge labels, we will mainly be concerned with these edge labels. In particular, an answer is given to the question "why do we call some of the edge labels final, and the others nonfinal?" At first sight, these two names do not seem to be appropriate: final edges can still be changed in the embedding process (so they may not be "really final"), and nonfinal edges can be incident with two terminal nodes (so they can, in fact, be "final"). We would like to consider only graph grammars in which final and nonfinal edges do satisfy their intuitive "meaning":

(i) final edges are ready, i.e., the two nodes incident with such an edge are both terminal.

(ii) nonfinal edges are not yet ready, i.e., at least one of the nodes incident with such an edge is nonterminal.

In this section we investigate the power of grammars that satisfy these two demands. It will turn out that eNCE languages can always be generated by grammars that satisfy the first demand, but not always by grammars satisfying the second. Therefore, a new subclass of the eNCE languages is obtained, generated by all eNCE grammars satisfying the second demand (and the first). The situation for boundary (linear, apex) languages is quite different: we show that they can always be generated by a boundary (linear or apex, respectively) grammar satisfying both demands. This shows again the usefulness of the boundary restriction.

We first define normalized eNCE grammars, i.e., eNCE grammars that satisfy the first demand.

DEFINITION 11. Let $G=(\Sigma, \Delta, \Gamma, \Omega, P, S)$ be an eNCE grammar. G is *normalized* if, for all H such that $S \overset{*}{\Rightarrow} H$ and all $(x, \lambda, y) \in E_H$, if $\lambda \in \Omega$, then $\varphi_H(x) \in \Delta$ and $\varphi_H(y) \in \Delta$.

We now show that each eNCE language can be generated by a normalized eNCE grammar.

THEOREM 12. *For every $L \in eNCE$ there is a normalized eNCE grammar G such that $L = L(G)$.*

Proof. Let $G=(\Sigma, \Delta, \Gamma, \Omega, P, S)$ be an eNCE grammar. We will construct a normalized eNCE grammar $\bar{G}=(\Sigma, \Delta, \bar{\Gamma}, \Omega, \bar{P}, S)$ with $L(\bar{G})=L(G)$. The obvious trick is to use for each final edge label $\lambda \in \Omega$ a new nonfinal edge label $\bar{\lambda}$. Whenever G places λ on an edge incident with a nonterminal node, then $\bar{G}$ will place $\bar{\lambda}$ on that edge instead. Formally, $\bar{G}$ is defined as follows.

First, let $\bar{\Omega}=\{\bar{\lambda} \mid \lambda \in \Omega\}$ be such that $\bar{\Omega} \cap \Gamma = \varnothing$, and let $\bar{\Gamma}=\Gamma \cup \bar{\Omega}$. Second, define $\sigma: \Gamma \to \bar{\Gamma} - \Omega$ by $\sigma(\lambda)=\bar{\lambda}$ if $\lambda \in \Omega$, and $\sigma(\lambda)=\lambda$ if $\lambda \in \Gamma - \Omega$. If now (X, D, B) is a production in P, then $\bar{P}$ contains the production $(X, \bar{D}, \bar{B})$, where $\bar{D} \in GR_{\Sigma, \Gamma}$ and $\bar{B}$ are defined as follows: $V_{\bar{D}}=V_D$, $E_{\bar{D}}=\{(x, \lambda, y) \in E_D \mid \varphi_D(x), \varphi_D(y) \in \Delta\} \cup \{(x, \sigma(\lambda), y) \mid (x, \lambda, y) \in E_D, \varphi_D(x) \notin \Delta$ or $\varphi_D(y) \notin \Delta\}$, $\varphi_{\bar{D}}=\varphi_D$, and $\bar{B}=\{(x, \sigma(\lambda), \mu, a) \mid (x, \lambda, \mu, a) \in B, \varphi_D(x) \in \Delta, a \in \Delta\} \cup \{(x, \sigma(\lambda), \sigma(\mu), a) \mid (x, \lambda, \mu, a) \in B, \varphi_D(x) \notin \Delta$ or $a \notin \Delta\}$. It is not difficult to check that indeed $\bar{G}$ is normalized, and that $L(\bar{G})=L(G)$. ∎

The construction used in the theorem above clearly preserves the boundary, the linear, and the apex property. Hence, for each boundary (linear, apex) eNCE language, there is a normalized boundary (linear, apex, respectively) eNCE grammar.

Note that, according to the construction used above, we may assume that, for each production π of a normalized eNCE grammar, if $(x, \lambda, \mu, a) \in B(\pi)$ then $\lambda \in \Gamma - \Omega$, and if $\mu \in \Omega$ then $\varphi_{\mathrm{rhs}(\pi)}(x) \in \Delta$ and $a \in \Delta$. Thus the embedding relations of the productions "know" that the grammar is normalized.

By Theorem 12, we may always assume that the sentential forms of eNCE grammars have no final edge labels on edges incident with a nonterminal node (in other words, edges that will be "rewritten" are always nonfinal). But what about the labels of the edges incident with two terminal nodes? Unfortunately it is not possible to take care that these labels are always final (i.e., nonfinal edges are "really nonfinal"). There are eNCE languages that can only be generated by graph grammars that have sentential forms with nonfinal labels on some of the edges between terminal nodes. These edges are called *blocking edges* (see [Nag, p. 38]), because they prevent the grammar from accepting any graph that can be derived from such a sentential form. So blocking edges can be used to filter out of the language some of the graphs with terminal nodes only (see [JanRoz2, Theorem 9] for a closely related filtering mechanism). The next example is meant to give a better insight in the power of these blocking edges. (We also observe that several results in Chapter 1 of [Nag] rely on the use of blocking edges.)

EXAMPLE 13. Consider the (normalized) eNCE grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ with $\Sigma = \{S, A, C, a, b\}$, $\Delta = \{a, b\}$, $\Gamma = \{\lambda, \mu, *, \$\}$, $\Omega = \{*\}$, and with P containing the following five productions:

$\pi_0 = (S, D, \varnothing)$, with $V_D = \{x, y\}$, $E_D = \{(x, \mu, y)\}$, $\varphi_D(x) = C$, and $\varphi_D(y) = A$.

$\pi_1 = (C, D, B)$, with $V_D = \{x\}$, $E_D = \varnothing$, $\varphi_D(x) = C$, and $B = \{(x, \mu, \lambda, A)\} \cup \{(x, p, \$, q) \mid p \in \Gamma, q \in \Sigma, p \neq \mu \text{ or } q \neq A\}$.

$\pi_2 = (C, D, B)$, as for π_1, except that $\varphi_D(x) = b$.

$\pi_3 = (A, D, B)$, with $V_D = \{x, y\}$, $E_D = \varnothing$, $\varphi_D(x) = \varphi_D(y) = A$, and $B = \{(w, \lambda, \mu, C) \mid w \in \{x, y\}\} \cup \{(w, p, \$, q) \mid w \in \{x, y\}, p \in \Gamma, q \in \Sigma, p \neq \lambda \text{ or } q \neq C\}$.

$\pi_4 = (A, D, B)$, with $V_D = \{x\}$, $E_D = \varnothing$, $\varphi_D(x) = a$, and $B = \{(x, \lambda, *, b)\} \cup \{(x, p, \$, q) \mid p \in \Gamma, q \in \Sigma, p \neq \lambda \text{ or } q \neq b\}$.

This grammar generates the language of all "star"-graphs of the form depicted in Fig. 7a, with a b-node in the middle, and with 2^n a-nodes attached to this b-node, for some $n \geq 0$. This can be understood from the following arguments:

(i) Whenever a \$-edge is generated by G, then this edge will not disappear anymore, so eventually it will lead to a blocking edge. Hence, a sentential form with

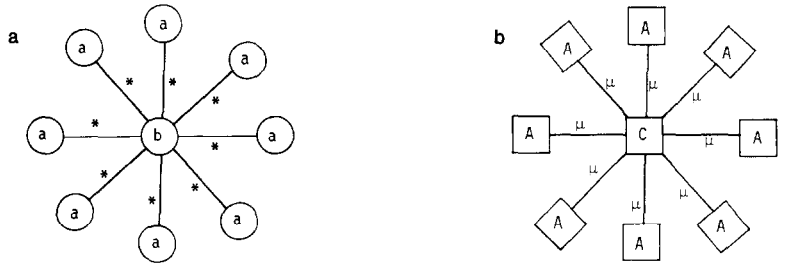

FIGURE 7

a \$-edge cannot lead to a graph in the language. So we only have to consider the derivations that do not introduce these edges.

(ii) It is easily seen that A-nodes cannot be replaced whenever they are incident with a μ-edge (this means: if such a node is replaced, then a \$-edge is introduced). Moreover, when an A-node is connected to a C-node by a λ-edge, then only π_3 may be used (to replace it by two A-nodes, connected to the C-node by μ-edges). When it is connected to a b-node by a λ-edge, then only π_4 may be used (to replace it by an a-node).

(iii) Similarly, a C-node cannot be replaced whenever it is incident with a λ-edge. When it is connected to all A-nodes by a μ-edge, then both π_1 and π_2 may be used (replacing it by the same C-node or by a b-node, respectively, and changing all μ's into λ's).

We will now use an induction argument to show that $L(G)$ is indeed the set of graphs described above. We state that, after $2^n + n$ derivation steps (with $n \geqslant 0$), the derived sentential form either is the graph of Fig. 7a with 2^{n-1} a-nodes, or is the graph of Fig. 7b, with 2^n A-nodes (or is a graph with a \$-edge, but we do not consider these graphs, see (i)). For $n = 0$, this is trivially true since π_0 is the only production we can start with. Assume now that it is true for n. To obtain a derivation with $2^{n+1} + n + 1$ steps, we have to proceed with the graph of Fig. 7b. So, as the first step of the continuation of the derivation, either π_1 or π_2 has to be applied to the C-node (this follows from (ii)). First, when π_2 is used, then all A-nodes have to be replaced by production π_4 in the next 2^n steps (see again (ii)). Hence, we get the graph of Fig. 7a with 2^n a-nodes. Second, when π_1 is used to replace the C-node (all μ's now become λ's), then all A-nodes have to be replaced by production π_3 in the next 2^n steps (see (ii) and (iii); each A-node is replaced by two A-nodes, and the λ's become μ's again), so we get the graph of Fig. 7b with $2.2^n = 2^{n+1}$ A-nodes.

From the example above it can be seen that blocking edges can be used to prevent a production from being applied to a node in a sentential form, by the introduction of a nonfinal edge that never disappears. In this way, blocking edges work like application conditions. This is a kind of context-sensitivity we want to avoid. Therefore we define graph grammars that cannot generate blocking edges, and thus are much less powerful than eNCE grammars. In particular, the language of Example 13 cannot be generated by them.

DEFINITION 14. An eNCE grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ is *nonblocking* if, for all H such that $S \overset{*}{\Rightarrow} H$ and all $(x, \lambda, y) \in E_H$, if $\varphi_H(x) \in \Delta$ and $\varphi_H(y) \in \Delta$, then $\lambda \in \Omega$.

The class of all languages generated by a nonblocking eNCE grammar is denoted N-eNCE. In [EngLeiRoz2], only nonblocking grammars are used (these grammars were just called edNCE grammars, where the "d" means that directed graphs are generated, see also Section 8 of [EngLeiWel]). Note that, trivially, NCE $\subseteq$ N-eNCE. Note also that N-eNCE is closed under node relabeling (check the proof of Theorem 9).

If we reconsider the construction used to show that each eNCE language can be generated by a normalized eNCE grammar (Theorem 12), then we see that it preserves the nonblocking property. Hence we may assume that each nonblocking grammar is normalized, and so we might (and will) say that the first N inN-eNCE is standing for "nonblocking" as well as for "normalized." This means that N-eNCE grammars form the subclass of eNCE grammars that satisfy demands (i) and (ii) mentioned at the beginning of this section.

It is easily seen that we can take care that the productions of an N-eNCE grammar $G=(\Sigma, \Delta, \Gamma, \Omega, P, S)$ "know" that the grammar is nonblocking and normalized, by requiring that each production π is such that

(a) whenever $(x, \lambda, \mu, a) \in B(\pi)$, then $\lambda \in \Gamma - \Omega$, and , moreover, $\mu \in \Omega$ iff both $\varphi_{\mathrm{rhs}(\pi)}(x) \in \Delta$ and $a \in \Delta$,

(b) if $(x, \mu, y) \in E_{\mathrm{rhs}(\pi)}$ then $\mu \in \Omega$ iff both $\varphi_{\mathrm{rhs}(\pi)}(x) \in \Delta$ and $\varphi_{\mathrm{rhs}(\pi)}(y) \in \Delta$.

Thus, requiring (a) and (b) gives us a static definition of an N-eNCE grammar. This definition will be used too, instead of Definition 14 (and Definition 11).

To show that N-eNCE grammars are less powerful than ordinary eNCE grammars, we need the following concept.

DEFINITION 15. Let $G=(\Sigma, \Delta, \Gamma, \Omega, P, S)$ be an eNCE grammar. Then the *context-free string grammar associated with* G is $\mathrm{CF}(G)=((\Sigma-\Delta)\cup\{a\}, \{a\}, \bar{P}, S)$, where $\bar{P}$ consists of all productions $X \to w$ with $w \in ((\Sigma-\Delta)\cup\{a\})^*$, such that there exists a $(X, D, B) \in P$ that satisfies $\#_a(w) = \#\{x \in V_D \mid \varphi_D(x) \in \Delta\}$, and, for all $Y \in \Sigma - \Delta$, $\#_Y(w) = \#\{x \in V_D \mid \varphi_D(x) = Y\}$.

$\mathrm{CF}(G)$ simulates G in a certain sense. Whenever a production π of G generates i terminal nodes, then a corresponding production π' of $\mathrm{CF}(G)$ generates i a's. Moreover, for each nonterminal node generated by π, π' generates the label of this node. The order in which π' generates these symbols is not important. Now the following lemma (which is similar to Lemma 3.3. of [Jef]) is easy to understand and prove.

LEMMA 16. *For every N-eNCE grammar* G, $L(\mathrm{CF}(G)) = \{a^n \mid n = \# V_H$ *for some* $H \in L(G)\}$.

Note that we have explicitly stated this lemma only for N-eNCE grammars. Since $\mathrm{CF}(G)$ only simulates an eNCE grammar G with respect to the node labels, it does not notice the blocking edges. Thus $L(\mathrm{CF}(G)) = \{a^n \mid n = \# V_H$ for some $H \in S(G)$ that contains no nonterminal nodes$\}$, but the languages of $\mathrm{CF}(G)$ and G may not be related to each other.

Now we are ready to show that N-eNCE grammars are less powerful than eNCE grammars.

THEOREM 17. *N-eNCE is properly included in eNCE.*

Proof. Reconsider the eNCE grammar G of Example 13, and assume that there is an N-eNCE grammar $\bar{G}$ such that $L(\bar{G}) = L(G)$. Then the context-free grammar $\mathrm{CF}(\bar{G})$ generates the language $\{a^k \mid k = 2^n + 1,\ n \geq 0\}$, according to Lemma 16. But this is not a context-free language. ∎

We have just seen that some eNCE languages cannot be generated without using blocking edges. However, as shown next, this does not hold for boundary grammars: all B-eNCE languages can be generated by a nonblocking B-eNCE grammar (the next nice property of boundary grammars!). This is based on the fact that the context of a node and the embedding relation of a production together decide whether a blocking edge will be introduced when applying that production to the node. So, if we take care that productions can only be applied to a node if the context of that node "permits" it, we can get rid of blocking edges (see Section 1 for the formal definition of context).

In so-called context consistent graph grammars (see [RozWel1]), the context of a node is stored in its label, i.e., if x is a nonterminal node in a sentential form H, and X is its label, then $\mathrm{context}_H(x) = \eta(X)$, for some fixed function η. Since the neighbours of a nonterminal node in a B-eNCE grammar cannot be rewritten, it can easily be proved that there is a context consistent B-eNCE grammar for each B-eNCE language (just add the context to the nonterminal labels, see [RozWel1, Theorem 3.2]). For such a grammar, the left-hand side of a production and the embedding relation of this production together determine whether a blocking edge will be generated, and so the grammar can be made nonblocking edge will be generated, and so the grammar can be made nonblocking by just removing such productions. All this is now stated more formally.

DEFINITION 18. An eNCE grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ is *context consistent* if there is a function $\eta: \Sigma - \Delta \to \mathcal{P}(\Sigma \times \Gamma)$ such that for every H with $S \overset{*}{\Rightarrow} H$ and for every $x \in V_H$ with $\varphi_H(x) \in \Sigma - \Delta$, $\mathrm{context}_H(x) = \eta(\varphi_H(x))$. The function η satisfying the above is called the *context describing function* of G.

LEMMA 19. *For every B-eNCE language L there is a context consistent B-eNCE grammar G such that $L = L(G)$.*

Proof. Fully analogous to the ones in [RozWel1, EngRoz]. ∎

THEOREM 20. *For every $L \in$ B-eNCE there is a nonblocking B-eNCE grammar G such that $L = L(G)$.*

Proof. Let $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ be a context consistent B-eNCE grammar, cf. Lemma 19, and let $\eta: \Sigma - \Delta \to \mathcal{P}(\Delta \times \Gamma)$ be the context describing function of G. Now we construct a nonblocking B-eNCE grammar $\bar{G} = (\Sigma, \Delta, \Gamma, \Omega, \bar{P}, S)$ with $L(\bar{G}) = L(G)$ as follows. If $\pi = (X, D, B)$ is a production in P with the properties

(i) if $(x, \lambda, y) \in E_D$ with $\varphi_D(x) \in \Delta$ and $\varphi_D(y) \in \Delta$, then $\lambda \in \Omega$, and

(ii) if $(x, \lambda, \mu, a) \in B$ with $a \in \Delta$, $\varphi_D(x) \in \Delta$, and $(a, \lambda) \in \eta(X)$, then $\mu \in \Omega$,

then π is a production in $\bar{P}$ too. ∎

Three remarks can be made now. First, Lemma 19 and Theorem 20 also hold for linear and apex eNCE grammars. Thus, we may always assume that linear and apex grammars are nonblocking. Second, by the discussion following Definition 14, B-eNCE grammars may be assumed to be nonblocking as well as normalized at the same time (and similarly for LIN-eNCE and A-eNCE). Third, the theorem above proves that B-eNCE $\subseteq$ N-eNCE. This inclusion is proper: as mentioned at the end of Section 2, the graph language of Example 4, which clearly is in N-eNCE (even in NCE), cannot be generated by a boundary eNCE grammar. This proves the correctness of the inclusion diagram in Fig. 6.

Finally, we would like to mention that we prefer N-eNCE grammars to eNCE grammars, because they have less context-sensitive aspects, and because final (nonfinal) edges are "really final" ("really nonfinal," respectively). We also prefer B-eNCE grammars to N-eNCE grammars because they have even less context-sensitive aspects (due to confluence).

4. Chomsky and Greibach Normal Form

In formal language theory one of the first topics of research has been the search for normal forms for context-free string grammars. The two most famous results in this direction are undoubtedly the Chomsky and the Greibach normal form. A grammar in Chomsky NF, which only has productions of the form $A \to BC$ and $A \to b$ (A, B, C nonterminals, b terminal), has the big advantage over ordinary context-free grammars that the productions are bounded in length. A grammar in Greibach NF, which only has productions of the form $A \to bB_1 \cdots B_n$ ($n \geqslant 0$, $A, B_1, ..., B_n$ nonterminals, b terminal) is often to prefer over ordinary context-free grammars since the length of a derivation of a string is known in advance.

It is thus not surprising that one has searched for Chomsky-like and Greibach-like normal forms in the graph grammar world, unfortunately often without success. The only positive result we know of can be found in [Nag (see Theorem I.3.21)], where it is shown that each context-free graph language (in the sense of [Nag]) can be generated by a graph grammar with at most two nodes in the right-hand sides of the productions. Negative results can be found in [Wel1], where it is shown that some graph languages (which are in NLC, (N-)eNCE, etc.) cannot be generated without using chain productions. This clearly implies that there exists no Greibach-like normal form for NLC and (N-)eNCE grammars. In [EhrMaiRoz], furthermore, it is shown that there does not exist a Chomsky-like normal form for NLC grammars. Finally, a Chomsky normal form neither exists for boundary NLC grammars ([RozWel1, Corollary 5.5]), nor for edge replacement systems ([HabKre, Corollary 5.5]).

In this section it is demonstrated that these two normal form results can be obtained for B-eNCE grammars. First, we define Chomsky and Greibach normal form for our graph grammars.

Definition 21. Let $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ be an eNCE grammar.

(i) G is in *Chomsky normal form* if each $\pi \in P$ is of one of the three forms: (1) $\text{rhs}(\pi)$ consists of one node, say x, with $\varphi_{\text{rhs}(\pi)}(x) \in \Delta$. In this case, π is called a type-1 production. (2) $\text{rhs}(\pi)$ consists of two nodes, say x and y, with $\varphi_{\text{rhs}(\pi)}(x) \in \Delta$ and $\varphi_{\text{rhs}(\pi)}(y) \in \Sigma - \Delta$. π is called a type-2 production. (3) Like case (2), but now with $\varphi_{\text{rhs}(\pi)}(x) \in \Sigma - \Delta$, too. π is now called a type-3 production.

(ii) G in *Greibach normal form* if for every $\pi \in P$ $\#\{x \in V_{\text{rhs}(\pi)} \mid \varphi_{\text{rhs}(\pi)}(x) \in \Delta\} = 1$, and $1 \leqslant \# V_{\text{rhs}(\pi)} \leqslant 3$.

The definition for Greibach NF is the obvious one, while the definition for Chomsky NF we have chosen seems at first sight a bit strange. The obvious choice would have been to allow only type-1 and type-3 productions, but it is not difficult to see that boundary grammars with just these productions can only generate discrete graphs. Nagl [Nag] uses a similar definition for Chomsky NF, but he calls these grammars "grammars in normal form." Note that the difference between Chomsky NF and Greibach NF is that in Greibach NF type-3 productions have an additional terminal node.

Now we prove the Chomsky normal form result.

THEOREM 22. *For every B-eNCE language L there is a B-eNCE grammar G in Chomsky NF such that $L(G) = L$.*

Proof. Consider a B-eNCE grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$. It may be assumed that P contains no Λ-productions (see Theorem 10).

There will be defined now a B-eNCE grammar $\bar{G} = (\bar{\Sigma}, \Delta, \bar{\Gamma}, \Omega, \bar{P}, S)$ with $L(\bar{G}) = L(G)$ such that $\bar{P}$ only contains type-1, type-2, type-3, and chain productions. It should be clear that eliminating the chain productions in the way as indicated in the proof of Theorem 10 turns such a grammar into Chomsky NF.

The definition of $\bar{G}$ is as follows. First, let $\bar{\Sigma} = \Sigma \cup \{\langle \pi, i \rangle \mid \pi \in P, 1 \leqslant i \leqslant \# V_{\text{rhs}(\pi)}\}$. Second, let $\bar{\Gamma} = \Gamma \cup (\Gamma \times \{x \in V_{\text{rhs}(\pi)} \mid \pi \in P\})$. Next we define the productions in $\bar{P}$. Let $\pi = (X, D, B)$ be a production in P, and let $x_1, \ldots, x_n$ be a fixed order of the nodes of V_D, such that there is a k, $0 \leqslant k \leqslant n$, with $\varphi_D(x_i) \in \Delta$ for $1 \leqslant i \leqslant k$, and $\varphi_D(x_i) \in \Sigma - \Delta$ for $k < i \leqslant n$. Then $\bar{P}$ contains the production $\pi_0 = (X, D_0, B_0)$ and, for $1 \leqslant i \leqslant n$, the productions $\pi_i = (\langle \pi, i \rangle, D_i, B_i)$, where $D_i = (V_i, E_i, \bar{\Sigma}, \bar{\Gamma}, \varphi_i)$ and B_i are defined as follows.

Case 1. $i = 0$:

$V_0 = \{\xi_0\}$, where ξ_0 is a new node, $E_0 = \varnothing$, $\varphi_0(\xi_0) = \langle \pi, 1 \rangle$, and $B_0 = \{(\xi_0, \lambda, (\mu, z), a) \mid (z, \lambda, \mu, a) \in B\}$.

Case 2. $1 \leqslant i \leqslant n - 1$:

$V_i = \{x_i, \xi_i\}$, where ξ_i is a new node,

$E_i = \{(x_i, (\lambda, x_j), \xi_i) \mid i < j \leqslant n, \lambda \in \Gamma, \text{ and } (x_i, \lambda, x_j) \in E_D\}$,

$\varphi_i(x_i) = \varphi_D(x_i)$ and $\varphi_i(\xi_i) = \langle \pi, i + 1 \rangle$,

$B_i = B_i(x_i) \cup B_i(\xi_i)$, where $B_i(x_i) = \{(x_i, (\lambda, x_i), \lambda, a) \mid \lambda \in \Gamma, a \in \Sigma\}$, and $B_i(\xi_i) = \{(\xi_i, (\lambda, x_j), (\lambda, x_j), a) \mid \lambda \in \Gamma, a \in \Sigma, i < j \leqslant n\}$.

Case 3. $i=n$:

$V_n=\{x_n\}$, $E_n=\varnothing$, $\varphi_n(x_n)=\varphi_D(x_n)$, and $B_n=B_i(x_i)$, where $B_i(x_i)$ is as defined in Case 2, with $i=n$.

These are all the productions that $\bar{P}$ contains according to π.

$\bar{G}$ simulates G: whenever a production π in P generates n nodes, there are $n+1$ productions in $\bar{P}$ which together generate these nodes. The first of these productions is the chain production π_0: it replaces a node labeled lhs(π) by a nonterminal node (called ξ_0 above) labeled $\langle \pi, 1\rangle$. The other productions (π_1 to π_n) generate the nodes in rhs(π), one by one in a certain order. This order is such that terminal nodes are generated before nonterminal nodes. The ith node x_i of rhs(π) is generated by a nonterminal node, called ξ_{i-1} in the construction above, labeled $\langle \pi, i\rangle$, for $1\leqslant i\leqslant n$. ξ_{i-1} also generates a new nonterminal node ξ_i, labeled $\langle \pi, i+1\rangle$ (if $i\neq n$). So, in fact, ξ_{i-1} eventually generates $x_i, \ldots, x_n$.

An edge incident with a nonterminal node ξ_{i-1} labeled by $\langle \pi, i\rangle$ always has a label of the form (λ, x_j), where $\lambda\in\Gamma$, and $i\leqslant j\leqslant n$. This label indicates that, later on, ξ_{j-1} has to establish an edge labeled λ to x_j.

A nonterminal node x_i has to be generated after a terminal node x_j: otherwise, if i would be smaller than j, and there would be a λ-edge between these two nodes in rhsx(π), then E_i would contain a (λ, x_j)-edge between two nonterminal nodes, which is forbidden in boundary grammars.

This explanation should convince the reader that $\bar{G}$ can simulate each derivation of G. Furthermore, $\bar{G}$ clearly is a boundary grammar. Therefore, we may assume that $\bar{G}$ always applies the $n+1$ productions simulating π in a row, without interruptions (the order of rewriting is not important). Thus, the simulation of π is first finished before we start applying productions to other nodes. If we keep this in mind, then it is not difficult to see that each derivation in $\bar{G}$ corresponds to a derivation in G. Hence, $L(\bar{G})=L(G)$. ∎

In Section 8 of [EngLeiWel] a construction similar to the one above is used to show that directed N-eNCE languages can be generated by grammars with one or two nodes in the right-hand sides of the productions. We do not know whether this also holds for (undirected) N-eNCE languages. As far as eNCE grammars are concerned, however, a similar result can be obtained by using blocking edges (in [Nag, Theorem I.3.21] blocking edges are used to prove an analogous result). To see this, we have to reconsider the proof of Theorem 22. If the grammar G in that proof is an eNCE grammar (without Λ-productions), then we can turn $\bar{G}$ into an eNCE grammar with $L(G)=L(\bar{G})$ and with at most two nodes in the right-hand sides of the productions by adding a symbol \$ to $\bar{\Gamma}-\Omega$, adding (for all $(\lambda, y)\in\bar{\Gamma}$ and for all $a\in\Sigma$) tuples $(\xi_0, (\lambda, y), \$, a)$ to B_0, and adding all tuples $(v, \$, \$, a)$ to all B's. Clearly, \$-edges lead to blocking edges, and hence production π_0 may not be applied to a nonterminal node x as long as there are still (λ, y)-edges incident with x (indicating that some nodes that have to be connected to x, according to the

right-hand side of the production in which x was generated, are not yet present). It is left to the reader to write out full details.

Theorem 22 can be used to prove the second main theorem of this section: the Greibach normal form result for boundary grammars. For linear grammars, however, we can conclude directly from the previous proof that a Greibach NF exists (in [EngLei1] such grammars were called "one-linear").

THEOREM 23. *For every $L \in LIN\text{-}eNCE$ there is a linear eNCE grammar G in Chomsky NF and in Greibach NF such that $L = L(G)$.*

Proof. If we apply the proof of the theorem above to a linear eNCE grammar, then the resulting grammar that generates the same language has just type-1 and type-2 productions (after eliminating again the chain productions). Thus, it is both in Chomsky NF and in Greibach NF. ▮

THEOREM 24. *For every B-eNCE language L there is a B-eNCE grammar G in Greibach NF such that $L = L(G)$.*

Proof. Let $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ be a B-eNCE grammar in Chomsky normal form such that $L = L(G)$, according to Theorem 22. The theorem will be proved now in three steps. In the third step, a B-eNCE grammar $\bar{G}$ in Greibach NF will be constructed such that $L(\bar{G}) = L(G)$. The first and the second step are just technical: they give us the resources with which $\bar{G}$ can be built. Note that the only productions of G that are not yet of the right shape are the type-3 productions. Therefore we are mainly concerned with them in the rest of this proof.

Step 1. In this step it will be shown that we may assume that G has the following property:

(*) For every type-3 production $\pi = (X, D, B) \in P$, the embedding relation is $B = \{(v, \lambda, \lambda, a) \mid v \in V_D,\ \lambda \in \Gamma,\ a \in \Delta\}$.

This is shown by proving that there exists a B-eNCE grammar $\bar{G} = (\bar{\Sigma}, \Delta, \Gamma, \Omega, \bar{P}, \bar{S})$ in Chomsky normal form with property (*), such that $L(\bar{G}) = L(G)$. $\bar{G}$ can be constructed as follows. Let $\bar{\Sigma} = \Delta \cup \{\langle A, \alpha \rangle \mid A \in \Sigma - \Delta,\ \alpha \subseteq \Gamma \times \Gamma \times \Delta\}$, and let $\bar{S} = \langle S, \varnothing \rangle$.

It will be explained first why these new nonterminals are introduced. When G replaces a node x in a graph H by applying a type-3 production, the two newly generated nonterminal nodes, say y and z, take over some of the edges incident with x, and the labels of these edges may be changed. When a corresponding production of $\bar{G}$ is applied to x, then y (and z) has to take over unchanged all the edges incident with x (see (*)). Therefore, $\bar{G}$ places information in the label of y that determines how these edges should have been changed. More precisely, if the label of y is $\langle A, \alpha \rangle$ in the sentential form of $\bar{G}$, then y has label A in the sentential form of G, and if $(\lambda, \mu, a) \in \alpha$, with $\lambda, \mu \in \Gamma$ and $a \in \Delta$, then an edge labeled λ incident with x (and hence also with y in the sentential form of $\bar{G}$) and an a-labeled

node should be changed into an edge with label μ incident with y and that same a-labeled node to obtain the sentential form of G. This can be repeated iteratively, when y itself is replaced by a type-3 production.

To take care that nodes and edges get a label as described above, $\bar{P}$ consists of three kinds of productions:

(1) If (X, D, B) is a type-3 production in P with $V_D = \{y, z\}$, $\varphi_D(y) = Y \in \Sigma - \Delta$, and $\varphi_D(z) = Z \in \Sigma - \Delta$, then $\bar{P}$ contains for all $\alpha \subseteq \Gamma \times \Gamma \times \Delta$ the production $(\langle X, \alpha \rangle, \bar{D}, \bar{B})$, where $V_{\bar{D}} = V_D$, $E_{\bar{D}} = \varnothing$, $\varphi_{\bar{D}}(y) = \langle Y, \beta \rangle$, $\varphi_{\bar{D}}(z) = \langle Z, \gamma \rangle$ and $\bar{B} = \{(v, \lambda, \lambda, a) \mid v \in \{y, z\}, \lambda \in \Gamma, a \in \Delta\}$; β and γ are defined by $\beta = \{(\lambda, \mu, a) \mid \exists \bar{\mu} \in \Gamma\colon (\lambda, \bar{\mu}, a) \in \alpha$ and $(y, \bar{\mu}, \mu, a) \in B\}$ and $\gamma = \{(\lambda, \mu, a) \mid \exists \bar{\mu} \in \Gamma\colon (\lambda, \bar{\mu}, a) \in \alpha$ and $(z, \bar{\mu}, \mu, a) \in B\}$.

(2) If (X, D, B) is a type-2 production in P with $V_D = \{y, z\}$, $\varphi_D(y) = b \in \Delta$, and $\varphi_D(z) = Z \in \Sigma - \Delta$, then $\bar{P}$ contains for all $\alpha \subseteq \Gamma \times \Gamma \times \Delta$ the production $(\langle X, \alpha \rangle, \bar{D}, \bar{B})$, where $V_{\bar{D}} = V_D$, $E_{\bar{D}} = E_D$, $\varphi_{\bar{D}}(y) = b$, $\varphi_{\bar{D}}(z) = \langle Z, \gamma \rangle$ and $\bar{B} = \{(v, \lambda, \mu, a) \mid v \in \{y, z\}, \lambda, \mu \in \Gamma, a \in \Delta, \exists \bar{\mu} \in \Gamma\colon (\lambda, \bar{\mu}, a) \in \alpha$ and $(v, \bar{\mu}, \mu, a) \in B\}$; γ is here defined by $\gamma = \{(\lambda, \lambda, a) \mid \lambda \in \Gamma, a \in \Delta\}$.

(3) If (X, D, B) is a type-1 production in P, then $\bar{P}$ contains for all $\alpha \subseteq \Gamma \times \Gamma \times \Delta$ the production $(\langle X, \alpha \rangle, \bar{D}, \bar{B})$, where $\bar{D} = D$ and $\bar{B} = \{(y, \lambda, \mu, a) \mid y \in V_D, \lambda, \mu \in \Gamma, a \in \Delta, \exists \bar{\mu} \in \Gamma\colon (\lambda, \bar{\mu}, a) \in \alpha$ and $(y, \bar{\mu}, \mu, a) \in B\}$.

Step 2. In this second step, we look yet a little bit closer at the type-3 productions of G. In particular, we are interested in the graphs that can be obtained from a singleton graph by using type-3 productions only. Therefore, we define for each $A \in \Sigma - \Delta$ the set $\text{border}(A) = \{H \in GR_{\Sigma, \Gamma} \mid A \overset{*}{\Rightarrow} H$ by using type-3 productions only$\}$. Border(A) clearly contains discrete graphs with nonterminal node labels only. It is furthermore not difficult to find, for each $A \in \Sigma - \Delta$, a context-free string grammar $\bar{G}_A$ with the properties

(i) for every $H \in \text{border}(A)$ there is a $w \in L(\bar{G}_A)$ such that $\#_Y(w) = \#_Y(H)$ for every $Y \in \Sigma - \Delta$, and

(ii) for every $w \in L(\bar{G}_A)$ there is an $H \in \text{border}(A)$ such that $\#_Y(H) = \#_Y(w)$ for every $Y \in \Sigma - \Delta$.

The definition of $\bar{G}_A$ is similar to the one for CF(G) in Definition 15. Note that $\bar{G}_A$ uses the nonterminals of G as its own terminals, so it will be necessary to introduce new nonterminals for $\bar{G}_A$. It now follows from Theorem 7.2 of Part I of [Sal] (a consequence of the well-known Parikh theorem) that there is, for each $A \in \Sigma - \Delta$, a right-linear string grammar $G_A = (\Sigma_A, \Sigma - \Delta, P_A, Q_A)$ such that (i) and (ii) above hold for G_A too, and so border(A) can be seen as a regular language. G_A will be used in the next step.

Step 3. We now combine the results of Step 1 and Step 2. Assume that G has property (*), according to Step 1. Consider a graph $K \in S(G)$, and a nonterminal node x of K with label A. If we now consecutively apply a number of type-3

productions to x and the descendants of x, then the net result is that we pick a discrete graph $H \in \text{border}(A)$ and plug it into K in place of x in such a way that each node of H inherits unchanged all the edges that were incident with x in K. Hence, since this plugging process is so simple, the right-linear string grammar G_A (as defined in Step 2) determines exactly which graphs can be obtained from K by applying type-3 productions to x and its descendants. After observing this, we are now ready to define a B-eNCE grammar $\bar{G} = (\bar{\Sigma}, \Delta, \Gamma, \Omega, \bar{P}, \bar{S})$ in Greibach normal form such that $L(\bar{G}) = L(G)$, as follows. First, we assume that for all $A, B \in \Sigma - \Delta$ with $A \neq B$, $x \in \Sigma_A \cap \Sigma_B$ implies that $x \in \Sigma - \Delta$ (i.e., the sets of nonterminals of G_A and G_B are disjoint). Then we define $\bar{\Sigma} = \bigcup \{\Sigma_A - (\Sigma - \Delta) \mid A \in \Sigma - \Delta\} \cup \Delta$ and $\bar{S} = Q_S$. Thus, $\bar{\Sigma}$ contains the nonterminals of G_A, for all $A \in \Sigma - \Delta$, and the terminals of G. The initial nonterminal of G_S is also the initial nonterminal of $\bar{G}$. Finally, $\bar{P}$ is defined as follows (with $A \in \Sigma - \Delta$, $Q, Q' \in \Sigma_A - (\Sigma - \Delta)$, $X, Y \in \Sigma - \Delta$, and $a \in \Delta$):

(i) If $Q \to XQ'$ is a production in P_A, and if $\pi = (X, D, B)$ is a type-2 production in P, with $V_D = \{x, y\}$, $\varphi_D(x) = a$ and $\varphi_D(y) = Y$, then $\bar{P}$ contains the production $\bar{\pi} = (Q, \bar{D}, \bar{B})$, with $V_{\bar{D}} = \{x, y, z\}$, $E_{\bar{D}} = E_D$, $\varphi_{\bar{D}}(x) = a$, $\varphi_{\bar{D}}(y) = Q_Y$, and $\varphi_{\bar{D}}(z) = Q'$. Furthermore, $\bar{B} = B \cup \{(z, \lambda, \lambda, a) \mid \lambda \in \Gamma, a \in \Delta\}$.

(ii) If $Q \to XQ'$ is a production in P_A, and if $\pi = (X, D, B)$ is a type-1 production in P, with $V_D = \{x\}$, and $\varphi_D(x) = a$, then $\bar{P}$ contains the production $\bar{\pi} = (Q, \bar{D}, \bar{B})$, with $V_{\bar{D}} = \{x, z\}$, $E_{\bar{D}} = \varnothing$, $\varphi_{\bar{D}}(x) = a$ and $\varphi_{\bar{D}}(z) = Q'$. Furthermore, as in (i), $\bar{B} = B \cup \{(z, \lambda, \lambda, a) \mid \lambda \in \Gamma, a \in \Delta\}$.

(iii) If $Q \to X$ is a production in P_A, and if $\pi = (X, D, B)$ is a type-2 production in P, defined as in (i), then $\bar{P}$ contains the production $\bar{\pi} = (Q, \bar{D}, B)$, with $V_{\bar{D}} = V_D$, $E_{\bar{D}} = E_D$, $\varphi_{\bar{D}}(x) = a$, and $\varphi_{\bar{D}}(y) = Q_Y$.

(iv) If $Q \to X$ is a production in P_A, and if $\pi = (X, D, B)$ is a type-1 production in P, then $\bar{P}$ contains the production $\bar{\pi} = (Q, D, B)$.

A little comment on this construction will be useful now. In (i), for example, G_A is half-way generating the labels of a graph in border(A). At this moment, $\bar{G}$ does not generate an X-labeled node, as suggested by G_A, but applies a type-2 production to this node and generates the right-hand side D, with Q_Y, the initial nonterminal of G_Y, instead of Y as the label of y. Q_Y is used here since G may apply $\geqslant 0$ type-3 productions to y and the descendants of y. G_Y takes care that this can be done properly. Moreover, $\bar{G}$ generates a node labeled Q' that generates the rest of the graph in border(A). Since (*) of Step 1 holds, the embedding relation $\bar{B}$ indeed works correctly.

Hence, these four types of productions simulate the productions of G. With the explanation given above, it should not be difficult to see that $L(\bar{G}) = L(G) = L$. ∎

As an example of the use of Greibach NF we mention that it considerably simplifies recognition algorithms: just one node of the given graph needs to be recognized at each step of the algorithm (see [EngLei3]).

Finally, we want to mention that there is no Chomsky or Greibach NF for A-eNCE grammars. In fact, when a fixed bound m on the number of nodes in the right-hand sides of the productions of an A-eNCE grammar is assumed, then only graphs with nodes of degree $O(m^2)$ can be generated (see the proof of Lemma 25 of [EngLeiRoz1]). Moreover, an apex grammar with exactly one terminal node in the right-hand side of each production can only generate graphs without cycles (whereas the apex language of Example 3 contains cycles).

5. A Characterization of Boundary and Linear Languages

The previous section has provided us with a nice normal form for B-eNCE grammars: the Greibach NF. This normal form will be used in this section to prove a characterization theorem for B-eNCE languages in terms of regular (string and tree) languages. It will be shown that a B-eNCE language can be obtained from a regular tree language by using, for each final edge label λ, a regular string language that tells whether a λ-edge has to be established between two nodes in such a tree. In fact, it t is a tree in this regular tree language, and $a_1, a_2, \ldots, a_n$ are, in this order, the labels of a path in t that leads from a node to one of its descendants, then a λ-edge is established between these two nodes of the tree if $a_1 a_2 \cdots a_n$ is a string in the regular language belonging to λ. More precisely, the characterization theorem states that L is a B-eNCE language if and only if L is a node relabeling of a graph language that can be obtained from a regular tree language as explained above. Using a regular string language instead of the regular tree language, a characterization of the LIN-eNCE languages is obtained, entirely in terms of regular string languages.

Intuitively, the regular tree language is the set of all derivation trees of a B-eNCE grammar (in Greibach NF). Since, in a boundary grammar, only descendant nodes can be connected by edges, this gives the above path property. Note that the Greibach NF ensures that each node in a derivation tree corresponds to exactly one node in the corresponding graph.

The characterization results of this section may be viewed as generalizations of similar results in the literature for regular (tree) languages: every regular language is the "relabeling" of a local regular language (cf. Proposition I.4.3 of [Ber]), and every regular tree language is the relabeling of the set of derivation trees of a context-free grammar [Tha]. In fact, these results can also be used to sharpen our characterization results: in the discussion above we may assume the regular tree language to be the set of derivation trees of a context-free grammar and each regular language to be local.

First we need some definitions, to formalize the concepts explained above.

Definition 25. A *ranked alphabet* is an alphabet Δ together with a mapping $\text{rank}: \Delta \to \{0, 1, 2, \ldots\}$. For $k \geqslant 0$, $\Delta_k = \{\sigma \in \Delta \mid \text{rank}(\sigma) = k\}$. The set of *trees over* Δ, denoted T_Δ, is the smallest subset of Δ^* such that (i) for every $\sigma \in \Delta_0$, σ is in T_Δ,

and (ii) for every $\sigma \in \Delta_k$ with $k \geqslant 1$, and for every $t_1, t_2, ..., t_k \in T_\Delta$, $\sigma t_1 t_2 \cdots t_k$ is in T_Δ. For a set Y, $T_\Delta[Y]$ denotes $T_{\Delta \cup Y}$, where the elements of Y are given rank 0. A language $L \subseteq \Delta^*$ is called a *tree language* if $L \subseteq T_\Delta$.

If $t \in T_\Delta$, then tree(t) is the usual directed graph corresponding to t, without edge labels and with node labels in Δ (for the notion of directed graph, see Section 1). We assume that edges in tree(t) point from the root downwards. Note that the mapping "tree" is not injective, because tree(t) is unordered. It is left to the reader to give a formal definition of tree(t).

In a similar way, we can associate a directed graph with each string. If $w = w_1 \cdots w_n$ is a string over an alphabet Δ, with $n \geqslant 0$ and $w_i \in \Delta$ for $1 \leqslant i \leqslant n$, then chain($w$) is defined as the directed graph H without edge labels with $V_H = \{1, 2, ..., n\}$, $E_H = \{(i, *, i+1) \mid 1 \leqslant i \leqslant n-1\}$, and $\varphi_H(i) = w_i$ for $1 \leqslant i \leqslant n$. Observe that the edges in E_H are directed.

DEFINITION 26. A *regular tree grammar* is a context-free string grammar $G = (\Sigma, \Delta, P, S)$ such that Δ is a ranked alphabet and, for every production $X \to \xi$ in P, $\xi \in T_\Delta[\Sigma - \Delta]$. G is *in normal form* if, moreover, $\xi \in \Delta(\Sigma - \Delta)^*$.

The class of languages that can be generated by a regular tree grammar is denoted REGT. In [GécSte, Lemma II.3.4], it is proved that for every $L \in$ REGT there is a regular tree grammar G in normal form such that $L = L(G)$. From now on we assume that each regular tree grammar is in normal form.

Next, the process of transforming a regular string or tree language into a graph language by using a set of regular string languages, as described above, is defined formalyy.

DEFINITION 27. Let Δ and Ω be alphabets.

(i) Let $w \in \Delta^*$, and let $D = \text{chain}(w)$ be the directed graph $D = (V, E, \Delta, \{*\}, \varphi)$. Let, furthermore, K be a mapping from Ω to the class of string languages over Δ, i.e., for every $\lambda \in \Omega$, $K(\lambda) \subseteq \Delta^*$. Then fat-chain($w, K$) is the (undirected) graph $H \in GR_{\Delta, \Omega}$ with $V_H = V$, $E_H = \{(x, \lambda, y) \mid x, y \in V, \lambda \in \Omega$: there is a directed path $v_1, v_2, ..., v_r$ in D, with $r \geqslant 2$, such that $v_1 = x$, $v_r = y$, and $\varphi(v_1)\varphi(v_2)\cdots\varphi(v_r) \in K(\lambda)\}$, and $\varphi_H = \varphi$. If $M \subseteq \Delta^*$ is a string language, then FAT-CHAIN(M, K) is the graph language $\{\text{fat-chain}(w, K) \mid w \in M\}$.

(ii) Let Δ be ranked, let $t \in T_\Delta$, and let K be as described in (i). Then fat-tree(t, K) is defined as fat-chain(w, K) in (i), with $D = \text{tree}(t)$ instead of $D = \text{chain}(w)$. If $M \subseteq T_\Delta$, then FAT-TREE(M, K) = $\{\text{fat-tree}(t, K) \mid t \in M\}$.

(iii) A graph language $L \subseteq GR_{\Delta, \Omega}$ is *regular substring defined* if there exist an alphabet $\bar{\Delta}$, a mapping K from Ω to the set of regular string languages over $\bar{\Delta}$, a regular string language $M \subseteq \bar{\Delta}^*$, and a node relabeling $\rho: \bar{\Delta} \to \Delta$ such that $L = \rho(\text{FAT-CHAIN}(M, K))$. If $\bar{\Delta}$ is ranked, $M \subseteq T_{\bar{\Delta}}$ is a regular tree language, and $L = \rho(\text{FAT-TREE}(M, K))$ then we say that L is *regular path defined.*

To make the definition above a bit clearer, we give some examples of languages which are regular path or substring defined.

EXAMPLE 28. First, the language L of Example 3 (the "ladders") is regular substring defined, for $L = \rho(\text{FAT-CHAIN}(M, K))$, with regular string languages M and $K(*)$ over an alphabet $\bar{\Delta} = \{a, b\}$ and relabeling $\rho: \bar{\Delta} \to \{a\}$ defined as follows: $\rho(a) = \rho(b) = a$, $M = (ab)^*$, and $K(*) = \{ab, aba, bab\}$. In Fig. 8 a picture can be found of fat-chain(w, K), with $w = (ab)^4$; the broken edges belong to chain(w). Applying ρ to this graph clearly results in the graph of Fig. 1b.

Second, the language L of Example 7 (the binary trees) is regular path defined. We can take $\bar{\Delta} = \{a, b\}$, where $\bar{\Delta}_0 = \{b\}$ and $\bar{\Delta}_2 = \{a\}$. M is defined by the regular tree grammar with productions $S \to aSS$ and $S \to b$, $K(*) = \{aa, ab\}$, and ρ is defined as above. It is easy to see that $L = \rho(\text{FAT-TREE}(M, K))$. If $K(*) = \{a, b\}^*$ instead, then L is the set of all "transitively closed" binary trees.

Third, the language L of Example 6 is again regular substring defined. This time we can take $M = a^*$ and $K(*) = aaaa^*$. Clearly, $L = \text{FAT-CHAIN}(M, K)$. If $K(*) = a^*$ instead, then L is the set of all complete graphs without edge labels and with node label a.

After these examples, we wish to point out a relationship between the notions "regular path defined" and "regular substring defined". Let $L = \rho(\text{FAT-TREE}(M, K))$ as in (iii) of Definition 27, and $M = L(G)$ for some regular tree grammar G in normal form. Suppose now that G happens to be a right-linear string grammar (recall from Definition 26 that G is a special kind of context-free string grammar). Then it easily follows from Definition 27 that $L = \rho(\text{FAT-CHAIN}(M, K))$, too. On the other hand, if $L = \rho(\text{FAT-CHAIN}(M, K))$ and $M \subseteq \Sigma_1^* \Sigma_0$ for two disjoint alphabets Σ_1 and Σ_0, then each (reduced) right-linear string grammar generating M is also a regular tree grammar (over the ranked alphabet $\Sigma_1 \cup \Sigma_0$, with $\text{rank}(a) = 1$ if $a \in \Sigma_1$ and $\text{rank}(a) = 0$ if $a \in \Sigma_0$). In this case, $L = \rho(\text{FAT-TREE}(M, K))$ too. Note, however, that this does not work for, e.g., a regular string language $M = a^*$. Still, it is not difficult to see that if a language is regular substring defined, then it is also regular path defined; this will also follow from Theorems 31 and 32.

Next the characterization theorem for boundary eNCE languages is proved in two steps.

LEMMA 29. *Let L be a graph language that is regular path defined. Then $L \in$ B-eNCE.*

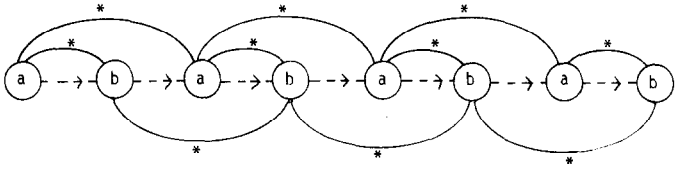

FIGURE 8

Proof. Let Δ and Ω be alphabets such that $L \subseteq GR_{\Delta,\Omega}$. Since L is regular path defined, there is a ranked alphabet $\bar{\Delta}$, there is a regular tree language $M \subseteq T_{\bar{\Delta}}$, there are, for every $\lambda \in \Omega$, regular languages $K(\lambda) \subseteq \bar{\Delta}^*$, and there is a node relabeling $\rho: \bar{\Delta} \to \Delta$ such that $L = \rho(\text{FAT-TREE}(M, K))$. Let now $G = (\Sigma, \bar{\Delta}, P, S)$ be a regular tree grammar in normal form and let, for every $\lambda \in \Omega$, $G_\lambda = (\Sigma_\lambda, \bar{\Delta}, P_\lambda, S_\lambda)$ be a right-linear string grammar, such that $L(G) = M$ and $L(G_\lambda) = K(\lambda)$, for every $\lambda \in \Omega$.

We will now show how to define a B-eNCE grammar $\bar{G} = (\Sigma, \bar{\Delta}, \bar{\Gamma}, \Omega, \bar{P}, S)$ such that $L(\bar{G}) = \text{FAT-TREE}(M, K)$. Since B-eNCE is closed under node relabeling (see Theorem 9), the lemma will follow from this.

First, $\bar{\Gamma} = \{\langle Q, \lambda \rangle \mid \lambda \in \Omega,\ Q \in \Sigma_\lambda - \bar{\Delta}\} \cup \Omega$. Then the productions in $\bar{P}$ can be defined as follows: If $X \to aY_1 \cdots Y_n$ is a production in P, with $n \geqslant 0$ and $X, Y_1, \ldots, Y_n \in \Sigma - \bar{\Delta}$ and $a \in \bar{\Delta}$, then $\bar{P}$ contains the production $\pi = (X, D, B)$, where

$$V_D = \{x, y_1, \ldots, y_n\} \text{ (new nodes, all different)},$$
$$\varphi_D(x) = a,\ \varphi_D(y_i) = Y_i \text{ for all } 1 \leqslant i \leqslant n,$$
$$E_D = \{(x, \langle Q, \lambda \rangle, y_i) \mid \lambda \in \Omega,\ Q \in \Sigma_\lambda - \bar{\Delta},\ 1 \leqslant i \leqslant n\colon S_\lambda \to aQ \in P_\lambda\}, \text{ and}$$
$$B = \{(x, \langle Q, \lambda \rangle, \lambda, b) \mid \lambda \in \Omega,\ Q \in \Sigma_\lambda - \bar{\Delta},\ b \in \bar{\Delta}\colon Q \to a \in P_\lambda\}$$
$$\cup \{(y_i, \langle Q, \lambda \rangle, \langle Q', \lambda \rangle, b) \mid \lambda \in \Omega,\ Q, Q' \in \Sigma_\lambda - \bar{\Delta},\ b \in \bar{\Delta},\ 1 \leqslant i \leqslant n\colon Q \to aQ' \in P_\lambda\}.$$

$\bar{P}$ contains no more productions than the ones defined above. While $\bar{G}$ is generating one of the trees in $L(G)$, it keeps track of the states (=nonterminals) in which G_λ can be after following a path form a terminal tto a nonterminal node in this tree by placing them, together with λ, on edges between these two nodes. More precisely, for a terminal node x and a nonterminal node y in this tree, there is an edge labeled $\langle Q, \lambda \rangle$ between x and y if and only if G_λ can be in state Q after accepting the labels (in $\bar{\Delta}$) on the path from x to y. In this way, $\bar{G}$ establishes the correct edges in the terminal tree. ∎

LEMMA 30. *Let $L \in$ B-eNCE. Then L is regular path defined.*

Proof. Let $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ be a B-eNCE grammar in Greibach normal form (cf. Theorem 24) such that $L(G) = L$. Since the constructions in Lemma 19 and Theorem 20 preserve the Greibach property, we may assume that G is non-blocking. Moreover, it can easily be seen that we may assume that for each $\pi \in P$ no two nodes have the same label in $\text{rhs}(\pi)$. In order to prove this lemma, it has to be shown that there is an alphabet $\bar{\Delta}$, there is a regular tree language $M \subseteq T_{\bar{\Delta}}$, there are, for every $\lambda \in \Omega$, regular languages $K(\lambda) \subseteq \bar{\Delta}^*$, and there is a node relabeling $\rho: \bar{\Delta} \to \Delta$ such that $L = \rho(\text{FAT-TREE}(M, K))$. These four objects are defined in (i)–(iv) as follows:

(i) The definition of $\bar{\Delta}$. We let $\bar{\Delta} = P$, where $\text{rank}(\pi) = \# V_{\text{rhs}(\pi)} - 1$, for

every $\pi \in P$. Hence, the rank of a production equals the number of nonterminal nodes in its right-hand side.

(ii) The definition of M by way of a regular tree grammar $\bar{G} = (\bar{\Sigma}, \bar{\Delta}, \bar{P}, \bar{S})$ such that $L(\bar{G}) = M$. Let $\bar{\Sigma} = (\Sigma - \Delta) \cup \bar{\Delta}$, and $\bar{S} = S$. Then $\bar{P}$ can be defined by $\bar{P} = \{\text{lhs}(\pi) \to \pi Y_1 Y_2 \cdots Y_n \mid \pi \in P,\ n = \text{rank}(\pi),\ Y_1, Y_2, \ldots, Y_n \in \Sigma - \Delta$: $[x \in V_{\text{rhs}(\pi)}$ implies $\exists i,\ 1 \leqslant i \leqslant n$: $\varphi_{\text{rhs}(\pi)}(x) = Y_i]\}$. Hence, a production in $\bar{P}$ generates a terminal $\pi\ (\in P)$ followed by the labels of all nonterminal nodes of $\text{rhs}(\pi)$ in some arbitrary order. The label of the only node in $\text{rhs}(\pi)$ is not generated. This label will later be placed on the node again by the relabeling ρ. Hence, in fact, $\{\text{tree}(t) \mid t \in L(\bar{G})\}$ is the set of all derivation trees of G as defined in [EngLeiRoz1].

(iii) The definition of $K(\lambda)$ by way of a right-linear grammar $G_\lambda = (\Sigma_\lambda, \bar{\Delta}, P_\lambda, S_\lambda)$ such that $L(G_\lambda) = K(\lambda)$, for every $\lambda \in \Omega$. To give the definition of G_λ, we need some additional definitions.

First, let $\pi = (X, D, B)$ be a production in P, and let $x \in V_D$ be a nonterminal node. Then $\langle \pi \rangle_x$ is defined to be the production (X, D', B') such that D' is the subgraph of D induced by $\{x\}$, and B' is the subset of B consisting of all tuples concerning x. So $\langle \pi \rangle_x$ simulates π only with respect to this particular nonterminal node.

Second, let ξ_T and ξ_{NT} be two different new objects. If now D is a graph consisting of one terminal node and one nonterminal node, then $\langle D \rangle$ is the graph isomorphic to D such that ξ_T and ξ_{NT} are nodes of $\langle D \rangle$, and ξ_T has a terminal label (hence, ξ_{NT} has a nonterminal label). It is obvious that there is exactly one graph D' such that $D' = \langle D \rangle$, for each such graph D.

Third, let $\langle GR_{\Sigma, \Gamma} \rangle = \{\langle D \rangle \mid D \in GR_{\Sigma, \Gamma}$ is a graph with one terminal node and one nonterminal node$\}$. It is obvious that $\langle GR_{\Sigma, \Gamma} \rangle$ is a finite set.

Now we can define G_λ itself. Let $\Sigma_\lambda = \langle GR_{\Sigma, \Gamma} \rangle \cup \bar{\Delta} \cup \{S_\lambda\}$, where S_λ is a new symbol. Then P_λ is defined as follows.

First, for all $\pi \in P$, and for all subgraphs D of $\text{rhs}(\pi)$ induced by the terminal node and one of the nonterminal nodes of $\text{rhs}(\pi)$: (1) $S_\lambda \to \pi \langle D \rangle$ is a production in P_λ. Second, if $\pi \in P$, then for every $\langle H \rangle \in \langle GR_{\Sigma, \Gamma} \rangle$ and for every nonterminal node $x \in V_{\text{rhs}(\pi)}$, P_λ contains the production: (2) $\langle H \rangle \to \pi \langle \bar{H} \rangle$, where $\bar{H}$ is a graph such that $\langle H \rangle \Rightarrow_{(\xi_{NT}, \langle \pi \rangle_x)} \bar{H}$. If, furthermore, N is a graph such that $\langle H \rangle \Rightarrow_{(\xi_{NT}, \pi)} N$, and there is a λ-edge between the two terminal nodes in N, then P_λ also contains the production: (3) $\langle H \rangle \to \pi$. These are all the productions in P_λ.

G_λ follows a path downwards through a (derivation) tree generated by $\bar{G}$ (starting anywhere in the tree, see production (1)), and it reads the labels (i.e., productions of G) on this path. It then decides, by only looking at these labels, whether a λ-edge should be added between the first node on this path and the last one. G_λ can do this by just following the derivation of G that has led to this derivation tree. In particular, when G_λ is half-way accepting the string of productions belonging to the path in consideration, it "remembers" the terminal node that has been generated by the production on the first node of the path, the nonterminal

node that is the left-hand side of the first production that has not yet been accepted so far, and the set of all edges between these two nodes. This information is stored in the nonterminal of G_λ. After accepting the next production on this path, the information can be updated, see production (2). The assumption made at the beginning of this proof ensures that the information remembered in the nonterminal of G_λ is uniquely determined, in the sense that there are not two different paths leading downwards from a node in the tree labeled by the same productions. When G_λ finally accepts the last production on the path, it can easily decide whether there is a λ-edge between the terminal node that was generated in the first production on the path and the terminal node that is now generated by this last production. Depending on this, production (3) is added to P_λ or not.

(iv) The definition of ρ. For each $\pi \in \bar{\Delta} = P$, $\rho(\pi)$ is the label of the terminal node of $\text{rhs}(\pi)$.

With the four definitions given above, it is not difficult to see that indeed $L = \rho(\text{FAT-TREE}(M, K))$. ▮

Note that, since G in the proof above is in Greibach NF, the elements in $\bar{\Delta}$ have rank $\leqslant 2$.

In order to illustrate the lemma above, we consider the B-eNCE grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ with $\Sigma = \{S, Y, Z, a\}$, $\Delta = \{a\}$, $\Gamma = \{\lambda, 1, 2, 3\}$, $\Omega = \{1, 2, 3\}$, and with productions π_1 to π_4 as defined in Fig. 9. Clearly, G is in Greibach NF, and $L(G)$ is the set of "k-leaved clovers," for all $k \geqslant 1$ (where the 4-leaved clover is drawn in Fig. 10). The regular tree grammar $\bar{G}$ as defined in the proof of Lemma 30 has productions $S \to \pi_1 Y$, $Y \to \pi_2 YZ$, $Y \to \pi_2 ZY$, $Y \to \pi_3 Z$, and $Z \to \pi_4$. Here the productions of G are the terminals of $\bar{G}$, with $\text{rank}(\pi_1) = 1$, $\text{rank}(\pi_2) = 2$, $\text{rank}(\pi_3) = 1$, and $\text{rank}(\pi_4) = 0$. Taking $K(1) = \pi_1 \pi_2^*(\pi_2 + \pi_3)$, $K(2) = (\pi_2 + \pi_3)\pi_4$, $K(3) = \pi_1 \pi_2^*(\pi_2 + \pi_3)\pi_4$, and $\rho(\pi_i) = a$ for $1 \leqslant i \leqslant 4$, it is not difficult to see that $\rho(\text{FAT-TREE}(L(\bar{G}), K)) = L(G)$. As an example, Fig. 11 shows fat-tree$(\pi_1 \pi_2 \pi_4 \pi_2 \pi_4 \pi_2 \pi_4 \pi_3 \pi_4, K)$; it is clear that applying ρ to this graph results in the 4-leaved clover of Fig. 10.

With these two lemmas, we have proved the main theorem of this section.

THEOREM 31. *For every graph language L, $L \in$ B-eNCE if and only if L is regular path defined.*

Proof. Immediately from the previous lemmas. ▮

As a direct consequence of the proofs of Lemma 29 and 30, we can also show now that a language is linear if and only if it is regular substring defined.

THEOREM 32. *$L \in$ LIN-eNCE if and only if L is regular substring defined.*

Proof. This theorem follows from the proofs of the previous two lemmas. First, if in the proof of Lemma 29, G is a right-linear string grammar instead of a regular tree grammar, then $\bar{G}$ as defined in that proof is a linear eNCE grammar. Second,

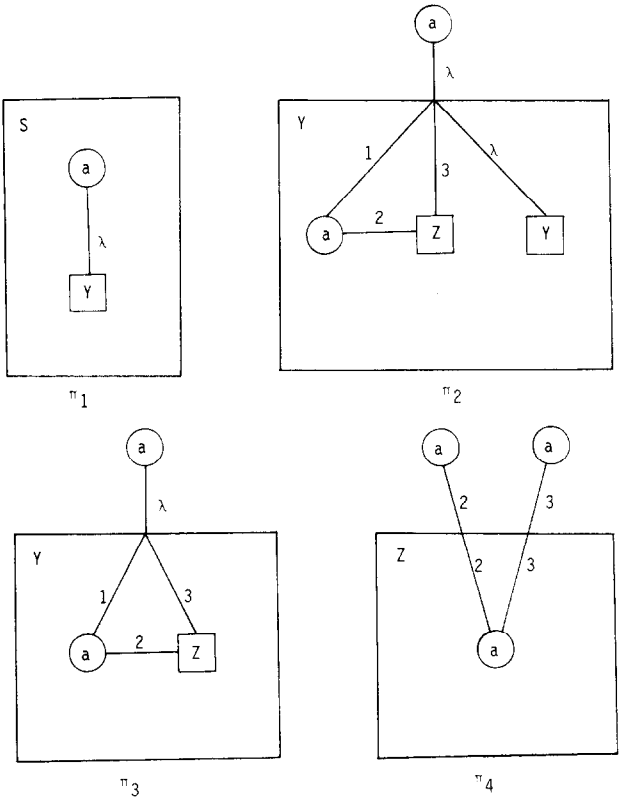

FIGURE 9

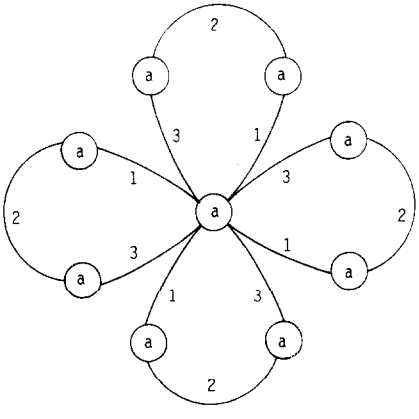

FIGURE 10

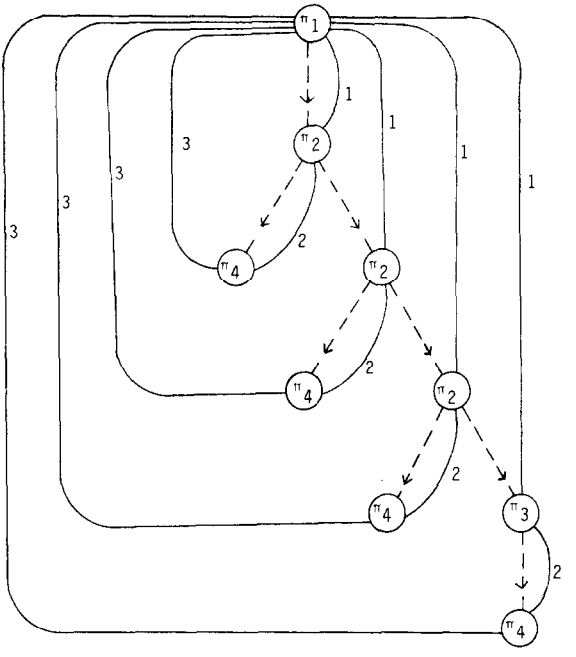

FIGURE 11

if G in the proof of Lemma 30 is a linear eNCE grammar in Greibach NF (see Theorem 23), then $\bar{G}$ as defined in that proof is clearly a right-linear string grammar. ▮

In [EngLeiRoz1, Section 6], a characterization theorem can be found for A-dNCE languages, i.e., apex NCE languages consisting of directed graphs without edge labels. This characterization is also based on a certain operation on the set of derivation trees of a graph grammar to obtain the language of this grammar. Only the operation used is quite different: nodes in the tree are replaced by a "cloud" of nodes (the terminal nodes in the right-hand side of the label of the node in the tree), and then edges are added between nodes in "neighbouring" clouds, according to some connection relation. This theorem can easily be adapted for A-cNCE languages (see [EngLeiRoz2]). We conjecture that a characterization result similar to Theorems 31 and 32 can also be shown for A-eNCE grammars. In fact, we think that a graph language is in A-eNCE if and only if it is regular path defined with finite sets $K(\lambda)$, for all $\lambda \in \Omega$.

There is as yet no characterization of the class of (N-)eNCE languages. This is probably due to the fact that derivation trees for (N-)eNCE grammars do not exist, since the order of rewriting is important. It may be of interest to consider confluent N-eNCE grammars (see [Cou]).

These two characterization theorems (Theorems 31 and 32) enable us to specify boundary and linear eNCE languages in an elegant direct way, as can be seen from Example 28. As another illustration of the use of such results we show that LIN-eNCE is closed under the operation of taking the complement of the edges of a graph. This result will be used again in the following section.

DEFINITION 33. Let $H=(V, E, \Sigma, \Gamma, \varphi)$ be a graph. Then the *edge-complement* of H is $\mathrm{com}(H)=(V, \bar{E}, \Sigma, \Gamma, \varphi)$, where $\bar{E}=\{(x, \mu, y) \mid x, y \in V,\ \mu \in \Gamma: x \neq y, (x, \mu, y) \notin E\}$. Let $L \subseteq GR_{\Sigma, \Gamma}$ be a graph language. Then $\mathrm{com}(L)=\{\mathrm{com}(H) \mid H \in L\}$.

THEOREM 34. *If* $L \in$ *LIN-eNCE, then* $\mathrm{com}(L) \in$ *LIN-eNCE.*

Proof. Let $L \subseteq GR_{\Delta, \Omega}$. The theorem follows easily from Theorem 32, as follows. Since $L \in$ LIN-eNCE, $L=\rho(\text{FAT-CHAIN}(M, K))$, for a relabeling $\rho: \bar{\Delta} \to \Delta$, and regular string languages M and $K(\lambda)$, for every $\lambda \in \Omega$. Clearly, $\mathrm{com}(L)=\rho(\text{FAT-CHAIN}(M, K^c))$, where $K^c(\lambda)=\bar{\Delta}^*\text{-}K(\lambda)$, for every $\lambda \in \Omega$. Thus, since the class of regular string languages is closed under complement, it follows that $\mathrm{com}(L)$ is a linear eNCE language too. ∎

The reader might already expect that a similar result cannot be obtained for boundary eNCE languages. Intuitively, this is due to the fact that the complement of $K(\lambda)$ can establish the complementary λ-edges on a path through the (derivation) tree, but it does not establish the edges between nodes that are on different paths through this tree. Formally, this can be proved as follows.

THEOREM 35. *There exists an* $L \in$ *B-eNCE such that* $\mathrm{com}(L) \notin$ *B-eNCE.*

Proof. Let L be any graph language of bounded degree in B-eNCE but not in LIN-eNCE. The set of binary trees (Example 7) is an example of such a language, see [EngLei1]. We will show that $\mathrm{com}(L) \notin$ B-eNCE. Assume, to the contrary, that $G=(\Sigma, \Delta, \Gamma, \Omega, P, S)$ is a nonblocking B-eNCE grammar without Δ-productions such that $L(G)=\mathrm{com}(L)$ (cf. Theorems 20 and 10). We will derive a contradiction from this. Clearly, from Theorem 32 it follows that $\mathrm{com}(L) \notin$ LIN-eNCE. From [EngLei1, Theorem 12] it thus follows that G is not "nonterminal bounded," which means that there is for each $k \geqslant 1$ a derivation $S \overset{*}{\Rightarrow} H_k$ such that H_k contains at least k nonterminal nodes. But since G contains no Δ-productions, each of these nonterminal nodes generates at least one terminal node. Furthermore, since G is boundary, at least k terminal nodes will be mutually disconnected when the derivation above finally results in a terminal graph (which is in $L(G)$, for G is nonblocking). This contradicts the fact that L is of bounded degree. ∎

The set of binary trees mentioned in the proof above is also an A-eNCE language. Hence, A-eNCE is not closed under edge-complement either. Finally, we want to remark that N-eNCE and eNCE are closed under edge-complement (this

shows in another way that B-eNCE is properly included in N-eNCE). The proof of this is left to the reader.

6. Comparison with NCE Grammars

In this section we look at the restricted versions of eNCE grammars that can only generate graphs without edge labels (we have called them e_1NCE grammars in Section 2). This enables us to compare our grammars with NCE grammars [JanRoz5]. Recall that N-e_1NCE is closed under node relabeling (cf. Theorem 9), but NCE is not [RozWel2]. Thus, since NCE grammars are a special case of N-e_1NCE grammars, it follows directly that N-e_1NCE grammars are more powerful than NCE grammars and that the (node) relabelings of NCE languages are also N-e_1NCE languages (this also holds in the boundary, linear, and apex case). However, we will show the existence of very simple linear e_1NCE languages which are not the relabeling of any NCE language. Hence, (boundary, linear) N-e_1NCE grammars have even more power than (boundary, linear) NCE grammars equiped with a relabeling. In the apex case, however, it will be shown that e_1NCE and NCE grammars with a relabeling are equally strong (this "explains" the important role played in [EngLeiRoz1] by the class of relabelings of apex NCE languages).

First, we show the result concerning apex grammars.

THEOREM 36. *A-e_1NCE = R(A-NCE).*

Proof. The fact that R(A-NCE) is a subset of A-e_1NCE is trivial, since A-e_1NCE is closed under node relabeling, cf. the proof of Theorem 9. We will next show it the other way around.

Let $G = (\Sigma, \Delta, \Gamma, \{*\}, P, S)$ be an A-e_1NCE grammar. It may be assumed that G is nonblocking, cf. Theorem 20. We construct an A-NCE grammar $\bar{G} = (\bar{\Sigma}, \bar{\Delta}, \{*\}, \{*\}, \bar{P}, S)$ and a node relabeling $\rho: \bar{\Delta} \to \Delta$ such that $\rho(L(\bar{G})) = L(G)$. First, define $\bar{\Sigma}$ and $\bar{\Delta}$ by $\bar{\Sigma} = \{S\} \cup \{\langle X, v, \pi\rangle \mid \pi \in P,\ v \in V_{\mathrm{rhs}(\pi)},\ \varphi_{\mathrm{rhs}(\pi)}(v) = X\}$, and $\bar{\Delta} = \{\langle a, v, \pi\rangle \in \bar{\Sigma} \mid a \in \Delta\}$. The new labels will be used to give each node appearing in the right-hand side of a production a unique label. Then we can define the productions of $\bar{P}$ as follows. If $p = (X, D, B)$ is a production in P, and if $\langle X, v, \pi\rangle$ is a nonterminal in $\bar{\Sigma} - \bar{\Delta}$, then $\bar{P}$ contains the production $\bar{p} = (\langle X, v, \pi\rangle, \bar{D}, \bar{B})$. Here, $\bar{D} \in GR_{\bar{\Sigma}, \{*\}}$ and $\bar{B}$ are defined by $V_{\bar{D}} = V_D$, $E_{\bar{D}} = \{(x, *, y) \mid \exists \lambda \in \Gamma: (x, \lambda, y) \in E_D\}$, $\varphi_{\bar{D}}(x) = \langle \varphi_D(x), x, p\rangle$ for all $x \in V_D$, and $\bar{B} = \{(x, *, *, \langle a, w, \pi\rangle) \mid \langle a, w, \pi\rangle \in \bar{\Delta}, \exists \lambda \in \Gamma: (w, \lambda, v) \in E_{\mathrm{rhs}(\pi)}, (x, \lambda, *, a) \in B\}$. Moreover, if in production $\bar{p}$ above X equals S, then $\bar{P}$ contains the production $(S, \bar{D}, \bar{B})$, too.

First, it is necessary to demand that G is nonblocking, for an NCE grammar has just one edge label *, so it can not simulate G with respect to blocking edges. Second, in apex grammars, a terminal and a nonterminal node which are connected by an edge are always generated by the same production, and therefore $\bar{G}$ can use

their unique labels to decide whether a λ-edge connects them in G, as demonstrated in $\bar{B}$ above.

By defining the node relabeling ρ by $\rho(\langle a, v, \pi\rangle) = a$ for all $\langle a, v, \pi\rangle \in \bar{\Delta}$, we conclude the construction of this proof. It will not be difficult to check that the construction is correct. ∎

Hence, in the apex case, the one-final eNCE grammars and the NCE grammars with a relabeling have the same power. It will be shown next that this is not the case for (boundary, linear) e_1NCE grammars and (boundary, linear) NCE grammars with a relabeling. This is done by proving that there is a LIN-e_1NCE language that is not in R(NCE). First, however, we describe a large class of languages such that none of the languages in this class is in R(NCE). Recall the definition of edge complement from Section 5.

LEMMA 37. *Let L be an infinite language of graphs without edge labels such that* com(L) *is connected and of bounded degree. Then $L \notin R(NCE)$.*

Proof. Let $\bar{\Delta}$ be an alphabet such that $L \subseteq GR_{\bar{\Delta},\{*\}}$. We will suppose, to the contrary, that there are an NCE grammar $G = (\Sigma, \Delta, \{*\}, \{*\}, P, S)$ and a node relabeling $\rho: \Delta \to \bar{\Delta}$ such that $\rho(L(G)) = L$. From this a contradiction will be derived, by using a pumping argument. Since L is infinite, it can easily be argued that G has a recursive nonterminal (just as in the pumping theorem for context-free string languages, the pumping theorem for NLC graph languages in [JanRoz1], and the vertex pumping lemma in [Jef]; see also the proof of Theorem 2 of [EngLei2]). So, there exist a nonterminal $X \in \Sigma - \Delta$ and graphs H_1, H_2, and H_3 such that

(1) $S \overset{*}{\Rightarrow} H_1$,

(2) $X \overset{*}{\Rightarrow} H_2$,

(3) $X \overset{*}{\Rightarrow} H_3$,

(4) both H_1 and H_2 have a unique nonterminal node, labeled X, and all other nodes are terminal,

(5) H_2 has at least one terminal node, and

(6) H_3 has terminal nodes only.

Hence, there are infinitely many successful derivations, obtained by starting with derivation (1), applying derivation (2) any number of times, and ending with derivation (3). By point (5), all these derivations generate different terminal graphs. The contradiction will now be derived in two steps.

(i) If a terminal node x is generated during the ith application of (2), and a terminal node y is generated during the jth application of (2), where $i < j$, then x and y have to be connected to each other by an edge. Otherwise, since (a copy of) y is generated in the kth application of (2) for each $k > i$, and since (2) can

be repeated arbitrarily often, there can be arbitrarily many nodes that are not connected to x, contradicting the fact that com(L) is of bounded degree.

(ii) Since com(L) contains connected graphs only, there has to be, for each $i \geq 1$, a terminal node x_i generated during the ith application of (2) and a terminal node y_i not generated during the ith application of (2) such that y_i is not connected to x_i by an edge. From (i) it follows that this y_i has to be generated during (1) or (3). But since there can be arbitrarily many of these x_i's, and since H_1 and H_3 contain a fixed number of nodes, there has to be a node y in H_1 or H_3 that is not connected to arbitrarily many x_i's. Again, since com(L) is of bounded degree, this is a contradiction.

This proves the theorem. Note that we do not have to use the relabeling ρ at all, since the whole proof relies fully on the structure of the "underlying" graphs. ∎

Then we are now ready to show that N-e_1NCE, grammars have more power than NCE grammars with a relabeling. It is proved for boundary and linear grammars too, in one stroke.

THEOREM 38. *R(NCE) is a proper subset of N-e_1NCE, R(B-NCE) is a proper subset of B-e_1NCE, and R(LIN-NCE) is a proper subset of LIN-e_1NCE.*

Proof. The inclusions follow from the proof of Theorem 9. The fact that the inclusions are proper follows from Theorem 34 and Lemma 37. Consider any language L in LIN-e_1NCE which is connected and of bounded degree, e.g., the set $L = \text{FAT-CHAIN}(a^*, K)$, with $K(*) = \{aa\}$. From Lemma 37 it follows that com(L) (the graph language of Examples 6 and 28) is not an R(NCE) language. Since L is a linear language, it follows from Theorem 34 (or from Example 6) that com(L) $\in$ LIN-e_1NCE. This proves the theorem. ∎

Note that since the graph language L in the above proof is clearly in NCE, this shows, in particular, that NCE is not closed under com, as proved by a different method in [HofMai] (with the same counterexample).

REFERENCES

[Ber] J. BERSTEL, "Transductions and Context-free Languages," Teubner, Stuttgart, 1979.

[Bra] F. J. BRANDENBURG, On partially ordered graph-grammars, *in* [EhrNagRozRos], pp. 99–111.

[Cou] B. COURCELLE, An axiomatic definition of context-free rewriting and its application to NLC graph grammars, *Theoret. Comput. Sci.* **55** (1988), 141–182.

[EhrMaiRoz] A. EHRENFEUCHT, M. G. MAIN, AND G. ROZENBERG, Restrictions on NLC graph grammars, *Theoret. Comput. Sci.* **31** (1984), 211–223.

[EhrNagRoz] H. EHRIG, M. NAGL, AND G. ROZENBERG (Eds.), "Graph-Grammars and Their Application to Computer Science," Lecture Notes in Computer Science, Vol. 153, Springer-Verlag, Berlin, 1983.

[EhrNagRozRos] H. EHRIG, M. NAGL, G. ROZENBERG, AND A. ROSENFELD (Eds.), "Graph-Grammars and Their Application to Computer Science," Lecture Notes in Computer Science, Vol. 291, Springer-Verlag, Berlin, 1987.

[EngLei1] J. ENGELFRIET AND G. LEIH, Linear Graph Grammars: Power and Complexity, *Inform. and Computation* **81** (1989), 88–121.

[EngLei2] J. ENGELFRIET AND G. LEIH, Nonterminal bounded NLC graph grammars, *Theoret. Comput. Sci.* **59** (1988), 309–315.

[EngLei3] J. ENGELFRIET AND G. LEIH, "Complexity of Boundary Graph Languages," Report 88-07, Leiden, March 1988; *RAIRO*, in press.

[EngLeiRoz1] J. ENGELFRIET, G. LEIH, AND G. ROZENBERG, Apex graph grammars and attribute grammars, *Acta Inform.* **25** (1988), 537–571.

[EngLeiRoz2] J. ENGELFRIET, G. LEIH, AND G. ROZENBERG, Apex graph grammars, *in* [EhrNagRozRos], pp. 167–185.

[EngLeiWel] J. ENGELFRIET, G. LEIH, AND E. WELZL, "Boundary Graph Grammars with Dynamic Edge Relabeling," Report 87-30, Leiden, December 1987.

[EngRoz] J. ENGELFRIET AND G. ROZENBERG, "A Comparison of Boundary Graph Grammars and Context-free Hypergraph Grammars," Report 88-06, Leiden, February 1988.

[GécSte] F. GÉCSEG AND M. STEINBY, "Tree Automata," Akademiai Kiado, Budapest, 1984.

[HabKre] A. HABEL AND H.-J. KREOWSKI, Characteristics of graph languages generated by edge replacement, *Theoret. Comput. Sci.* **51** (1987), 81–115.

[Har] F. HARARY, "Graph Theory," Addison–Wesley, Reading, MA, 1969.

[HofMai] J. HOFFMANN AND M. G. MAIN, "Results on NLC Grammars with One-Letter Terminal Alphabets," Report CU-CS-348-86, Boulder, CO, Sept. 1986.

[Jan] D. JANSSENS, "Node Label Controlled Graph Grammars," Ph.D. thesis, University of Antwerp, 1983.

[JanRoz1] D. JANSSENS AND G. ROZENBERG, On the structure of node-label-controlled graph languages, *Inform. Sci.* **20** (1980), 191–216.

[JanRoz2] D. JANSSENS AND G. ROZENBERG, Restrictions, extensions, and variations of NLC grammars, *Inform. Sci.* **20** (1980), 217–244.

[JanRoz3] D. JANSSENS AND G. ROZENBERG, A characterization of context-free string languages by directed node-label controlled graph grammars, *Acta Inform.* **16** (1981), 63–85.

[JanRoz4] D. JANSSENS AND G. ROZENBERG, Graph grammars with node-label controlled rewriting and embedding, *in* [EhrNagRoz], pp. 186–205.

[JanRoz5] D. JANSSENS AND G. ROZENBERG, Graph gramars with neighbourhood-controlled embedding, *Theoret. Comput. Sci.* **21** (1982), 55–74.

[JanRozVer] D. JANSSENS, G. ROZENBERG, AND R. VERRAEDT, On sequential and parallel node-rewriting graph grammars, *Comput. Graphics Image Process.* **18** (1982), 279–304.

[Jef] J. JEFFS, Embedding rule independent theory of graph grammars, *in* [EhrNagRozRos], pp. 299–308.

[Kau1] M. KAUL, "Syntaxanalyse von Graphen bei Präzedenz-Graph-Grammatiken," dissertation, Universität Osnabrück, 1985.

[Kau2] M. KAUL, Practical applications of precedence graph grammars, *in* [EhrNagRozRos], pp. 326–342.

[Nag] M. NAGL, "Graph-grammatiken," Vieweg, Braunschweig, 1979.

[RosMil] A. ROSENFELD AND D. L. MILGRAM, Web automata and web grammars, *Mach. Intell.* **7** (1972), 307–324.

[RozWel1] G. ROZENBERG AND E. WELZL, Boundary NLC graph grammars—Basic definitions, normal forms, and complexity, *Inform. Control* **69** (1986), 136–167.

[RozWel2] G. ROZENBERG AND E. WELZL, Graph theoretic closure properties of the family of boundary NLC graph languages, *Acta Inform.* **23** (1986), 289–309.

[RozWel3] G. ROZENBERG AND E. WELZL, Combinatorial properties of boundary NLC graph languages, *Discrete Appl. Math.* **16** (1987), 58–73.

[Sal] A. SALOMAA, "Formal Languages," Academic Press, New York, 1973.

[Tha] J. W. THATCHER, Characterizing derivation trees of context-free grammars through a generalization of finite automata theory, *J. Comput. System Sci.* **1** (1967), 317–322.

[Wel1] E. WELZL, Encoding graphs by derivations and implications for the theory of graph grammars, *in* "Proceedings, 11th ICALP" (J. Paradaens, Ed.), Lecture Notes in Computer Science, Vol. 172, pp. 503–513, Springer-Verlag, Berlin, 1984.

[Wel2] E. WELZL, Boundary NLC and partition controlled graph grammars, *in* [EhrNagRozRos], pp. 593–609.