# Regular Description of Context-Free Graph Languages

Joost Engelfriet * and Vincent van Oostrom **

Department of Computer Science, Leiden University
P.O.Box 9512, 2300 RA Leiden, The Netherlands
email: engelfri@wi.leidenuniv.nl

**Abstract.** A set of (labeled) graphs can be defined by a regular tree language and one regular string language for each possible edge label, as follows. For each tree $t$ from the regular tree language the graph $gr(t)$ has the same nodes as $t$ (with the same labels), and there is an edge with label $\gamma$ from node $x$ to node $y$ if the string of labels of the nodes on the shortest path from $x$ to $y$ in $t$ belongs to the regular string language for $\gamma$. Slightly generalizing this definition scheme, we allow $gr(t)$ to have only those nodes of $t$ that have certain labels, and we allow a relabeling of these nodes. It is shown that in this way exactly the class of C-edNCE graph languages (generated by C-edNCE graph grammars) is obtained, one of the largest known classes of context-free graph languages.

## 1 Introduction

There are many kinds of context-free graph grammars (see, e.g., [ENRR, EKR]). Some are node rewriting and others are edge rewriting. In both cases a production of the grammar is of the form $X \to (D, C)$. Application of such a production to a labeled graph $H$ consists of removing a node (or edge) labeled $X$ from $H$, replacing it by the graph $D$, and connecting $D$ to the remainder of $H$ according to the embedding procedure $C$. Since these grammars are context-free in the sense that one node (or edge) is replaced, their derivations can be modeled by derivation trees, as in the case of context-free grammars for strings. However (in particular for certain types of node rewriting grammars), the grammar may still be context-sensitive in the sense that the (edges of the) graph generated according to the derivation tree may depend on the order in which the productions are applied. A graph grammar that does not suffer from this (quite disastrous) context-sensitivity, is said to be *confluent* (or to have the finite Church-Rosser property), see [Cou1] for a uniform treatment. Thus, for a confluent graph grammar $G$, each derivation tree of $G$ yields a unique graph

* The first author was supported by ESPRIT BRWG No.7183 COMPUGRAPH II.

** The present address of the second author is: Faculty of Mathematics and Computer Science, Vrije Universiteit, de Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands, email: oostrom@cs.vu.nl

in the graph language generated by $G$. Due to this close relationship to derivation trees, the generated graph language can be described in terms of a regular tree language (the set of derivation trees) and a finite number of regular string languages (to simulate the embedding procedure). We will show this for the particular case of the (node rewriting) *edNCE graph grammars*, studied in [Kau, Bra1, Bra2, ELR1, ELR2, Schu, ELW, EL1, EL2, ER1, CER]. Thus, we define the notion of a *regular path description* of a graph language (mainly determined by a regular tree language and a finite number of regular string languages) and prove that regular path descriptions have the same power as the confluent edNCE grammars (or C-edNCE grammars). The idea of using regular (tree and string) languages for the description of graphs was introduced in [Wel] and investigated in [ELW], for special cases of the C-edNCE grammar.

The structure of this paper is as follows. In Section 2 we define the edNCE grammar, and in particular the confluent edNCE grammar. In Section 3 we introduce the notion of a regular path description of a graph language, generalizing the regular path descriptions of [Wel, ELW]. In Sections 2 and 3 also some examples and some easy lemmas can be found. Section 4 contains the proof of the main result: the characterization of the C-edNCE graph languages by regular path descriptions. We use this result to show that the boundary edNCE grammars (or B-edNCE grammars, cf., e.g., [RW, ELW]) have less generating power than the C-edNCE grammars. In Section 5 we consider a number of special cases of the main result. In particular we define special types of regular path descriptions that characterize the boundary, apex, and linear edNCE graph languages. In Section 6 we investigate the string generating power of C-edNCE grammars: we view a graph grammar as a generator of all the strings that label directed paths in the generated graphs. We use the main result to show that the class of string languages generated by C-edNCE grammars in this way, equals the class of output languages of nondeterministic tree-walking transducers. This implies that this string generating method is more powerful than the one of [EH1] (that gives the output languages of deterministic tree-walking transducers).

The main result of this paper strengthens our belief that the class of C-edNCE graph languages (which seems to be the largest known class of graph languages that can be generated by context-free graph grammars, where 'context-free' is taken in the sense of [Cou1]) is a robust class of context-free graph languages: it can be characterized in several different ways. Other characterizations can be found in [CER] (by handle rewriting hypergraph grammars) and in [Oos, Eng2] (by monadic second order logic).

The results of this paper were established in 1988, and presented in [Oos] and in [Eng1]. The only added result is the characterization of apex edNCE languages (Theorem 26), which uses [EHL]. More recent work on the class of C-edNCE graph languages (or its subclasses) can be found in, e.g., [Bra3, Cou2, Cou3, Eng2, Eng4, SW1, SW2, KL]. For a survey, see [ER2].

The reader is assumed to be familiar with the basic concepts of formal language theory (see, e.g. [HU]), and of regular tree languages (see, e.g., [GS]).

## 2 Confluent edNCE Graph Grammars

In this subsection we give formal definitions for the edNCE graph grammars, and in particular for the confluent edNCE (C-edNCE) graph grammars. These grammars generate directed graphs with labeled nodes and labeled edges.

Let $\Sigma$ be an alphabet of node labels and $\Gamma$ an alphabet of edge labels. A *graph* over $\Sigma$ and $\Gamma$ is a tuple $H = (V, E, \lambda)$, where $V$ is the finite set of nodes, $E \subseteq \{(v, \gamma, w) \mid v, w \in V, v \neq w, \gamma \in \Gamma\}$ is the set of edges, and $\lambda : V \to \Sigma$ is the node labeling function. The components of $H$ are also denoted as $V_H$, $E_H$, and $\lambda_H$, respectively. Thus, we consider directed graphs without loops; multiple edges between the same pair of nodes are allowed, but they must have different labels. A graph is undirected if for every $(v, \gamma, w) \in E$, also $(w, \gamma, v) \in E$. Graphs with unlabeled nodes and/or edges can be modeled by taking $\Sigma$ and/or $\Gamma$ to be a singleton, respectively.

The set of all graphs over $\Sigma$ and $\Gamma$ is denoted $GR_{\Sigma,\Gamma}$. A subset of $GR_{\Sigma,\Gamma}$ is called a *graph language*.

As usual, two graphs $H$ and $K$ are *disjoint* if $V_H \cap V_K = \emptyset$. Also as usual, $H$ and $K$ are *isomorphic* if there is a bijection $f : V_H \to V_K$ such that $E_K = \{(f(v), \gamma, f(w)) \mid (v, \gamma, w) \in E_H\}$ and, for all $v \in V_H$, $\lambda_K(f(v)) = \lambda_H(v)$. The reader is assumed to be familiar with the way in which concrete graphs are used as representatives of abstract graphs, which are equivalence classes of concrete graphs with respect to isomorphism. We are usually interested in abstract graphs, but mostly discuss concrete ones. For instance, whereas a graph language is defined to be a set of concrete graphs, we usually view it as a set of abstract graphs.

After these preliminaries, we turn to the definition of edNCE graph grammar. The name of these grammars can be explained as follows. NCE stands for *neighbourhood controlled embedding*, the d stands for "directed graphs", and the e means that not only the nodes but also the edges of the graphs are labeled; in particular, the e stresses the fact that the edNCE grammar allows for *dynamic edge relabeling*. Thus, edNCE grammars are *graph grammars with neighbourhood controlled embedding and dynamic edge relabeling*. They were introduced in [Nag1, Nag2, Nag3] (as depth-1 context-free graph grammars), and studied in, e.g., [Kau, Bra1, Bra2, Schu]. They were also investigated as generalizations of NLC graph grammars in, e.g., [EL1, EL2, ELW].

**Definition 1.** An *edNCE grammar* is a tuple $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ where $\Sigma$ is the alphabet of node labels, $\Delta \subseteq \Sigma$ is the alphabet of terminal node labels, $\Gamma$ is the alphabet of edge labels, $\Omega \subseteq \Gamma$ is the alphabet of final edge labels, $P$ is the finite set of productions, and $S \in \Sigma - \Delta$ is the initial nonterminal. A production is of the form $X \to (D, C)$ with $X \in \Sigma - \Delta$, $D \in GR_{\Sigma,\Gamma}$, and $C \subseteq \Sigma \times \Gamma \times \Gamma \times V_D \times \{\text{in}, \text{out}\}$. □

Elements of $\Sigma - \Delta$ are called nonterminal node labels, and elements of $\Gamma - \Omega$ nonfinal edge labels. A node with a terminal or nonterminal label is said to be a terminal or nonterminal node, respectively, and similarly for final and nonfinal edges. For a production $p : X \to (D, C)$, $X$ is the left-hand side of

$p$, $D$ is the right-hand side of $p$, and $C$ is its connection relation. We write $\mathrm{lhs}(p) = X$, $\mathrm{rhs}(p) = D$, and $\mathrm{con}(p) = C$. Each element $(\sigma, \beta, \gamma, x, d)$ of $C$ (with $\sigma \in \Sigma$, $\beta, \gamma \in \Gamma$, $x \in V_H$, and $d \in \{\mathrm{in}, \mathrm{out}\}$) is a *connection instruction* of $p$. To improve readability, a connection instruction $(\sigma, \beta, \gamma, x, d)$ will always be written as $(\sigma, \beta/\gamma, x, d)$. In the literature the elements of a connection instruction are often listed in another order. Two productions $X_1 \to (D_1, C_1)$ and $X_2 \to (D_2, C_2)$ are called isomorphic if $X_1 = X_2$ and there is an isomorphism $f$ from $D_1$ to $D_2$ such that $C_2 = \{(\sigma, \beta/\gamma, f(x), d) \mid (\sigma, \beta/\gamma, x, d) \in C_1\}$. We will assume that $P$ does not contain distinct isomorphic productions. By $\mathrm{copy}(P)$ we denote the (infinite) set of all productions that are isomorphic to a production in $P$; an element of $\mathrm{copy}(P)$ will be called a *production copy* of $G$.

The process of rewriting in an edNCE grammar is defined through the application of productions (or rather, production copies), in the usual way. Informally, a rewriting step according to a production $p : X \to (D, C)$ consists of removing a node $v$ labeled $X$ (the "mother node") from the given "host"-graph $H$, substituting $D$ (the "daughter graph") in its place, and connecting $D$ to the remainder of $H$ in a way specified by the connection instructions in $C$. Together with $v$, all edges incident with $v$ are removed too. A connection instruction $(\sigma, \beta/\gamma, x, \mathrm{out})$ of $C$ means that if there was a $\beta$-labeled edge from the mother node $v$ to a node $w$ with label $\sigma$ in $H$, then the connecting process will establish a $\gamma$-labeled edge from $x$ to $w$. And similarly for 'in' instead of 'out', where 'in' refers to incoming edges of $v$ and 'out' to outgoing edges of $v$. Note in particular that the edge label is changed from $\beta$ into $\gamma$ (which explains the notation $\beta/\gamma$). The formal definition is as follows.

**Definition 2.** Let $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ be an edNCE grammar. Let $H$ and $H'$ be graphs in $GR_{\Sigma,\Gamma}$, let $v \in V_H$, and let $p = X \to (D, C)$ be a production copy of $G$ such that $D$ and $H$ are disjoint. Then we write $H \Rightarrow_{v,p} H'$, or just $H \Rightarrow_p H'$ or $H \Rightarrow H'$, if $\lambda_H(v) = X$ and $H'$ is the graph $(V, E, \lambda)$ in $GR_{\Sigma,\Gamma}$ such that

$$
\begin{aligned}
V &= (V_H - \{v\}) \cup V_D, \\
E &= \{(x, \gamma, y) \in E_H \mid x \neq v, y \neq v\} \cup E_D \\
&\quad \cup \{(w, \gamma, x) \mid \exists \beta \in \Gamma : (w, \beta, v) \in E_H, (\lambda_H(w), \beta/\gamma, x, \mathrm{in}) \in C\} \\
&\quad \cup \{(x, \gamma, w) \mid \exists \beta \in \Gamma : (v, \beta, w) \in E_H, (\lambda_H(w), \beta/\gamma, x, \mathrm{out}) \in C\}, \\
\lambda(x) &= \lambda_H(x) \text{ if } x \in V_H - \{v\}, \text{ and } \lambda(x) = \lambda_D(x) \text{ if } x \in V_D.
\end{aligned}
$$

$H \Rightarrow_{v,p} H'$ is called a *derivation step*, and a sequence of such derivation steps is called a *derivation*. A derivation

$$H_0 \Rightarrow_{v_1,p_1} H_1 \Rightarrow_{v_2,p_2} \cdots \Rightarrow_{v_n,p_n} H_n,$$

$n \geq 0$, is *creative* if the graphs $H_0$ and $\mathrm{rhs}(p_i)$, $1 \leq i \leq n$, are mutually disjoint. *We will restrict ourselves to creative derivations.* Thus, we write $H \Rightarrow^* H'$ if there is a *creative* derivation as above, with $H_0 = H$ and $H_n = H'$. Let $sn(S, z)$ denote the graph with a single $S$-labeled node $z$, and no edges. A *sentential form*

of $G$ is a graph $H$ such that $sn(S,z) \Rightarrow^* H$ for some $z$. The set of all sentential forms of $G$ is denoted $SF(G)$. The *graph language generated by* $G$ is

$$L(G) = \{H \in GR_{\Delta,\Omega} \mid sn(S,z) \Rightarrow^* H \text{ for some } z\}.$$

□

It is not difficult to show that if $H$ and $H'$ are isomorphic and $sn(S,z) \Rightarrow^* H$, then $sn(S,z') \Rightarrow^* H'$ for some $z'$. Thus, $L(G)$ is closed under taking isomorphic copies.

An edNCE grammar is *nonblocking* if $L(G) = \{H \in GR_{\Delta,\Gamma} \mid sn(S,z) \Rightarrow^* H \text{ for some } z\}$. This means that if a sentential form $H$ has terminal nodes only (i.e., cannot be rewritten any more), then all its edges are final. Note that we do not assume that edNCE grammars are nonblocking (as opposed to [Eng1, Eng2]).

*Example 1.* To draw a production $X \to (D,C)$ of an edNCE grammar, we draw the graph $D$ in the usual fashion, with nodes represented by dots and edges by arrows, and we add $C$ to $D$ in the following way: a connection instruction $(\sigma, \beta/\gamma, x, \text{in}) \in C$ is represented by a dashed arrow from a symbol $\sigma$ to (the dot representing) $x$, with label $\beta/\gamma$; for the connection instruction $(\sigma, \beta/\gamma, x, \text{out})$ the direction of the arrow is reversed.

(1) As a first example consider the edNCE grammar $G_1 = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ with $\Sigma = \{S, X, i, n, f\}$, $\Delta = \{i, n, f\}$, $\Gamma = \{\gamma, \rho, \lambda, \delta, \upsilon\}$, $\Omega = \{\rho, \lambda, \delta, \upsilon\}$, and $P$ consists of the three productions drawn in Fig. 1. Thus, production $p_3$ is $X \to (D,C)$ with $V_D = \{x, y\}$, $E_D = \{(x, \delta, y)\}$, $\lambda_D(x) = n$, $\lambda_D(y) = f$, and $C = \{(n, \gamma/\rho, x, \text{in}), (n, \gamma/\lambda, y, \text{out})\}$. $L(G_1)$ consists of all "ladders" of the form shown in Fig. 2 (with at least six nodes). Note that $\rho$, $\lambda$, $\delta$, $\upsilon$ intuitively stand for 'right', 'left', 'down', and 'up', respectively.

(2) As another example, consider the edNCE grammar $G_2 = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ with $\Sigma = \{S, X, n\}$, $\Delta = \{n\}$, $\Gamma = \Omega = \{\delta, \upsilon, \rho\}$, and $P$ consists of the three productions $p_a$, $p_b$, $p_c$ shown in Fig. 3. A dashed arrow from or to '$X, n$' represents two connection instructions, one for $X$ and one for $n$, in the obvious way. $G_2$ generates all rooted binary trees with $\delta$-labeled edges from each parent to its children, with additional $\upsilon$-labeled edges from each leaf to the root, and with additional $\rho$-labeled edges that chain the leaves of the tree. An example of such a "tree-like" graph is given in Fig. 4, except that the labels $a, b_l, b_r, c_l, c_r$ of all nodes should be replaced by $n$. Note that, in a derivation of $G_2$, the $\upsilon$-labeled edges to the root are created by production $p_a$, are "passed" from nonterminal node to nonterminal node by production $p_b$, and are finally attached to the leaves by production $p_c$.

Let $G_2'$ be the edNCE grammar that is obtained from $G_2$ by erasing all edges and connection instructions that involve $\rho$. It should be clear that $L(G_2')$ consists of all rooted binary trees with additional edges from the leaves to the root (i.e., all graphs of $L(G_2)$ without their $\rho$-labeled edges).

(3) As a third, and final example, consider the edNCE grammar $G_3 = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ with $\Sigma = \{S, n\}$, $\Delta = \{n\}$, $\Gamma = \Omega = \{\gamma\}$, and $P$ consists of the three productions $p_a$, $p_b$, $p_n$ shown in Fig. 5. An undirected edge between

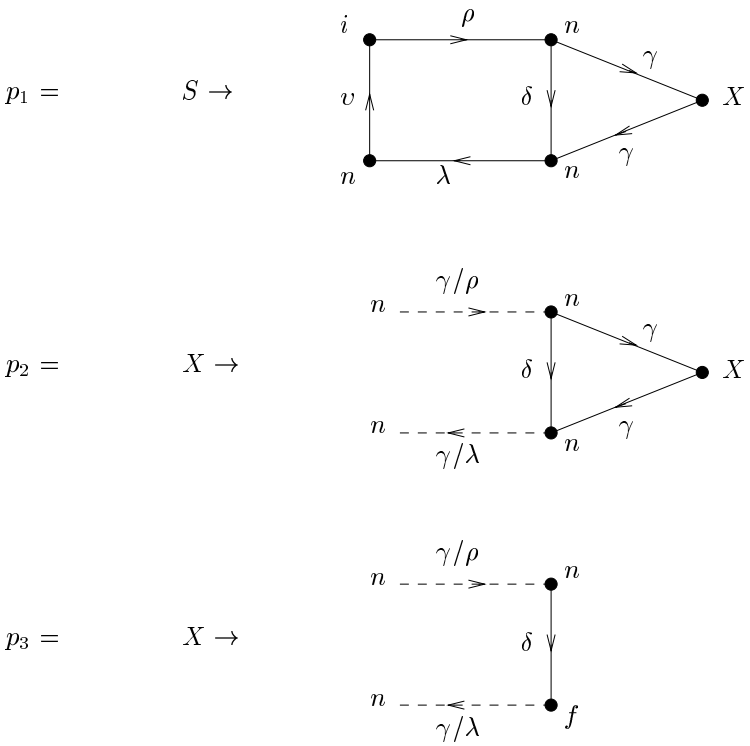


**Fig. 1.** Productions of $G_1$.

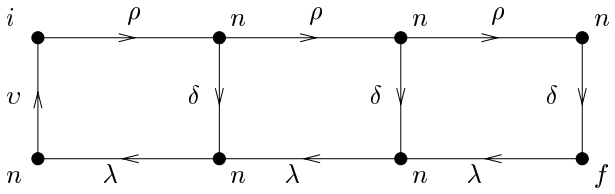


**Fig. 2.** A "ladder" generated by $G_1$.

$x$ and $y$ stands for two directed edges, one from $x$ to $y$ and one from $y$ to $x$ (and similarly for connection instructions). $L(G_3)$ is the set of all cographs (see [CLS]). It is the smallest set of (unlabeled, undirected) graphs that contains the one-node graph and is closed under the operations of join and disjoint union. The one-node graph corresponds to production $p_n$, the join operation (i.e., taking the disjoint union of two graphs, and joining every node of the one graph with every node of the other graph) corresponds to production $p_a$, and the operation of disjoint union corresponds to production $p_b$. An example of a cograph is the square: it is the join of two discrete graphs, each of which is the disjoint

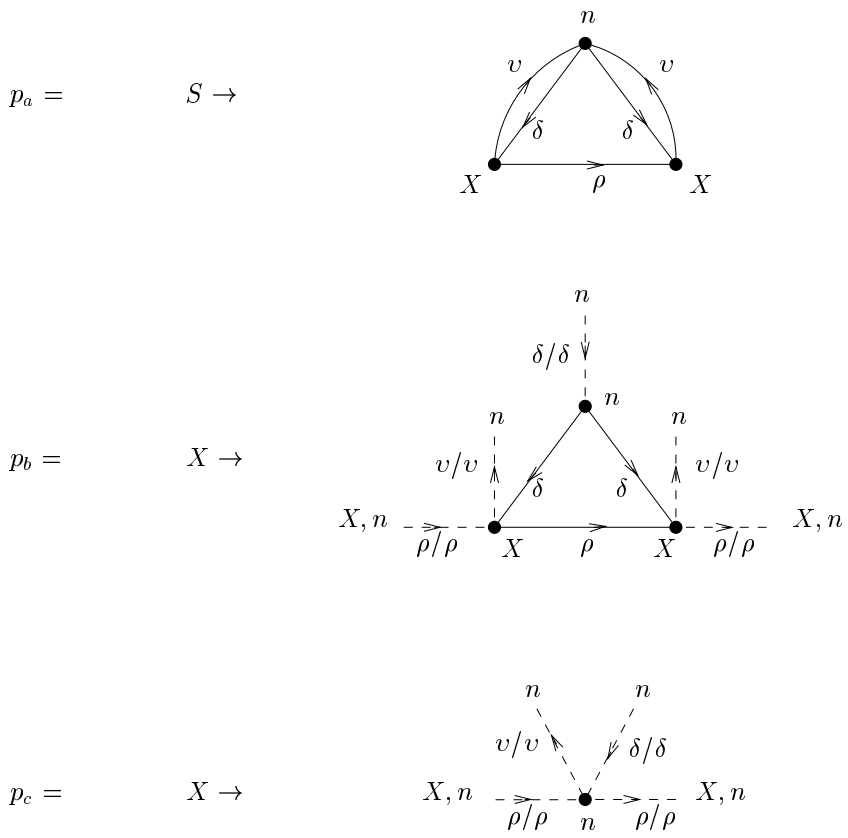


**Fig. 3.** Productions of $G_2$.

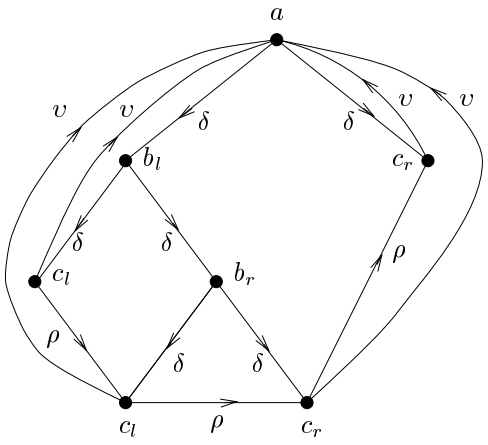


**Fig. 4.** A "tree-like" graph generated by $G_2$ (where all node labels should be $n$).

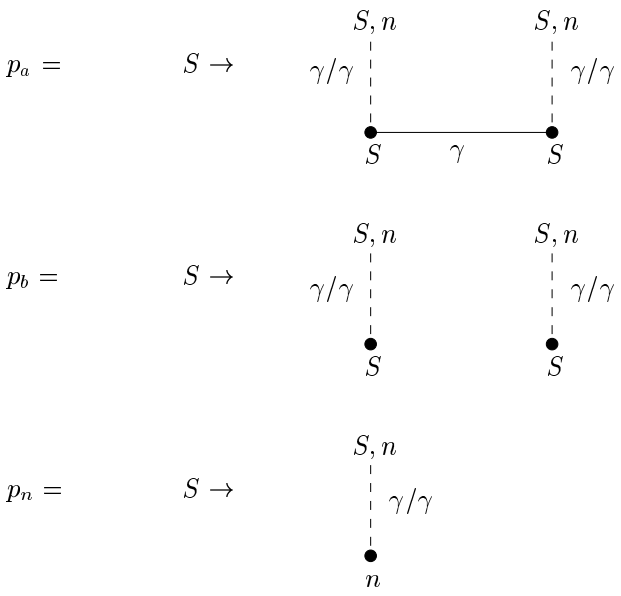


**Fig. 5.** Productions of $G_3$.

union of two one-node graphs. □

The edNCE grammar has certain undesirable non-context-free properties. This is caused by the fact that it *need not be confluent*, i.e., that the result of a derivation may depend on the order in which the productions are applied. This problem turns up in sentential forms that have edges between two nonterminal nodes, as in the following example.

*Example 2.* Consider an edNCE grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ with $\Sigma = \{S, A, B, a, b\}$, $\Delta = \{a, b\}$, $\Gamma = \{\alpha, \beta, \gamma, \gamma'\}$, $\Omega = \{\gamma, \gamma'\}$, and the following three productions $X \to (D, C)$:

$$
\begin{aligned}
p_S &: X = S, V_D = \{u, v\}, E_D = \{(u, \alpha, v)\}, \lambda_D(u) = A, \lambda_D(v) = B, \text{ and } C = \emptyset, \\
p_A &: X = A, V_D = \{x\}, E_D = \emptyset, \lambda_D(x) = a, \text{ and} \\
&\quad C = \{(B, \alpha/\beta, x, \text{out}), (b, \beta/\gamma', x, \text{out})\}, \\
p_B &: X = B, V_D = \{y\}, E_D = \emptyset, \lambda_D(y) = b, \text{ and} \\
&\quad C = \{(A, \alpha/\beta, y, \text{in}), (a, \beta/\gamma, y, \text{in})\}.
\end{aligned}
$$

The application of productions $p_S$, $p_A$, $p_B$ (in that order), gives a derivation $sn(S, z) \Rightarrow_{z,p_S} H \Rightarrow_{u,p_A} H_1 \Rightarrow_{v,p_B} H_{12}$, where $H$ is $\text{rhs}(p_S)$, and $H_{12}$ is the graph with two nodes $x$ and $y$, labeled $a$ and $b$, respectively, and one edge $(x, \gamma, y)$. However, interchanging the application of $p_A$ and $p_B$ to nodes $u$ and $v$, respectively, gives a derivation $sn(S, z) \Rightarrow_{z,p_S} H \Rightarrow_{v,p_B} H_2 \Rightarrow_{u,p_A} H_{21}$, where

$H_{21}$ is the same as $H_{12}$ except that its edge is $(x, \gamma', y)$. Since the order of application of productions results in two different graphs $H_{12}$ and $H_{21}$, the grammar is not confluent. □

In the literature it is customary to give a dynamic definition of confluence (see, e.g., [Kau, Schu, Bra2, Cou1, Eng1, Eng2, CER]. Here we propose a static one that can easily be checked on the embedding relations of the productions of the grammar (cf. Definition 5.1 of [EJKR] and Lemma 3.11 of [Cou1]).

**Definition 3.** An edNCE grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ is *confluent*, or a *C-edNCE* grammar, if for all productions $X_1 \to (D_1, C_1)$ and $X_2 \to (D_2, C_2)$ in $P$, all nodes $x_1 \in V_{D_1}$ and $x_2 \in V_{D_2}$, and all edge labels $\alpha, \gamma \in \Gamma$, the following equivalence holds:

$\exists \beta \in \Gamma : (X_2, \alpha/\beta, x_1, \text{out}) \in C_1$ and $(\lambda_{D_1}(x_1), \beta/\gamma, x_2, \text{in}) \in C_2$

$\Longleftrightarrow$

$\exists \beta \in \Gamma : (X_1, \alpha/\beta, x_2, \text{in}) \in C_2$ and $(\lambda_{D_2}(x_2), \beta/\gamma, x_1, \text{out}) \in C_1$. □

By C-edNCE we denote the class of graph languages generated by C-edNCE grammars.

All grammars discussed in Example 1 are C-edNCE grammars. Many other sets of graphs with "tree-like" graph theoretic properties can be defined by C-edNCE grammars. For example series-parallel graphs, transitive VSP graphs, complete bipartite graphs, (maximal) outerplanar graphs, edge complements of trees, and for fixed $k$, $k$-trees, graphs of treewidth $\leq k$, pathwidth $\leq k$, cutwidth $\leq k$, bandwidth $\leq k$, cyclic bandwidth $\leq k$, and topological bandwidth $\leq k$ (see, e.g., [RW, EL1]).

A symbolic picture of Definition 3 is given in Fig. 6. Intuitively the definition means that if the two productions are applied to a graph with a single edge $(v_1, \alpha, v_2)$, where $v_i$ is labeled $X_i$, then the same edges $(x_1, \gamma, x_2)$ are established between nodes of their right-hand sides, independent of the order in which the productions are applied. From this intuition the following characterization of confluence easily follows: the result of a derivation does not depend on the order in which the productions are applied.

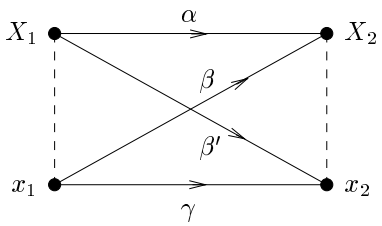


**Fig. 6.** Confluence.

**Proposition 4.** *An edNCE grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ is confluent if and only if the following holds for every graph $H \in GR_{\Sigma,\Gamma}$:*
*if $H \Rightarrow_{v_1,p_1} H_1 \Rightarrow_{v_2,p_2} H_{12}$ and $H \Rightarrow_{v_2,p_2} H_2 \Rightarrow_{v_1,p_1} H_{21}$ are (creative) derivations of $G$ with $v_1, v_2 \in V_H$ and $v_1 \neq v_2$, then $H_{12} = H_{21}$.*

The definition of confluence from the literature (where it is also called the finite Church-Rosser, or fCR, property) is exactly the same as the previous proposition, except that $H$ is restricted to be a sentential form of $G$; let us call this "dynamic confluence". Although there are more dynamically confluent than confluent edNCE grammars, it can be shown that they generate the same class C-edNCE of graph languages (see [ER2]). Although dynamic confluence is a decidable property of edNCE grammars (see [Kau]), the advantage of our notion of confluence is that it is completely static.

It is shown in [SW1] that for every C-edNCE grammar an equivalent nonblocking C-edNCE grammar can be constructed. This fact will not be used, but will be a consequence of our proofs (cf. the discussion after Theorem 21).

Several natural subclasses of the C-edNCE grammars have been investigated in the literature. We will consider three of them. Whenever we define an X-edNCE grammar for some X, X-edNCE will denote the class of graph languages generated by X-edNCE grammars.

An edNCE grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ is *boundary*, or a B-edNCE grammar, if, for every production $X \to (D, C)$, $D$ does not contain edges between nonterminal nodes, and $C$ does not contain connection instructions $(\sigma, \beta/\gamma, x, d)$ where $\sigma$ is nonterminal. Obviously, the second condition implies that boundary grammars are confluent. A B-edNCE grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ is *apex*, or an A-edNCE grammar, if for every production $X \to (D, C)$ and every connection instruction $(\sigma, \beta/\gamma, x, d) \in C$, $x$ and $\sigma$ are terminal. The boundary restriction on graph grammars was introduced in [RW]; the apex restriction was first considered in [ELR1]. An edNCE grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ is *linear*, or a LIN-edNCE grammar, if for every production $X \to (D, C)$, $D$ has at most one nonterminal node. It is easy to see that LIN-edNCE $\subseteq$ B-edNCE. It is shown in [EL1] that A-edNCE and LIN-edNCE are incomparable subclasses of B-edNCE. The class B-edNCE also contains the B-NLC languages of [RW], and (as shown, e.g., in [ER1]) the hyperedge replacement (HR) graph languages of [BC, HK, Hab].

Grammar $G_1$ from Example 1 is both linear and apex. Grammar $G_2'$ is boundary, but not linear or apex. Grammars $G_2$ and $G_3$ are not boundary.

The class A-edNCE can be characterized within the class C-edNCE: there is a simple condition on a graph language $L \in$ C-edNCE that expresses membership of $L$ in A-edNCE, viz. that $L$ is of bounded degree (i.e., that there is a number $k$ such that all nodes in all graphs of $L$ have degree at most $k$). This characterization was shown in [EHL] (see also [Eng3]).

**Proposition 5.** *For every graph language $L \in$ C-edNCE,*
*$L \in$ A-edNCE if and only if $L$ is of bounded degree.*

The class C-edNCE of graph languages generated by C-edNCE grammars has a

large number of nice closure properties. Here we need a very simple one: closure under edge relabeling. Let $\rho$ be an edge relabeling, i.e., a mapping $\rho : \Omega \to \Omega'$, where $\Omega$ and $\Omega'$ are edge label alphabets. For a graph $H \in GR_{\Delta,\Omega}$ we define $\rho(H) \in GR_{\Delta,\Omega'}$ to be the graph $(V_H, E, \lambda_H)$ with $E = \{(v, \rho(\gamma), w) \mid (v, \gamma, w) \in E_H\}$.

**Proposition 6.** *C-edNCE is closed under edge relabelings, i.e., if $\rho$ is an edge relabeling and $L \in$ C-edNCE, then $\rho(L) \in$ C-edNCE. The classes B-edNCE, A-edNCE, and LIN-edNCE are also closed under edge relabelings.*

*Proof.* Let $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ be a C-edNCE grammar, and let $\rho : \Omega \to \Omega'$ be an edge relabeling. It can be assumed that $\Gamma \cap \Omega' = \emptyset$.

For a graph $D \in GR_{\Sigma,\Gamma}$, $\rho(D) \in GR_{\Sigma,\Gamma\cup\Omega'}$ is defined to be the graph $(V_D, E, \lambda_D)$ such that

$$E = \{(v, \rho(\gamma), w) \mid (v, \gamma, w) \in E_D, \lambda_D(v) \in \Delta, \text{ and } \gamma \in \Omega, \text{ and } \lambda_D(w) \in \Delta\}$$
$$\cup \{(v, \gamma, w) \in E_D \mid \lambda_D(v) \in \Sigma - \Delta, \text{ or } \gamma \in \Gamma - \Omega, \text{ or } \lambda_D(w) \in \Sigma - \Delta\}.$$

Thus, only the final edges between terminal nodes are relabeled. Similarly, for a connection relation $C \subseteq \Sigma \times \Gamma \times \Gamma \times V_D \times \{\text{in}, \text{out}\}$, we define the connection relation $\rho(C)$ to be

$$\{(\sigma, \beta/\rho(\gamma), x, d) \mid (\sigma, \beta/\gamma, x, d) \in C, \lambda_D(x) \in \Delta, \text{ and } \gamma \in \Omega, \text{ and } \sigma \in \Delta\}$$
$$\cup \{(\sigma, \beta/\gamma, x, d) \in C \mid \lambda_D(x) \in \Sigma - \Delta, \text{ or } \gamma \in \Gamma - \Omega, \text{ or } \sigma \in \Sigma - \Delta\}.$$

We now construct the edNCE grammar $G' = (\Sigma, \Delta, \Gamma \cup \Omega', \Omega', P', S)$ with $P' = \{X \to (\rho(D), \rho(C)) \mid X \to (D, C) \text{ is in } P\}$. Using the confluence of $G$, it is easy to verify that $G'$ is still confluent (in Definition 3, first consider the case that $x_1$ and $x_2$ are terminal and $\gamma$ is final, and then the remaining case). Clearly, the boundary, apex, and linear properties are preserved. It is also easy to see that the derivations of $G'$ starting with $sn(S, z)$ are of the form $sn(S, z) \Rightarrow \rho(H_1) \Rightarrow \cdots \Rightarrow \rho(H_n)$ where $sn(S, z) \Rightarrow H_1 \Rightarrow \cdots \Rightarrow H_n$ is a derivation of $G$. Formally this can be proved by induction on $n$. It shows that $L(G') = \rho(L(G))$. □

## 3 Regular Path Descriptions

As observed before, all graphs generated by a C-edNCE grammar are "tree-like". An alternative, grammar independent, way of describing a set of "tree-like" graphs $H$ is by taking a tree $t$ from some regular tree language, defining the nodes of $H$ as a subset of the vertices of $t$, and defining an edge between nodes $u$ and $v$ of $H$ if the string of vertex labels on the shortest (undirected) path between $u$ and $v$ in $t$ belongs to some regular string language. Such a description of a graph language will be called a *regular path description*. This idea was introduced in [Wel], for linear graph grammars, and investigated for LIN-eNCE and B-eNCE grammars in [ELW] (where the missing d means that the generated graphs are undirected). Note that the nodes of trees will also be called 'vertices', in order not to confuse them with the nodes of the graphs involved.

Let us first define the regular tree languages. The following definition of a regular tree grammar as a special type of edNCE grammar, is equivalent (in generating power) with the usual one in tree language theory (see [GS]). The rooted, ordered trees from tree language theory will be viewed here (as usual) as a special type of graph: each vertex of the tree has a directed edge to each of its $k$ children, $k \geq 0$, and the order of the children is indicated by using the numbers $1, \ldots, k$ to label these edges.

As usual, a *ranked alphabet* is an alphabet $\Delta$ together with a mapping rank : $\Delta \to \{0, 1, 2, \ldots\}$. By $m_\Delta$ we denote the maximal number $\text{rank}(\sigma)$, $\sigma \in \Delta$.

**Definition 7.** An edNCE grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ is a *regular tree grammar*, or a REGT grammar, if $\Delta$ is a ranked alphabet, $\Gamma = \Omega = \{1, \ldots, m_\Delta\}$, and for every production $X \to (D, C)$ in $P$:
$V_D = \{x_0, x_1, \ldots, x_k\}$ for some $k \geq 0$,
$\lambda_D(x_0)$ is in $\Delta$ and has rank $k$, and, for every $1 \leq i \leq k$,
either $\lambda_D(x_i)$ is in $\Sigma - \Delta$ or $\lambda_D(x_i)$ is in $\Delta$ and has rank 0,
$E_D = \{(x_0, i, x_i) \mid 1 \leq i \leq k\}$, and
$C = \{(\sigma, i/i, x_0, \text{in}) \mid i \in \Gamma, \sigma \in \Delta\}$.

$G$ is *in normal form* if $\lambda_D(x_i) \in \Sigma - \Delta$ for all $1 \leq i \leq k$. □

Since $\Gamma$ and $\Omega$ are uniquely determined by $\Delta$, we will specify a REGT grammar as $(\Sigma, \Delta, P, S)$.

Thus, for a production $p$ of a regular tree grammar $G$, $\text{rhs}(p)$ consists of a (terminal) parent $x_0$, and $k$ (terminal or nonterminal) children $x_1, \ldots, x_k$. Edges lead from the parent to each child, and the children are ordered by numbering their edges from 1 to $k$. Such a production $X \to (D, C)$ can (modulo isomorphism) be denoted uniquely as $X \to \sigma\tau_1 \cdots \tau_k$ with $\sigma = \lambda_D(x_0)$ and $\tau_i = \lambda_D(x_i)$ for $1 \leq i \leq k$. Every sentential form of $G$ is a tree, of which all internal vertices are terminal; thus, only leaves are rewritten (into an internal vertex with $k$ children).

A graph language generated by a REGT grammar is a *regular tree language*. It is easy to see (and well known), that every regular tree language can be generated by a regular tree grammar in normal form. For a ranked alphabet $\Delta$, we denote by $T_\Delta$ the set of all trees over $\Delta$, i.e., the regular tree language generated by the REGT grammar with one nonterminal $S$ and all productions $S \to \sigma S^k$ for every $\sigma \in \Delta$, where $k = \text{rank}(\sigma)$.

It is obvious that every REGT grammar is a confluent edNCE grammar, because it is even an A-edNCE grammar. Hence every regular tree language is an A-edNCE graph language.

We now turn to the regular path description of graph languages. An essential concept to be used is the string of labels on the shortest (undirected) path from one vertex $u$ of a tree to another vertex $v$. Clearly, such a path first ascends from $u$ to the least common ancestor of $u$ and $v$, and then descends to $v$. In the string we indicate this change of direction by barring the label of the least common ancestor.

**Definition 8.** Let $\Sigma$ be a ranked alphabet, and let $\overline{\Sigma} = \{\overline{\sigma} \mid \sigma \in \Sigma\}$. For $t \in T_\Sigma$ and $u, v \in V_t$, we define $\mathrm{path}_t(u,v) \in \Sigma^*\overline{\Sigma}\Sigma^*$ as follows. Let $z \in V_t$ be the least common ancestor of $u$ and $v$ in $t$. Let $u_1, \ldots, u_m$ $(m \geq 1)$ and $v_1, \ldots, v_n$ $(n \geq 1)$ be the vertices on the directed paths in $t$ from $z$ to $u$ and from $z$ to $v$, respectively (thus, $z = u_1 = v_1$, $u = u_m$, and $v = v_n$). Then

$$\mathrm{path}_t(u,v) = \lambda_t(u_m) \cdots \lambda_t(u_2)\overline{\lambda_t(z)}\lambda_t(v_2) \cdots \lambda_t(v_n).$$

□

We are now ready for the definition of a regular path description.

**Definition 9.** A *regular path description* is a tuple $R = (\Sigma, \Delta, \Omega, T, h, W)$, where $\Sigma$ is a ranked alphabet, $\Delta$ and $\Omega$ are alphabets (of node and edge labels, respectively), $T \subseteq T_\Sigma$ is a regular tree language, $h$ is a partial function from $\Sigma$ to $\Delta$, and $W$ is a mapping from $\Omega$ to the class of regular string languages, such that, for every $\gamma \in \Omega$, $W(\gamma) \subseteq \Sigma^*\overline{\Sigma}\Sigma^*$.

The *graph language described by* $R$ is $L(R) = \{gr_R(t) \mid t \in T\}$, where $gr_R(t)$ is the graph $H \in GR_{\Delta,\Omega}$ such that
$V_H$ is the set of vertices $v$ of $t$ for which $\lambda_t(v)$ is in the domain of $h$,
$\lambda_H(v) = h(\lambda_t(v))$ for $v \in V_H$, and
$E_H$ is the set of all edges $(u, \gamma, v)$ with $\mathrm{path}_t(u,v) \in W(\gamma)$. □

Note that $L(R) \subseteq GR_{\Delta,\Omega}$. Note that $h$ is used both to determine which vertices of the tree $t$ are nodes of the graph $gr_R(t)$, and to define their labels in that graph (on the basis of their labels in the tree). Note that for each edge label $\gamma$, $W(\gamma)$ is the regular string language that defines the graph edges with label $\gamma$. Note finally that $L(R)$ is closed under taking isomorphic copies (because $T$ is).

Let RPD denote the class of graph languages that are described by regular path descriptions. The main result of this paper is that C-edNCE = RPD.

We now define some natural subclasses X-RPD of RPD, by restricting the regular path descriptions to be of type X.

Let B-RPD be the subclass of RPD obtained by restricting every $W(\gamma)$ to be a subset of $\Sigma^*\overline{\Sigma} \cup \overline{\Sigma}\Sigma^*$. This means, for a regular path description of type B, that graph edges are only established between tree vertices of which one is a descendant of the other. We will show that B-edNCE = B-RPD (essentially the same result is shown in Theorem 31 of [ELW] for undirected graphs).

Let A-RPD be the subclass of RPD obtained by restricting every $W(\gamma)$ to be finite. Thus, for a regular path description of type A, graph edges can only be established between tree vertices that are at a bounded distance from each other. We will show that A-edNCE = A-RPD.

Let LIN-RPD be the subclass of RPD obtained by restricting the symbols of the ranked alphabet $\Sigma$ to have rank 1 or 0. This means that the trees in the regular tree language are in fact strings. Thus, a regular path description of type LIN uses regular string languages only. Note that, obviously, LIN-RPD $\subseteq$ B-RPD. We will show that LIN-edNCE = LIN-RPD (as for B, essentially the same result for undirected graphs is shown in Theorem 32 of [ELW]).

*Example 3.* We will give regular path descriptions of the graph languages generated by the C-edNCE grammars of Example 1.

(1) The language $L(G_1)$ of all "ladders" can be described by the regular path description $R_1 = (\Sigma, \Delta, \Omega, T, h, W)$ of type LIN, with $\Sigma = \{i, n, a, f\}$, $\mathrm{rank}(i) = \mathrm{rank}(n) = \mathrm{rank}(a) = 1$, $\mathrm{rank}(f) = 0$, $\Delta = \{i, n, f\}$, $\Omega = \{\rho, \lambda, \delta, \upsilon\}$, $T = iana(na)^*nf$ (where we have written the trees as strings in prefix notation), $h$ is the total function with $h(i) = i$, $h(n) = h(a) = n$, and $h(f) = f$, and $W$ is given by $W(\rho) = \{\overline{i}an, \overline{n}an\}$, $W(\delta) = \{\overline{n}a, \overline{n}f\}$, $W(\lambda) = \{fn\overline{a}, an\overline{a}\}$, and $W(\upsilon) = \{a\overline{i}\}$.

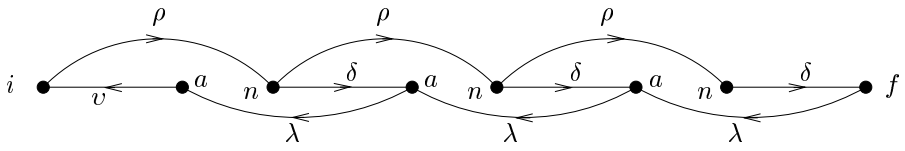


**Fig. 7.** Regular path description of a "ladder".

Note that $R_1$ is not only of type LIN, but also of type B and of type A. To see how $R_1$ works, see Fig. 7. Assuming that $a = n$, this figure represents the graph $gr_{R_1}(t)$ of the tree $t = ianananf$, which is in fact the "ladder" of Fig. 2. The tree $t$ itself is not drawn, but is suggested as a horizontal chain (with the root at the left, and the leaf at the right).

(2) The graph language $L(G_2)$ of all binary trees with additional edges is described by the regular path description $R_2 = (\Sigma, \Delta, \Omega, T, h, W)$ with $\Sigma = \{a, b_l, b_r, c_l, c_r\}$, $\mathrm{rank}(a) = \mathrm{rank}(b_l) = \mathrm{rank}(b_r) = 2$, $\mathrm{rank}(c_l) = \mathrm{rank}(c_r) = 0$, $\Delta = \{n\}$, $\Omega = \{\delta, \upsilon, \rho\}$, $T = L(G)$, where $G$ is the regular tree grammar with productions $S \to aLR$, $L \to b_lLR$, $R \to b_rLR$, $L \to c_l$, and $R \to c_r$ (with nonterminals $S, L, R$ of which $S$ is the initial one), $h$ is the total function with $h(\sigma) = n$ for all $\sigma \in \Sigma$, and $W$ is defined by $W(\delta) = (\overline{a} \cup \overline{b_l} \cup \overline{b_r})(b_l \cup b_r \cup c_l \cup c_r)$, $W(\upsilon) = (c_l \cup c_r)(b_l \cup b_r)^*\overline{a}$, and $W(\rho) = (c_rb_r^*b_l \cup c_l)(\overline{a} \cup \overline{b_l} \cup \overline{b_r})(b_rb_l^*c_l \cup c_r)$.

To see how $R_2$ works, see Fig. 4. Taking all node labels to be $n$, the figure represents the graph $gr_{R_2}(t)$ of the tree $t = ab_lc_lb_rc_lc_rc_r$ (in prefix notation). The tree $t$ itself is not drawn, but equals the tree spanned by the $\delta$-labeled edges (apart from the edge labels, which should be 1 or 2).

Note that $R_2$ is *not* of type B, A, or LIN. Removing $\rho$ from $\Omega$ (and $W(\rho)$ from $W$), a regular path description $R_2'$ of $L(G_2')$ is obtained that is of type B (but not of type A or LIN).

(3) The graph language $L(G_3)$ of all cographs is described by the regular path description $R_3 = (\Sigma, \Delta, \Omega, T, h, W)$ with $\Sigma = \{a, b, n\}$, $\mathrm{rank}(a) = \mathrm{rank}(b) = 2$ and $\mathrm{rank}(n) = 0$, $\Delta = \{n\}$, $\Omega = \{\gamma\}$, $T = T_\Sigma$, $h(n) = n$, $h(a)$ and $h(b)$ are undefined, and $W(\gamma) = n(a \cup b)^*\overline{a}(a \cup b)^*n$.

To see how $R_3$ works, see Fig. 8. Disregarding the dotted lines, the figure represents the tree $t = abnnbnn$ (in prefix notation), generated by $G$. The nodes

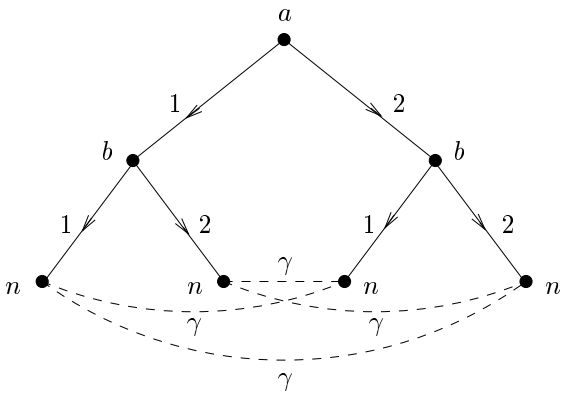


**Fig. 8.** Regular path description of a cograph.

with label $n$, together with the dotted (undirected) edges, form the cograph $gr_{R_3}(t)$ (which is the square). By the definition of $W(\gamma)$, two leaves of $t$ are connected by an edge in $gr(t)$ if their least common ancestor is labeled $a$. This means that $R_3$ formalizes the usual cotree representation of cographs (see [CLS]). In fact, the (co)trees in $L(G)$ can be viewed as expressions, with $a$ representing the join operation, $b$ disjoint union, and $n$ the one-node graph. □

We need the following easy closure property of RPD.

**Lemma 10.** *Let $L$ be a graph language in RPD with $L \subseteq GR_{\Delta,\Omega}$.*
*Then $L \cap GR_{\Delta',\Omega'}$ is in RPD for all $\Delta' \subseteq \Delta$ and $\Omega' \subseteq \Omega$.*
*A similar statement holds for B-RPD, A-RPD, and LIN-RPD.*

*Proof.* Let $L = L(R)$ for the regular path description $R = (\Sigma, \Delta, \Omega, T, h, W)$. Define $T' = \{t \in T_\Sigma \mid gr_R(t) \in GR_{\Delta',\Omega'}\}$. Obviously, $L \cap GR_{\Delta',\Omega'} = L(R')$ where $R' = (\Sigma, \Delta, \Omega, T \cap T', h, W)$. Since the regular tree languages are closed under intersection, it suffices to show that $T'$ is regular. Now $T' = T_1 \cap T_2$ where $T_1 = \{t \in T_\Sigma \mid gr_R(t) \in GR_{\Delta',\Omega}\}$ and $T_2 = \{t \in T_\Sigma \mid gr_R(t) \in GR_{\Delta,\Omega'}\}$. Clearly, $T_1$ is the regular tree language $T_{\Sigma'}$ where $\Sigma' = \{\sigma \in \Sigma \mid h(\sigma)$ is undefined or $h(\sigma) \in \Delta'\}$. Also, $T_2 = T_\Sigma - \bigcup_{\gamma \in \Omega - \Omega'} T_\gamma$ where $T_\gamma$ is the set of all $t \in T_\Sigma$ such that $gr_R(t)$ has at least one edge with label $\gamma$. Since the regular tree languages are closed under union and complement, it suffices to show that $T_\gamma$ is regular for every $\gamma$. Clearly, $T_\gamma$ is the set of all $t \in T_\Sigma$ such that there exist $u, v \in V_t$ with $\mathrm{path}_t(u, v) \in W(\gamma)$. Since $W(\gamma)$ is a regular string language, there is a finite automaton $A$ that accepts $W(\gamma)$. It is not difficult to write a regular tree grammar $G$ that generates $T_\gamma$. $G$ decides nondeterministically that it is generating the least common ancestor $z$ of some $u$ and $v$, guesses a state of $A$, and then simulates the behaviour of $A$ on the path from $z$ to $v$ (where it should arrive in a final state of $A$) and simulates the behaviour of $A$ backwards on the path from $z$ to $u$ (where it should arrive in an initial state of $A$). We leave the details of $G$ as an exercise to the reader. □

## 4 The Main Result

In this section we prove the main result: RPD = C-edNCE. We start with the inclusion of RPD in C-edNCE. We first consider a different, but strongly related, kind of graph description. The edge labels of the graphs are restricted to be pairs of states of a finite automaton.

It is convenient to consider finite automata that are deterministic, except that they have an arbitrary number of initial states. A *finite automaton* is a tuple $A = (Q, \Sigma, \delta, I, F)$ where $Q$ is the finite set of states, $\Sigma$ is the input alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, $I \subseteq Q$ is the set of initial states, and $F \subseteq Q$ is the set of final states. The transition function is extended in the usual way to a mapping $\delta : Q \times \Sigma^* \to Q$, and the language recognized by $A$ is $L(A) = \{w \in \Sigma^* \mid \exists i \in I : \delta(i, w) \in F\}$.

**Definition 11.** An *automaton path description* is a tuple $R = (\Sigma, \Delta, T, h, A)$ where $\Sigma$ is a ranked alphabet, $\Delta$ is an alphabet, $T$ is a regular tree language in $T_\Sigma$, $h$ is a partial function from $\Sigma$ to $\Delta$, and $A = (Q, \Sigma \cup \overline{\Sigma}, \delta, I, F)$ is a finite automaton with $L(A) \subseteq \Sigma^* \overline{\Sigma} \Sigma^*$. The *graph language described by* $R$ is $L(R) = \{gr_R(t) \mid t \in T\}$, where $gr_R(t)$ is the graph $H \in GR_{\Delta, I \times F}$ such that
$V_H$ is the set of vertices $v$ of $t$ for which $\lambda_t(v)$ is in the domain of $h$,
$\lambda_H(v) = h(\lambda_t(v))$ for $v \in V_H$, and
$E_H$ is the set of all edges $(u, \langle q_1, q_2 \rangle, v)$ such that $\delta(q_1, \mathrm{path}_t(u, v)) = q_2$, $q_1 \in I$, and $q_2 \in F$. □

**Lemma 12.** *If $R$ is an automaton path description, then $L(R) \in$ C-edNCE.*

*Proof.* Let $R = (\Sigma, \Delta, T, h, A)$ and $A = (Q, \Sigma \cup \overline{\Sigma}, \delta, I, F)$. Moreover, let $G = (N \cup \Sigma, \Sigma, P, S)$ be a regular tree grammar in normal form generating $T$ (where $N$, disjoint with $\Sigma$, is the set of nonterminals of $G$). We construct a C-edNCE grammar $G' = (N \cup \Delta, \Delta, Q \times Q, I \times F, P', S)$ such that $L(G') = L(R)$. The idea is that $G'$ simulates $G$, generating the appropriate vertices of the tree $t$ generated by $G$. To generate the edges of $gr_R(t)$, $G'$ simulates the behaviour of $A$ in its edge labels (through the use of the dynamic edge relabeling feature of edNCE grammars). To this aim, $G'$ also keeps edges between nonterminal vertices $u$ and $v$ of a sentential form $s$ generated by $G$, viz. all edges $(u, \langle q_1, q_2 \rangle, v)$ such that $\delta(q_1, \mathrm{path}'_s(u, v)) = q_2$, where $\mathrm{path}'_s(u, v)$ is obtained from $\mathrm{path}_s(u, v)$ by erasing the labels of $u$ and $v$ (note that all nonterminal vertices are leaves of $s$). Similarly, $G'$ keeps edges from each nonterminal vertex $u$ to each terminal vertex $v$ (with $q_2 \in F$, and only the label of $u$ erased) and from each terminal vertex $u$ to each nonterminal vertex $v$ (with $q_1 \in I$, and only the label of $v$ erased).

The set of productions $P'$ is defined as follows. Let $p = X \to \sigma X_1 \cdots X_k$ be a production of $G$, with $V_{\mathrm{rhs}(p)} = \{x_0, x_1, \ldots, x_k\}$, $\lambda_{\mathrm{rhs}(p)}(x_0) = \sigma \in \Sigma$, and $\lambda_{\mathrm{rhs}(p)}(x_i) = X_i \in N$ for $1 \leq i \leq k$. With this production $p \in P$ we associate one production $p' \in P'$. We first consider the case that $h(\sigma)$ is defined. Then $p' = X \to (D, C)$, where

$$V_D = \{x_0, x_1, \ldots, x_k\},$$

$$
\begin{aligned}
E_D &= \{(x_i, \langle q_1, q_2\rangle, x_j) \mid i,j \neq 0, \delta(q_1, \overline{\sigma}) = q_2\} \\
&\quad \cup \{(x_0, \langle q_1, q_2\rangle, x_i) \mid i \neq 0, q_1 \in I, \delta(q_1, \overline{\sigma}) = q_2\} \\
&\quad \cup \{(x_i, \langle q_1, q_2\rangle, x_0) \mid i \neq 0, q_2 \in F, \delta(q_1, \overline{\sigma}) = q_2\} \\
\lambda_D(x_0) &= h(\sigma), \text{ and } \lambda_D(x_i) = X_i \text{ for } 1 \leq i \leq k, \\
C &= \{(\tau, \langle q_1, q_2\rangle / \langle q_1', q_2\rangle, x_i, \text{out}) \mid \delta(q_1', \sigma) = q_1, i \neq 0\} \\
&\quad \cup \{(\tau, \langle q_1, q_2\rangle / \langle q_1', q_2\rangle, x_0, \text{out}) \mid \delta(q_1', \sigma) = q_1, q_1' \in I\} \\
&\quad \cup \{(\tau, \langle q_1, q_2\rangle / \langle q_1, q_2'\rangle, x_i, \text{in}) \mid \delta(q_2, \sigma) = q_2', i \neq 0\} \\
&\quad \cup \{(\tau, \langle q_1, q_2\rangle / \langle q_1, q_2'\rangle, x_0, \text{in}) \mid \delta(q_2, \sigma) = q', q_2' \in F\},
\end{aligned}
$$

where we have used $\tau$ to denote an arbitrary element of $N \cup \Delta$. In the case that $h(\sigma)$ is undefined, the above definition of $p'$ should be changed in the obvious way by dropping $x_0$ from $V_D$, and restricting all other components accordingly (such that only the first part of $E_D$, the second part of $\lambda_D$, and the first and third parts of $C$ remain).

This ends the definition of $G'$. It is straightforward to verify that $G'$ is confluent. Intuitively, the reason is that every edge in a sentential form $s$ of $G'$ is of the form $(u, \langle q_1, q_2\rangle, v)$. Moreover, if $u$ is rewritten, then the edge label changes into $\langle q_1', q_2\rangle$ for some $q_1'$, and the connection instructions do not inspect $q_2$ or $\lambda_s(v)$. Similarly, if $v$ is rewritten, it changes into $\langle q_1, q_2'\rangle$ for some $q_2'$, and $q_1$ and $\lambda_s(u)$ are not inspected. As a consequence, if both $u$ and $v$ are rewritten, the edge label will be $\langle q_1', q_2'\rangle$ independent of the order of rewriting. In fact, this idea is formalized in Lemma 10 of [ER2].

To show the correctness of the construction, we extend the definition of $gr_R$ to sentential forms of $G$. Let $s$ be a sentential form of $G$. Note that $s$ is a tree (with the nonterminals having rank 0). We define $gr_R(s)$ to be the graph $H \in GR_{N\cup\Delta, Q\times Q}$ such that
$V_H$ is the union of the set of nonterminal vertices of $s$ and the set of terminal vertices $v$ of $s$ for which $\lambda_s(v)$ is in the domain of $h$,
$\lambda_H(v) = \lambda_s(v)$ for a nonterminal vertex $v$, $\lambda_H(v) = h(\lambda_s(v))$ for a terminal vertex $v$, and
$E_H$ is the set of all edges $(u, \langle q_1, q_2\rangle, v)$ such that $\delta(q_1, \text{path}_s'(u,v)) = q_2$ and: if $u$ is terminal then $q_1 \in I$, and if $v$ is terminal then $q_2 \in F$ (where $\text{path}_s'(u,v)$ is obtained from $\text{path}_s(u,v)$ by erasing all elements of $N$).

It can now be shown (by induction on the length of the derivations) that for every graph $H \in GR_{N\cup\Delta, Q\times Q}$, $sn(S,z) \Rightarrow_{G'}^* H$ if and only if there exists $s \in T_{N\cup\Sigma}$ such that $sn(S,z) \Rightarrow_G^* s$ and $H = gr_R(s)$. Since $gr_R(s) \in GR_{\Delta, I\times F}$ iff $s \in T_\Sigma$, this implies that $L(G') = L(R)$. This proves the lemma.

Note that if $gr_R(s) \in GR_{\Delta, Q\times Q}$, then $s \in T_\Sigma$ and hence $gr_R(s) \in GR_{\Delta, I\times F}$. This shows that $G'$ is nonblocking (cf. the definition of the nonblocking property, just after Definition 2). □

Having done most of the work, we now show that RPD is included in C-edNCE.

**Lemma 13.** *RPD $\subseteq$ C-edNCE.*

*Proof.* Let $R = (\Sigma, \Delta, \Omega, T, h, W)$ be a regular path description. Since, for every $\gamma \in \Omega$, $W(\gamma)$ is regular, there is a finite automaton with one initial state that recognizes $W(\gamma)$. By putting all these automata (disjointly) together, it should be clear that there exists a finite automaton $A = (Q, \Sigma \cup \overline{\Sigma}, \delta, I, F)$ such that $I = \Omega$ and, for every $\gamma \in \Omega$, $W(\gamma) = \{w \in (\Sigma \cup \overline{\Sigma})^* \mid \delta(\gamma, w) \in F\}$. Now consider the automaton path description $R' = (\Sigma, \Delta, T, h, A)$, and let $\rho : I \times F \to \Omega$ be the edge relabeling such that $\rho(q_1, q_2) = q_1$. Obviously, $L(R) = \rho(L(R'))$. Hence, by Lemma 12 and Proposition 6, $L(R)$ is in C-edNCE. This proves the lemma.

It is easy to see that the construction in the proof of Proposition 6 preserves the nonblocking property of the edNCE grammars. Together with the remark at the end of the proof of Lemma 12, this shows that $L(R)$ can be generated by a nonblocking C-edNCE grammar. □

To prove that C-edNCE $\subseteq$ RPD, we will first develop a few technical tools. First of all, in order to find a regular path description for a given C-edNCE grammar, it is convenient to assume that all nodes of a right-hand side of a production have distinct labels. It is easy to do this for the nonterminal nodes; for the terminal nodes a node relabeling is needed to re-establish the original labels. Let $\rho$ be a node relabeling, i.e., a mapping $\Delta \to \Delta'$, where $\Delta$ and $\Delta'$ are node label alphabets. For a graph $H \in GR_{\Delta,\Omega}$ we define $\rho(H) \in GR_{\Delta',\Omega}$ to be the graph $(V_H, E_H, \lambda)$ where $\lambda(v) = \rho(\lambda_H(v))$ for every $v \in V_H$. Let us say that a graph $D$ is *uniquely labeled* if for all $u, v \in V_D$, if $u \neq v$ then $\lambda_D(u) \neq \lambda_D(v)$. And let us say that a C-edNCE grammar is uniquely labeled if all right-hand sides of its productions are uniquely labeled.

**Lemma 14.** *For every C-edNCE grammar $G$ one can construct a uniquely labeled C-edNCE grammar $G'$ and a node relabeling $\rho$ such that $L(G) = \rho(L(G'))$. The same is true for B-edNCE, A-edNCE, and LIN-edNCE grammars.*

*Proof.* Let $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$, and let $m$ be an upper bound on the number of nodes with the same label in the right-hand sides of the productions in $P$. Let $\Sigma'$ be any alphabet and let $\rho$ be any surjective mapping $\Sigma' \to \Sigma$ such that $\#\rho^{-1}(\sigma) = m$ for every $\sigma \in \Sigma$ (where $\#A$ is the cardinality of a finite set $A$). Then we construct $G' = (\Sigma', \Delta', \Gamma, \Omega, P', S')$ where $\Delta' = \rho^{-1}(\Delta)$, $S'$ is any element of $\rho^{-1}(S)$, and $P'$ is defined as follows. Let $p = X \to (D, C)$ be in $P$. Construct a uniquely labeled graph $D' \in GR_{\Sigma',\Gamma}$ such that $\rho(D') = D$. Define $C' = \{(\sigma', \beta/\gamma, x, d) \mid \sigma' \in \Sigma', (\rho(\sigma'), \beta/\gamma, x, d) \in C\}$. Then, corresponding to $p$, $P'$ contains the productions $X' \to (D', C')$ for every $X' \in \rho^{-1}(X)$.

It can easily be verified that $G'$ is confluent, and that the construction preserves the boundary, apex, and linear properties. It should also be clear that $\rho'(L(G')) = L(G)$ (where $\rho'$ is the restriction of $\rho$ to $\Delta'$). □

In the above lemma, the node relabeling is not really needed for the case of arbitrary C-edNCE grammars. In fact, the grammar can first be transformed into an equivalent one in which each right-hand side of a production contains at most one terminal node (see, e.g., [Oos, ER2]).

The following lemma is obvious: it suffices to compose the function $h$ of the regular path description with the node relabeling $\rho$.

**Lemma 15.** *RPD is closed under node relabelings. The same holds for B-RPD, A-RPD, and LIN-RPD.*

Thus, in the remainder of this section it suffices to consider uniquely labeled C-edNCE grammars.

The main idea in the construction of a regular path description $R$ for a C-edNCE grammar $G$ is to use the set of derivation trees of $G$ as the regular tree language of $R$. In our case it is convenient to define derivation trees in such a way that the internal vertices are labeled by productions of $G$, whereas the leaves are labeled by the node labels of $G$. Intuitively, if $H$ is generated by $G$ according to a derivation tree $t$, the productions that are used in the generation are on the labels of the internal vertices of $t$, whereas the leaves of $t$ represent the nodes of $H$. The edges between two nodes $u$ and $v$ of $H$ can then be determined by verifying that $\mathrm{path}_t(u, v)$ belongs to a certain regular language. Note that $\mathrm{path}_t(u, v)$ consists of the two sequences of productions that are used to generate $u$ and $v$, and of the labels of $u$ and $v$ (which determine $u$ and $v$ in the right-hand sides of the productions that are applied last). It suffices to know this information in order to simulate the connection instructions that build the edges between $u$ and $v$ (and this simulation can be done by a finite automaton).

We first define a regular tree grammar that generates the derivation trees of a C-edNCE grammar.

**Definition 16.** Let $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ be a uniquely labeled C-edNCE grammar. We will view the elements of $P$ and $\Sigma$ also as symbols of a ranked alphabet, where the rank of a production $p$ is the number of nodes of $\mathrm{rhs}(p)$, and the rank of every $\sigma \in \Sigma$ is 0. The *derivation tree grammar of* $G$ is the regular tree grammar $G' = (\Sigma', \Delta', P', S')$, where $\Sigma' = P \cup \Sigma$, $\Delta' = P \cup \Delta$, $S' = S$, and $P'$ consists of all productions $X \to p\sigma_1 \cdots \sigma_k$ with $p \in P$, $X = \mathrm{lhs}(p)$, $k = \#V_{\mathrm{rhs}(p)}$, and $\{\sigma_1, \ldots, \sigma_k\} = \{\lambda_{\mathrm{rhs}(p)}(v) \mid v \in V_{\mathrm{rhs}(p)}\}$. The order of the node labels of $\mathrm{rhs}(p)$ in this production of $P'$ is arbitrary (but fixed). Note that all $\sigma_i$ are distinct, because $\mathrm{rhs}(p)$ is uniquely labeled. □

We now define the set of all possible labels of paths from a vertex to a leaf in a derivation tree of $G$.

**Definition 17.** Let $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ be a uniquely labeled C-edNCE grammar. The set $PS(G)$ of *path strings of* $G$ is defined to be the set of all strings $p_1 \cdots p_n\sigma \in P^*\Sigma$ such that $n \geq 0$, $p_i \in P$ for $1 \leq i \leq n$, $\sigma \in \Sigma$, for every $1 \leq i \leq n-1$ there is a (unique) node in $\mathrm{rhs}(p_i)$ with label $\mathrm{lhs}(p_{i+1})$, and (if $n \geq 1$) there is a (unique) node in $\mathrm{rhs}(p_n)$ with label $\sigma$.

Let $p_1 \cdots p_n\sigma \in PS(G)$. Let $y_1, \ldots, y_n$ be the unique nodes described above, with $y_i \in \mathrm{rhs}(p_i)$. Then we define $\mathrm{first}(p_1 \cdots p_n\sigma) = y_1$ (where we assume that $n \geq 1$). We also define $\mathrm{conn}(p_1 \cdots p_n\sigma) = \{(\tau, \beta/\gamma, d) \mid \exists \alpha_0, \alpha_1, \ldots, \alpha_n \in \Gamma : \alpha_0 = \beta, \alpha_n = \gamma,$ and $(\tau, \alpha_{i-1}/\alpha_i, y_i, d) \in \mathrm{con}(p_i)$ for all $1 \leq i \leq n\}$. And we define $\mathrm{lhs}(p_1 \cdots p_n\sigma) = \mathrm{lhs}(p_1)$ if $n \geq 1$, and $\sigma$ if $n = 0$. □

Recall that $\mathrm{con}(p_i)$ is the connection relation of the production $p_i$. Intuitively, $p_1 \cdots p_n\sigma \in PS(G)$ is the sequence of labels of the vertices on a path from a

vertex to a leaf, in a sentential form of the derivation tree grammar $G'$ of $G$; the internal vertices are labeled by productions $p_1, \ldots, p_n$ and the leaf is labeled by $\sigma$. The set $\mathrm{conn}(p_1 \cdots p_n \sigma)$ formalizes the total effect of the connection instructions that are used when the productions are applied and produce the leaf $y_n$. Note that, for $n = 0$, $\mathrm{conn}(\sigma) = \{(\tau, \beta/\beta, d) \mid \tau \in \Sigma, \beta \in \Gamma, d \in \{\mathrm{in}, \mathrm{out}\}\}$. Note also that $\mathrm{conn}(p_1\sigma) = \{(\tau, \beta/\gamma, d) \mid (\tau, \beta/\gamma, y_1, d) \in \mathrm{con}(p_1)\}$ (where $y_1$ is the unique node of $\mathrm{rhs}(p_1)$ with label $\sigma$).

It should be clear that $PS(G)$ is regular: it is easy to define a finite automaton that checks the requirements in its definition. The following fact is obvious from the definition of 'conn'.

**Lemma 18.** *Let $w_1 w_2 \sigma \in PS(G)$, with $w_i \in P^*$ and $\sigma \in \Sigma$. Then $(\tau, \beta/\gamma, d) \in \mathrm{conn}(w_1 w_2 \sigma)$ if and only if there exists $\delta \in \Gamma$ such that $(\tau, \beta/\delta, d) \in \mathrm{conn}(w_1\sigma_1)$ and $(\tau, \delta/\gamma, d) \in \mathrm{conn}(w_2\sigma)$, where $\sigma_1 = \mathrm{lhs}(w_2\sigma)$.*

In the next lemma we show that the property of confluence can be generalized to two arbitrary sequences of productions, rather than just two productions. Although this is a well-known fact, we need it here in the following technical form.

**Lemma 19.** *Let $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$ be a uniquely labeled C-edNCE grammar. For all path strings $w_1\sigma_1, w_2\sigma_2 \in PS(G)$ (with $w_i \in P^*$ and $\sigma_i \in \Sigma$), and all edge labels $\alpha, \gamma \in \Gamma$, the following equivalence holds, where $X_i = \mathrm{lhs}(w_i\sigma_i)$:*

$\exists \beta \in \Gamma : (X_2, \alpha/\beta, \mathrm{out}) \in \mathrm{conn}(w_1\sigma_1)$ *and* $(\sigma_1, \beta/\gamma, \mathrm{in}) \in \mathrm{conn}(w_2\sigma_2)$

$\iff$

$\exists \beta \in \Gamma : (X_1, \alpha/\beta, \mathrm{in}) \in \mathrm{conn}(w_2\sigma_2)$ *and* $(\sigma_2, \beta/\gamma, \mathrm{out}) \in \mathrm{conn}(w_1\sigma_1)$.

*Proof.* For $w_1, w_2 \in P$ the above equivalence is exactly the one in the definition of confluence (Definition 3). For $w_1 = \Lambda$ (i.e., the empty string), both conditions say that $(\sigma_1, \alpha/\gamma, \mathrm{in}) \in \mathrm{conn}(w_2\sigma_2)$, and similarly for $w_2 = \Lambda$.

For the remaining cases we use induction on the sum of the lengths of $w_1$ and $w_2$. The basis of the induction has already been treated. In the induction step we may assume that $w_1$ and $w_2$ are nonempty. Or, viewed in another way, consider $p_1 w_1 \sigma_1$ and $p_2 w_2 \sigma_2$ in $PS(G)$, with $p_i \in P$, $w_i \in P^*$, and $\sigma_i \in \Sigma$. Let $\tau_i = \mathrm{lhs}(w_i\sigma_i)$. Intuitively, it should now be possible to change the sequence of productions $p_1 w_1 p_2 w_2$ into the sequence $p_1 p_2 w_1 w_2$ by interchanging $w_1$ and $p_2$, then changing it into $p_2 p_1 w_2 w_1$ by interchanging both $p_1, p_2$ and $w_1, w_2$, and finally changing it into $p_2 w_2 p_1 w_1$ by interchanging $p_1$ and $w_2$. Formally this is proved as follows.

$\exists \beta \in \Gamma : (X_2, \alpha/\beta, \mathrm{out}) \in \mathrm{conn}(p_1 w_1 \sigma_1)$ and $(\sigma_1, \beta/\gamma, \mathrm{in}) \in \mathrm{conn}(p_2 w_2 \sigma_2)$
$\iff$ (by Lemma 18 twice)
$\exists \beta, \delta, \delta' \in \Gamma$:
$(X_2, \alpha/\delta, \mathrm{out}) \in \mathrm{conn}(p_1\tau_1)$,
$(X_2, \delta/\beta, \mathrm{out}) \in \mathrm{conn}(w_1\sigma_1)$,
$(\sigma_1, \beta/\delta', \mathrm{in}) \in \mathrm{conn}(p_2\tau_2)$, and
$(\sigma_1, \delta'/\gamma, \mathrm{in}) \in \mathrm{conn}(w_2\sigma_2)$

$\iff$ (by induction for the second and third line)
$\exists \beta, \delta, \delta' \in \Gamma$:
$(X_2, \alpha/\delta, \text{out}) \in \text{conn}(p_1\tau_1)$,
$(\tau_1, \delta/\beta, \text{in}) \in \text{conn}(p_2\tau_2)$,
$(\tau_2, \beta/\delta', \text{out}) \in \text{conn}(w_1\sigma_1)$, and
$(\sigma_1, \delta'/\gamma, \text{in}) \in \text{conn}(w_2\sigma_2)$
$\iff$ (by confluence for the first two lines, and by induction for the last two lines)
$\exists \beta, \delta, \delta' \in \Gamma$:
$(X_1, \alpha/\delta, \text{in}) \in \text{conn}(p_2\tau_2)$,
$(\tau_2, \delta/\beta, \text{out}) \in \text{conn}(p_1\tau_1)$,
$(\tau_1, \beta/\delta', \text{in}) \in \text{conn}(w_2\sigma_2)$, and
$(\sigma_2, \delta'/\gamma, \text{out}) \in \text{conn}(w_1\sigma_1)$
$\iff$ (by induction for the second and third line)
$\exists \beta, \delta, \delta' \in \Gamma$:
$(X_1, \alpha/\delta, \text{in}) \in \text{conn}(p_2\tau_2)$,
$(X_1, \delta/\beta, \text{in}) \in \text{conn}(w_2\sigma_2)$,
$(\sigma_2, \beta/\delta', \text{out}) \in \text{conn}(p_1\tau_1)$, and
$(\sigma_2, \delta'/\gamma, \text{out}) \in \text{conn}(w_1\sigma_1)$
$\iff$ (by Lemma 18 twice)
$\exists \beta \in \Gamma : (X_1, \alpha/\beta, \text{in}) \in \text{conn}(p_2w_2\sigma_2)$ and $(\sigma_2, \beta/\gamma, \text{out}) \in \text{conn}(p_1w_1\sigma_1)$. □

We are now ready to show the inclusion of C-edNCE in RPD.

**Lemma 20.** *C-edNCE $\subseteq$ RPD.*

*Proof.* By Lemma's 14 and 15 it suffices to consider a uniquely labeled C-edNCE grammar $G = (\Sigma, \Delta, \Gamma, \Omega, P, S)$. Instead of constructing a regular path description of $L(G)$, we will construct one of $SF(G) \subseteq GR_{\Sigma,\Gamma}$, the set of sentential forms of $G$. This is sufficient, by Lemma 10, because $L(G) = SF(G) \cap GR_{\Delta,\Omega}$. We define the regular path description $R = (P \cup \Sigma, \Sigma, \Gamma, SF(G'), h, W)$, where $G' = (P \cup \Sigma, P \cup \Delta, P', S)$ is the derivation tree grammar of $G$, and $h$ is the identity mapping on $\Sigma$. It remains to define $W$.

Thus, we use the set of sentential forms of $G'$ as the regular tree language of $R$ (and it should be clear that it is indeed regular). By the definition of $h$, the nodes of $gr_R(t)$ are exactly the leaves of the sentential form $t$ of $G'$ (to be honest, this is not entirely true: if the right-hand side of a production $p$ is the empty graph, then $p$ is of rank 0 and hence may be the label of a leaf of $t$).

For a string $w \in P^*$ we denote by $\widetilde{w}$ the *reverse of* $w$, i.e., if $w = p_1p_2 \cdots p_n$, then $\widetilde{w} = p_n \cdots p_2p_1$. For $\gamma \in \Gamma$ we define

$$\begin{aligned} W(\gamma) = \{\sigma_1\widetilde{w_1}\overline{p}w_2\sigma_2 \mid\ & p \in P, w_1, w_2 \in P^*, \sigma_1, \sigma_2 \in \Sigma, \\ & pw_1\sigma_1, pw_2\sigma_2 \in PS(G), \text{ and, for } x_i = \text{first}(pw_i\sigma_i), \\ & \exists \alpha, \beta \in \Gamma : (x_1, \alpha, x_2) \in E_{\text{rhs}(p)}, \\ & \qquad (\lambda_{\text{rhs}(p)}(x_2), \alpha/\beta, \text{out}) \in \text{conn}(w_1\sigma_1), \text{ and} \\ & \qquad (\sigma_1, \beta/\gamma, \text{in}) \in \text{conn}(w_2\sigma_2)\}. \end{aligned}$$

It is straightforward to show that $W(\gamma)$ is regular. The main point is that $\mathrm{conn}(w)$ can be computed by a finite automaton.

This ends the definition of the regular path description $R$. To prove that $L(R) = SF(G)$, it can be shown by induction on the length of the derivations that, for every $H \in GR_{\Sigma,\Gamma}$, $sn(S,z) \Rightarrow_G^* H$ if and only if there exists $t \in T_{P\cup\Sigma}$ such that $sn(S,z) \Rightarrow_{G'}^* t$ and $gr_R(t) = H$. For the induction step it suffices to show the following statement, for every $t \in T_{P\cup\Sigma}$:

$$\text{if } gr_R(t) = H,\ H \Rightarrow_{v,p} H' \text{ in } G, \text{ and } t \Rightarrow_{v,p'} t' \text{ in } G', \text{ then } gr_R(t') = H'$$

where $p'$ is the production of $G'$ of the form $X \to p\sigma_1 \cdots \sigma_k$. Note that since $gr_R(t) = H$, every node of $H$, and in particular $v$, is also a vertex of $t$. Let us sketch the proof of the above statement. To be more precise, $p$ and $p'$ are production copies, and we (may) assume that the right-hand side of $p'$ has vertices $x_0, x_1, \ldots, x_k$ with labels $p, \sigma_1, \ldots, \sigma_k$, respectively, where $x_1, \ldots, x_k$ are the nodes of $\mathrm{rhs}(p)$, with the same labels. Thus, $t'$ is obtained from $t$ by replac-

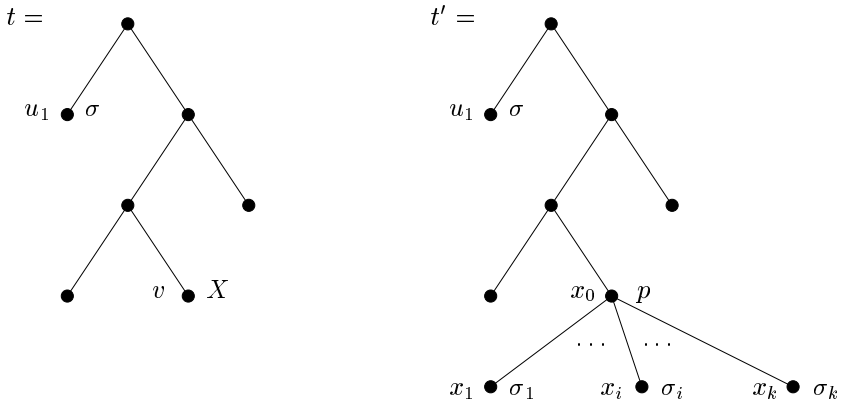


**Fig. 9.** A derivation step.

ing the leaf $v$ by the internal vertex $x_0$ with children $x_1, \ldots, x_k$ (cf. Fig. 9), and $H'$ is obtained from $H$ by replacing the node $v$ by the nodes $x_1, \ldots, x_k$ (and, of course, executing the connection instructions). From this, and the definition of $h$, it should be clear that $gr_R(t')$ and $H'$ have the same nodes, with the same labels. It remains to show that they have the same edges. Note that since $G$ is uniquely labeled, the labels $\sigma_1, \ldots, \sigma_k$ of $x_1, \ldots, x_k$ are all distinct. Let $u_1$ and $u_2$ be two leaves of $t'$. It should be clear that if $u_1$ and $u_2$ are both leaves of $t$, then $\mathrm{path}_{t'}(u_1, u_2) = \mathrm{path}_t(u_1, u_2)$, and hence they have the same edges in $gr_R(t')$ and $gr_R(t)$; since they also have the same edges in $H$ and $H'$ (by Definition 2), they have the same edges in $gr_R(t')$ and $H'$. If $u_1 = x_i$ and $u_2 = x_j$ for some $1 \le i, j \le k$, then $\mathrm{path}_{t'}(u_1, u_2) = \sigma_i \overline{p} \sigma_j$. Since, clearly, $\sigma_i \overline{p} \sigma_j \in W(\gamma)$ if and only if $(x_i, \gamma, x_j) \in E_{\mathrm{rhs}(p)}$, $u_1$ and $u_2$ have the same edges in $gr_R(t')$ and $H'$.

The last, and most important case to consider is that $u_1$ is a leaf of $t$ and $u_2 = x_i$ for some $1 \leq i \leq k$, see Fig. 9. Let us first show that $gr_R(t')$ and $H'$ have the same edges $(u_1, \gamma, x_i)$ from $u_1$ to $u_2$. Clearly, $\mathrm{path}_{t'}(u_1, x_i) = \sigma\widetilde{w_1}\overline{p'}w_2 p\sigma_i$ with $\mathrm{path}_t(u_1, v) = \sigma\widetilde{w_1}\overline{p'}w_2 X$ (recall that $X$ is the label of $v$). Now
$(u_1, \gamma, x_i) \in E_{gr(t')}$
$\iff$ (definition of $W(\gamma)$)
$\exists \alpha, \beta \in \Gamma$:
$(x_1, \alpha, x_2) \in E_{\mathrm{rhs}(p')}$,
$(\lambda_{\mathrm{rhs}(p')}(x_2), \alpha/\beta, \mathrm{out}) \in \mathrm{conn}(w_1\sigma)$, and
$(\sigma, \beta/\gamma, \mathrm{in}) \in \mathrm{conn}(w_2 p\sigma_i)$
(where $x_1 = \mathrm{first}(p'w_1\sigma)$ and $x_2 = \mathrm{first}(p'w_2 p\sigma_i)$)
$\iff$ (Lemma 18)
$\exists \alpha, \beta, \delta \in \Gamma$:
$(x_1, \alpha, x_2) \in E_{\mathrm{rhs}(p')}$,
$(\lambda_{\mathrm{rhs}(p')}(x_2), \alpha/\beta, \mathrm{out}) \in \mathrm{conn}(w_1\sigma)$,
$(\sigma, \beta/\delta, \mathrm{in}) \in \mathrm{conn}(w_2 X)$, and
$(\sigma, \delta/\gamma, \mathrm{in}) \in \mathrm{conn}(p\sigma_i)$
$\iff$ (definition of $W(\delta)$ and definition of 'conn')
$\exists \delta \in \Gamma : (u_1, \delta, v) \in E_{gr(t)}$ and $(\sigma, \delta/\gamma, x_i, \mathrm{in}) \in \mathrm{con}(p)$
$\iff$ $(gr(t) = H)$
$\exists \delta \in \Gamma : (u_1, \delta, v) \in E_H$ and $(\sigma, \delta/\gamma, x_i, \mathrm{in}) \in \mathrm{con}(p)$
$\iff$ (Definition 2)
$(u_1, \gamma, x_i) \in E_{H'}$.

Finally, it has to be shown that $gr_R(t')$ and $H'$ have the same edges $(x_i, \gamma, u_1)$ from $u_2$ to $u_1$. In this case we have $\mathrm{path}_{t'}(x_i, u_1) = \sigma_i p\widetilde{w_2}\overline{p'}w_1\sigma$ and $\mathrm{path}_t(v, u_1) = X\widetilde{w_2}\overline{p'}w_1\sigma$. Trying the same proof as above would not work, because the definition of $W(\gamma)$ forces the execution of $w_2 p$ *before* the execution of $w_1$, which does not allow the reduction to the case of $w_2$ and $w_1$. However, Lemma 19 allows us to interchange the order of these executions. To be more precise, by Lemma 19, the last two lines of the definition of $W(\gamma)$ can be changed into

$$(\lambda_{\mathrm{rhs}(p)}(x_1), \alpha/\beta, \mathrm{in}) \in \mathrm{conn}(w_2\sigma_2), \text{ and}$$
$$(\sigma_2, \beta/\gamma, \mathrm{out}) \in \mathrm{conn}(w_1\sigma_1).$$

Using this "alternative definition" of $W(\gamma)$, the proof is completely analogous to the one above (syntactically, change $(u_1, \gamma, x_i)$ into $(x_i, \gamma, u_1)$, $(x_1, \alpha, x_2)$ into $(x_2, \alpha, x_1)$, $(u_1, \delta, v)$ into $(v, \delta, u_1)$, and interchange 'in' and 'out').

This shows that $gr_R(t') = H'$, and ends the proof. □

Lemma's 13 and 20 give the main result.

**Theorem 21.** *C-edNCE = RPD.*

As can be seen from the remarks at the end of the proofs of Lemma 12 and Lemma 13, we have also shown that for every C-edNCE grammar there is an equivalent nonblocking C-edNCE grammar. The essence of this fact is contained in the proof of Lemma 10.

As an application of Theorem 21 we show that B-edNCE is a proper subclass of C-edNCE. For a graph $H$, its *edge complement* is the graph $\mathrm{com}(H) = (V_H, E, \lambda_H)$ where $E$ is the set of all $(v, \gamma, w)$, $v \neq w$, that are not in $E_H$. It is easy to see that RPD is closed under edge complement (i.e., if $L \in \mathrm{RPD}$ then $\{\mathrm{com}(H) \mid H \in L\} \in \mathrm{RPD}$). In fact, it suffices to change each $W(\gamma)$ into the regular string language $\Sigma^* \overline{\Sigma} \Sigma^* - W(\gamma)$. Hence, by Theorem 21, C-edNCE is closed under edge complement. The class of B-edNCE languages is *not* closed under edge complement (see Theorem 35 of [ELW]). This proves that B-edNCE is a proper subclass of C-edNCE. A concrete example of a graph language in C-edNCE that is not in B-edNCE is the set of edge complements of binary trees.

**Theorem 22.** *B-edNCE is a proper subclass of C-edNCE.*

## 5 Special Cases

In this section we consider a number of variations of the main result.

It follows from the proof of Theorem 21 (and in particular from the proof of Lemma 20) that, for a regular path description $R = (\Sigma, \Delta, \Omega, T, h, W)$, we may always assume that $h$ is only defined for symbols in $\Sigma$ of rank 0. This means that, for every tree $t \in T_\Sigma$, all the nodes of $gr_R(t)$ are leaves of $t$. In fact, if $L(R)$ does not contain the empty graph, then we may even assume that the domain of $h$ is exactly the set of symbols in $\Sigma$ of rank 0 (because it is easy to show that a C-edNCE language without the empty graph can be generated by a C-edNCE grammar of which all right-hand sides of productions are non-empty). That means that the nodes of $gr_R(t)$ are exactly the leaves of $t$. It is an *open problem* whether it may always be assumed that $h$ is a total function, i.e., that the nodes of $gr_R(t)$ are exactly all vertices of $t$ (again assuming that the empty graph is not in $L(R)$). In the following proposition we treat a special case.

**Proposition 23.** *Let $G$ be a C-edNCE grammar such that every right-hand side of a production has exactly one terminal node. Then there is a regular path description $R$ of $L(G)$ such that $h$ of $R$ is a total function.*

*Proof.* Note first that the two properties are preserved by Lemmas 14 and 15, respectively. Now consider the construction in the proof of Lemma 20, followed by the one in the proof of Lemma 10. This shows that $L(G)$ has a regular path description $R_1$ which can be obtained from $R$ (in the proof of Lemma 20) by changing the regular tree language of $R$ into some regular tree language $T \subseteq L(G')$. Thus, $R_1 = (P \cup \Delta, \Delta, \Omega, T, h, W)$ where $h$ is the identity on $\Delta$, and $W(\gamma)$ is defined as in the proof of Lemma 20 (for every $\gamma \in \Omega$). Clearly, we may assume that for every internal vertex $v$ of every tree $t \in T$, the last child of $v$ is a leaf, whereas the other children are not leaves. The nodes of $gr_R(t)$ are exactly the leaves of $t$. The idea is now to prune all leaves from $t$, and to let each (former) internal vertex $v$ take over the role of its (former) last child. Formally, we define the regular path description $R_2 = (P, \Delta, \Omega, T', h', W')$ such that the rank of a production $p$ of $G$ is the number of nonterminal nodes of $\mathrm{rhs}(p)$,

$T' = \{pr(t) \mid t \in T\}$ where $pr(t)$ is obtained from $t$ by removing all its leaves, $h'$ is the total function from $P$ to $\Delta$ such that for every $p \in P$, $h'(p) = \lambda_{\mathrm{rhs}(p)}(x_p)$ where $x_p$ is the unique terminal node of $\mathrm{rhs}(p)$, and $W'(\gamma)$ is the set of all strings $\widetilde{w_1}\overline{p}w_2$ (with $w_i \in P^*$ and $p \in P$) such that $\sigma_1\widetilde{w_1}\overline{p}w_2\sigma_2 \in W(\gamma)$ where $\sigma_i$ is the label of the terminal node of the right-hand side of the last production of $pw_i$. Using the regularity of $T$ and $W(\gamma)$, it is easy to show that $T'$ and $W'(\gamma)$ are regular. It should be clear that $L(R_2) = L(G)$. This proves the proposition.

As an example, we observe that the regular path description $R_2$ of Example 3 is obtained from the C-edNCE grammar $G_2$ of Example 1 by the above construction. Note that $G_2$ satisfies the assumption of this proposition. The regular tree grammar $G$ that generates the regular tree language of $R_2$ is obtained from $G_2$ as follows: apply the construction of Lemma 14 to $G_2$ in order to make it uniquely labeled (replacing $X$ by $L$ and $R$), take the derivation tree grammar of the resulting C-edNCE grammar, and prune the terminal leaves from the productions of the resulting regular tree grammar. Note that $a, b_l, b_r, c_l, c_r$ are the productions of the uniquely labeled C-edNCE grammar. □

Let us now consider the class B-edNCE of graph languages generated by boundary edNCE grammars (see Section 2 for the definition). Recall from Section 3 the definition of the class B-RPD of regular path descriptions of type B: every $W(\gamma)$ is a subset of $\Sigma^*\overline{\Sigma} \cup \overline{\Sigma}\Sigma^*$.

**Theorem 24.** *B-edNCE = B-RPD.*

*Proof.* First the inclusion B-RPD $\subseteq$ B-edNCE. Let us say that an automaton path description $R = (\Sigma, \Delta, T, h, A)$ is of type B if $L(A) \subseteq \Sigma^*\overline{\Sigma} \cup \overline{\Sigma}\Sigma^*$. Now consider the proof of Lemma 12 for $R$ of type B. It should be clear that the grammar $G'$ need not keep edges between nonterminal vertices any more. Hence, in the definition of the production $p' = X \to (D, C)$ of $G'$, the edges $(x_i, \langle q_1, q_2\rangle, x_j)$ can be dropped from $E_D$. Then the $\tau$ in the connection instructions of $C$ can be restricted to $\Delta$. This turns $G'$ into a B-edNCE grammar. It should be clear that the construction in the proof of Lemma 13 changes a regular path description of type B into an automaton path description of type B.

We now turn to the second inclusion: B-edNCE $\subseteq$ B-RPD. It is shown in Theorem 24 of [ELW] that every B-edNCE language (not containing the empty graph) can be generated by a B-edNCE grammar of which the right-hand side of every production contains exactly one terminal node (the proof is for undirected graphs, but can be adapted in a straightforward way to directed graphs). This allows us to use the construction in the proof of Proposition 23. To this aim, we first have to check the construction in the proof of Lemma 20. Consider the definition of $W(\gamma)$ in that proof. We claim that if $\sigma_1\widetilde{w_1}\overline{p}w_2\sigma_2$ is in $W(\gamma)$, then either $w_1 = \Lambda$ or $w_2 = \Lambda$ (where $\Lambda$ is the empty string). In fact, if $x_2$ is a nonterminal node, then $w_1 = \Lambda$ because $(\lambda_{\mathrm{rhs}(p)}(x_2), \alpha/\beta, \mathrm{out}) \in \mathrm{conn}(w_1\sigma_1)$ and $G$ is a B-edNCE grammar; if, on the other hand, $x_2$ is a terminal node, then $w_2 = \Lambda$ because $x_2 = \mathrm{lhs}(w_2\sigma_2)$. This shows that $W(\gamma) \subseteq \Sigma P^*\overline{P}\Sigma \cup \Sigma\overline{P}P^*\Sigma$. Hence, for the regular path description $R_2$ in the proof of Proposition 23, we obtain that $W'(\gamma) \subseteq P^*\overline{P} \cup \overline{P}P^*$, which shows that $R_2$ is of type B. □

Next we consider the class LIN-edNCE of graph languages generated by linear edNCE grammars (see again Section 2 for the definition). Recall again from Section 3 the definition of the class LIN-RPD of regular path descriptions of type LIN: the symbols of $\Sigma$ have rank 0 or 1.

**Theorem 25.** *LIN-edNCE = LIN-RPD.*

*Proof.* It is easy to check the proofs to see that they preserve linearity. The inclusion LIN-edNCE $\subseteq$ LIN-RPD is based on the fact (shown in Theorem 24 of [ELW] for undirected graphs) that every LIN-edNCE language can be generated by a LIN-edNCE grammar of which the right-hand side of every production contains exactly one terminal node. Then all elements of the ranked alphabet $P$ of $R_2$ in the proof of Proposition 23 have rank 0 or 1. □

We finally consider the class A-edNCE of graph languages generated by apex edNCE grammars (see again Section 2). Recall again from Section 3 the definition of the class A-RPD of regular path descriptions of type A: every $W(\gamma)$ is finite. The next result solves a conjecture on p.339 of [ELW].

**Theorem 26.** *A-edNCE = A-RPD.*

*Proof.* To show that A-RPD $\subseteq$ A-edNCE, we use Proposition 5. By this proposition and the inclusion of RPD in C-edNCE (Theorem 21), it suffices to show that every graph language in A-RPD is of bounded degree. Let $R = (\Sigma, \Delta, \Omega, T, h, W)$ be a regular path description such that $W(\gamma)$ is finite for every $\gamma \in \Omega$. Let $k$ be the maximal length of the strings in the $W(\gamma)$'s. Consider some $t \in T$ and $u \in V_t$. In $gr_R(t)$, if there is an edge between $u$ and $v$ then either $\mathrm{path}_t(u, v)$ or $\mathrm{path}_t(v, u)$ is in $W(\gamma)$ for some $\gamma \in \Gamma$. Hence the distance between $u$ and all such vertices $v$ is at most $k$. Let $m$ be the maximal rank of the elements of $\Sigma$. Then there are at most $(m+1)^k$ vertices within distance $k$ from $u$. Hence $u$ is of degree at most $2 \cdot \#\Omega \cdot (m+1)^k$ in $gr_R(t)$.

To show that A-edNCE $\subseteq$ A-RPD, consider the proof of Lemma 20. If $G$ is an A-edNCE grammar, then $W(\gamma)$ is finite, for every $\gamma \in \Gamma$. In fact, it is easy to see from Definition 17 that, in that case, if $n > 1$ then $\mathrm{conn}(p_1 \cdots p_n \sigma) = \emptyset$ (because all $y_i$ must be terminal). This implies that all path strings in $W(\gamma)$ have length at most 5 (and even 4, because at least one of the strings $w_i$ in the definition of $W(\gamma)$ is empty, cf. the proof of Theorem 24). □

It can be shown, using a slightly more complicated construction than the one in the proof of Proposition 23, that every regular path description of type A is equivalent to one for which every $W(\gamma)$ is a finite subset of $\Sigma^*\overline{\Sigma} \cup \overline{\Sigma}\Sigma^*$. The basic idea is to remove all children of an internal vertex $v$ that are leaves, and replace them by a sequence of unary vertices above $v$.

## 6 String Languages

It is well known that an edge labeled graph can be used to define a regular string language, consisting of the strings of edge labels along all directed paths in $H$,

from certain initial nodes of $H$ to certain final nodes of $H$. In fact, this is just another way of saying that the graph $H$ is viewed as a nondeterministic finite automaton. To define nonregular string languages one might use a set of graphs rather than just one graph. Clearly, allowing arbitrary sets of graphs would give arbitrary string languages. Thus, it would be more natural to use only graph languages that can be generated by certain graph grammars. Here we investigate the string generating power (in the above sense) of C-edNCE graph grammars. We will show that in this way they generate the class of output languages of nondeterministic tree-walking transducers (cf. [ERS]). The LIN-edNCE grammars generate the class of checking stack languages (cf. [Gre1]). To simplify the proofs, we will allow the edges of the graphs to be labeled by arbitrary strings (including the empty string). This corresponds to finite automata that can read an arbitrary (possibly empty) string in one step. It is formalized by applying a string homomorphism to the language recognized by an ordinary finite automaton.

**Definition 27.** Let $H \in GR_{\Delta,\Omega}$, and let $i, f \in \Delta$. Then $\mathrm{path}_{i,f}(H)$ is defined to be the set of all $\gamma_1 \cdots \gamma_n \in \Omega^*$ such that there is a directed path with nodes $v_0, v_1, \ldots, v_n$, $n \geq 1$, in $H$ with $\lambda_H(v_0) = i$, $\lambda_H(v_n) = f$, and $(v_{j-1}, \gamma_j, v_j) \in E_H$ for all $1 \leq j \leq n$.

Let $K$ be a class of graph languages. Then $\mathrm{Path}(K)$ is the class of all string languages $\mathrm{path}_{i,f}(L) = \{\mathrm{path}_{i,f}(H) \mid H \in L\}$, where $L \in K$, and $i, f$ are arbitrary node labels. And $\mathrm{HPath}(K)$ is the class of string languages $\phi(L)$, where $L \in \mathrm{Path}(K)$ and $\phi$ is an arbitrary string homomorphism. □

As an example, for the graph language $L(G_1)$ of "ladders" from Example 1, $\mathrm{path}_{i,f}(L(G_1))$ is the set of all strings $\rho^{n_1}\delta\lambda^{n_1}\upsilon \cdots \rho^{n_k}\delta\lambda^{n_k}\upsilon\rho^m\delta$ with $k \geq 0$, $m \geq 2$, and $1 \leq n_j \leq m$ for all $1 \leq j \leq k$. Clearly this language is not regular (not even context-free), but it is a checking stack language (guess the number $m$ on the checking stack, walk up and down part of the stack, $k$ times, and walk up the whole stack).

We now describe the tree-walking transducer in an informal way (detailed definitions can be found in, e.g., [ERS]). A *tree-walking transducer*, abbreviated twt, is a nondeterministic automaton with a finite control, an input tree, and an output tape. The input trees are taken from a given regular tree language (over some ranked alphabet). At any moment of time the automaton is at a certain vertex of the input tree. Depending on the state of its finite control and the label of the vertex, it changes state, outputs a string to the output tape, and either moves to the parent or to a specific child of the vertex. The automaton starts in its initial state at the root of the input tree, and halts whenever it reaches a final state. In this way it nondeterministically translates the input tree into an output string. The *output language* of the automaton is the set of all output strings that are translations of input trees from the given regular tree language. By OUT(TWT) we denote the class of all output languages of tree-walking transducers. From a slightly different point of view, one could view a twt as computing a relation between trees and strings; OUT(TWT) is the class of images of regular tree languages under such twt relations. In the case that the labels of the input tree all have rank 1 or 0, the input tree can be viewed

as a two-way input tape, and the twt as a nondeterministic two-way gsm (i.e., a finite state transducer with a two-way input tape; 'gsm' abbreviates generalized sequential machine, see, e.g., [HU]). By OUT(2GSM) we denote the class of all output languages of two-way gsm's. It is well known that OUT(2GSM) equals the class of (one-way) checking stack languages; in fact, the input tape and output tape of the two-way gsm can be viewed as the checking stack and the one-way input tape of the checking stack automaton, respectively. For more details on the above, see [ERS] (where the twt is called checking tree transducer or ct-transducer, and the two-way gsm is called checking string transducer or cs-transducer).

We now use our main result (Theorem 21) to show that C-edNCE grammars have the same string generating power as tree-walking transducers. For LIN-edNCE grammars we use Theorem 25 to show that they have the same string generating power as two-way gsm's (or checking stack automata). The proof will be as informal as the description of the twt above.

**Theorem 28.** *HPath(C-edNCE) = OUT(TWT) and HPath(LIN-edNCE) = OUT(2GSM).*

*Proof.* In this proof, whenever we consider a regular tree language $T \subseteq T_\Sigma$ (either of a regular path description or of a twt), we will assume that there is a mapping $\mathrm{num} : \Sigma \to \{0, 1, 2, \ldots\}$ such that for every vertex $x$ of a tree $t \in T$, if $\mathrm{num}(\lambda_t(x)) = j$, then either $x$ is not the root and $j$ is the label of the incoming edge of $x$ (i.e., $x$ is the $j$th child of its parent), or $x$ is the root of $t$ and $j = 0$. Clearly, this assumption can be made without loss of generality.

We first show that HPath(RPD) $\subseteq$ OUT(TWT) and HPath(LIN-RPD) $\subseteq$ OUT(2GSM). Let $R = (\Sigma, \Delta, \Omega, T, h, W)$ be a regular path description, let $i, f \in \Delta$, and let $\phi$ be a string homomorphism with domain $\Delta^*$. It is not difficult to construct a twt $M$ with input tree language $T$ and with output language $\phi(\mathrm{path}_{i,f}(L(R)))$. For a given input tree $t \in T$, $M$ first nondeterministically walks to a vertex $x$ of $t$ such that $h(\lambda_t(x)) = i$. Then, repeatedly, $M$ chooses a symbol $\gamma \in \Omega$, outputs $\phi(\gamma)$, and nondeterministically walks to another vertex $y$ for which $h(\lambda_t(y))$ is defined, walking along the shortest undirected path from $x$ to $y$, and using its finite control to check that $\mathrm{path}_t(x, y)$ is in $W(\gamma)$. Finally, $M$ halts after checking that $h(\lambda_t(x)) = f$ for the current vertex $x$. Note that, when walking from $x$ to $y$ along the shortest path from $x$ to $y$, $M$ first ascends to the least common ancestor $z$ of $x$, and then (in general) descends to $y$; to do this, $M$ has to store the number $\mathrm{num}(\lambda_t(x_1))$ of the child $x_1$ of $z$ of which $x$ is a descendant, in its finite control, in order to be able to descend to *another* child $y_1$ of $z$, of which $y$ will be a descendant.

Next we show that OUT(TWT) $\subseteq$ HPath(RPD). Let $M$ be a twt with input tree language $T \subseteq T_\Sigma$ and output alphabet $\Omega$. Since a string homomorphism is incorporated into the definition of HPath(RPD), it clearly suffices to assume that $M$ outputs exactly one symbol from $\Omega$ at each move, and to prove that the output language of $M$ is in Path(RPD). Also, we may assume that $M$ never re-enters its initial state, and that it has exactly one final state. Let $Q$ be the

set of states of $M$, and let $i, f \in Q$ $(i \neq f)$ be its initial and final state, respectively. We construct a regular path description $R = (\Sigma', \Delta, \Omega, T', h, W)$ such that $\mathrm{path}_{i,f}(L(R))$ is the output language of $M$. The trees of $T'$ are obtained from those of $T$ as follows: for a tree $t \in T$ we construct the tree $t' \in T'$ by adding $(\#Q) - 1$ new children to every vertex $x$ of $t$, labeled (distinctly) with the elements of $Q - \{i\}$, and adding one more new child, with label $i$, if $x$ is the root of $t$. Such a new child, with label $q \in Q$, intuitively represents the fact that $M$ is at vertex $x$ of $t$ in state $q$. Thus, $\Sigma' = \Sigma \cup Q$, where $\mathrm{rank}'(\sigma) = \mathrm{rank}(\sigma) + \#Q - 1$ for every $\sigma \in \Sigma$ with $\mathrm{num}(\sigma) \neq 0$, $\mathrm{rank}'(\sigma) = \mathrm{rank}(\sigma) + \#Q$ for every $\sigma \in \Sigma$ with $\mathrm{num}(\sigma) = 0$, and $\mathrm{rank}(q) = 0$ for every $q \in Q$. Clearly, $T'$ is a regular tree language. We take $\Delta = \Sigma'$, and we take $h$ to be the identity on $\Sigma'$. Finally, for every $\gamma \in \Omega$, $W(\gamma)$ is a finite language obtained from the finite control of $M$ as follows. If $M$, in state $q$ and reading vertex label $\sigma$, may go into state $p$, output $\gamma$, and move to the parent, then $W(\gamma)$ contains the string $q\sigma\overline{\tau}p$ for every $\tau \in \Sigma$. If $M$, in state $q$ and reading vertex label $\sigma$, may go into state $p$, output $\gamma$, and move to the $j$th child, then $W(\gamma)$ contains the string $q\overline{\sigma}\tau p$ for every $\tau \in \Sigma$ such that $\mathrm{num}(\tau) = j$. From this construction of $R$ it should be clear that $\mathrm{path}_{i,f}(L(R))$ is the output language of $M$. Intuitively, for a tree $t$, a directed path in the graph $gr_R(t')$ represents a walk of $M$ on $t$.

Note that we have even shown that OUT(TWT) $\subseteq$ HPath(A-RPD).

To prove that OUT(2GSM) $\subseteq$ HPath(LIN-edNCE) we first construct $R$ as above. It is not difficult to see that the regular tree language $T'$ of $R$ can be generated by a linear REGT grammar. Consider the proof of Lemma 12. It is left to the reader to show that it can be adapted easily for an arbitrary regular tree grammar $G$ (not necessarily in normal form); the only thing that changes is the definition of $E_D$ for the production $p' = X \to (D, C)$. Since the proofs of Lemma 12 and Proposition 6 both preserve linearity (cf. the proof of Theorem 25), this shows that $L(R)$ can be generated by a LIN-edNCE grammar. □

It can be shown that HPath(C-edNCE) = Path(C-edNCE), i.e., that the class Path(C-edNCE) is closed under homomorphisms, but with the tools we have now, the proof is not easy to present, and thus will not be given here.

Another way of generating string languages by graph grammars was investigated in [EH1]. Every string $\sigma_1 \cdots \sigma_n$ can be viewed as a graph with $n+1$ nodes, that are connected into a directed chain by $n$ edges, labelled $\sigma_1, \ldots, \sigma_n$. In this way, every string language can be viewed as a (special type of) graph language. For a class of graph languages $K$, we denote by $\mathrm{STR}(K)$ the class of all string languages in $K$. The string languages generated by graph grammars in this way were investigated in [EH1] for another type of context-free graph grammars: the hyperedge replacement grammars of [Hab]. However, it is known that these grammars have the same string generating power (in this sense) as the C-edNCE grammars (see, e.g., [Bra3, EH2]). From this we find that STR(C-edNCE) = OUT(DTWT), the output languages of *deterministic* tree-walking transducers, and STR(LIN-edNCE) = OUT(2DGSM), the output languages of *deterministic* two-way gsm's. Hence, this string generation method is weaker than the one

discussed in this section. For instance, the language $\{(a^m)^n \mid m, n \geq 2\}$ is in OUT(2GSM): guess $m$ on the checking stack, and then walk up and down the whole stack $n$ times (or: there is a LIN-edNCE grammar that generates the set of all directed cycles). This language is however not in OUT(DTWT), because it is not Parikh (see [ERS]). Note that, in the other direction, OUT(DTWT) contains all context-free languages, but OUT(2GSM) does not (see Theorem 4.26 of [Gre2]).

## References


[BC] M.Bauderon, B.Courcelle; Graph expressions and graph rewritings, Math. Systems Theory 20 (1987), 83-127

[Bra1] F.J.Brandenburg; On partially ordered graph grammars, in [ENRR], 99-111

[Bra2] F.J.Brandenburg; On polynomial time graph grammars, Proc. STACS 88, Lecture Notes in Computer Science 294, Springer-Verlag, Berlin, 1988, pp.227-236

[Bra3] F.J.Brandenburg; The equivalence of boundary and confluent graph grammars on graph languages of bounded degree, in *Rewriting Techniques and Applications* (R.V.Book, ed.), Lecture Notes in Computer Science 488, Springer-Verlag, Berlin, 1991, pp.312-322

[CER] B.Courcelle, J.Engelfriet, G.Rozenberg; Handle-rewriting hypergraph grammars, J. of Comp. Syst. Sci. 46 (1993), 218-270

[CLS] D.G.Corneil, H.Lerchs, L.Stewart Burlingham; Complement reducible graphs, Discr. Appl. Math. 3 (1981), 163-174

[Cou1] B.Courcelle; An axiomatic definition of context-free rewriting and its application to NLC graph grammars, Theor. Comput. Sci. 55 (1987), 141-181

[Cou2] B.Courcelle; The monadic second-order logic of graphs VII: Graphs as relational structures, Theor. Comput. Sci. 101 (1992), 3-33

[Cou3] B.Courcelle; Structural properties of context-free sets of graphs generated by vertex replacement, Inform. and Comput. 116 (1995), 275-293

[EH1] J.Engelfriet, L.M.Heyker; The string generating power of context-free hypergraph grammars, J. of Comp. Syst. Sci. 43 (1991), 328-360

[EH2] J.Engelfriet, L.M.Heyker; Hypergraph languages of bounded degree, J. of Comp. Syst. Sci. 48 (1994), 58-89

[EHL] J.Engelfriet, L.M.Heyker, G.Leih; Context-free graph languages of bounded degree are generated by apex graph grammars, Acta Informatica 31 (1994), 341-378

[EJKR] H.Ehrig, D.Janssens, H.-J.Kreowski, G.Rozenberg; Concurrency of node-label controlled graph transformations, Report 82-38, University of Antwerp, U.I.A., 1982

[EKR] H.Ehrig, H.-J.Kreowski, G.Rozenberg (eds.); *Graph-Grammars and their Application to Computer Science*, Lecture Notes in Computer Science 532, Springer-Verlag, Berlin, 1991

[EL1] J.Engelfriet, G.Leih; Linear graph grammars: power and complexity, Inform. and Comput. 81 (1989), 88-121

[EL2] J.Engelfriet, G.Leih; Complexity of boundary graph languages, RAIRO Theoretical Informatics and Applications 24 (1990), 267-274

[ELR1] J.Engelfriet, G.Leih, G.Rozenberg; Apex graph grammars, in [ENRR], 167-185

[ELR2] J.Engelfriet, G.Leih, G.Rozenberg; Nonterminal separation in graph grammars, Theor. Comput. Sci. 82 (1991), 95-111

[ELW] J.Engelfriet, G.Leih, E.Welzl; Boundary graph grammars with dynamic edge relabeling, J. of Comp. Syst. Sci. 40 (1990), 307-345

[Eng1] J.Engelfriet; Context-free NCE graph grammars, Proc. FCT '89, Lecture Notes in Computer Science 380, Springer-Verlag, Berlin, 1989, pp.148-161

[Eng2] J.Engelfriet; A characterization of context-free NCE graph languages by monadic second-order logic on trees, in [EKR], 311-327

[Eng3] J.Engelfriet; A Greibach normal form for context-free graph grammars, Proc. ICALP'92 (W.Kuich, ed.), Lecture Notes in Computer Science 623, Springer-Verlag, Berlin, 1992, pp.138-149

[Eng4] J.Engelfriet; Graph grammars and tree transducers, Proc. CAAP'94 (S.Tison, ed.), Lecture Notes in Computer Science 787, Springer-Verlag, Berlin, 1994, pp.15-36

[ENRR] H.Ehrig, M.Nagl, G.Rozenberg, A.Rosenfeld (eds.); *Graph-Grammars and their Application to Computer Science*, Lecture Notes in Computer Science 291, Springer-Verlag, Berlin, 1987

[ER1] J.Engelfriet, G.Rozenberg; A comparison of boundary graph grammars and context-free hypergraph grammars, Inform. and Comput. 84 (1990), 163-206

[ER2] J.Engelfriet, G.Rozenberg; Node replacement graph grammars, Chapter for the *Handbook of Graph Transformations*, Volume I: Foundations (G.Rozenberg, ed.), World Scientific, to appear

[ERS] J.Engelfriet, G.Rozenberg, G.Slutzki; Tree transducers, L systems, and two-way machines, J. of Comp. Syst. Sci. 20 (1980), 150-202

[Gre1] S.A.Greibach; Checking automata and one-way stack languages, J. of Comp. Syst. Sci. 3 (1969), 196-217

[Gre2] S.A.Greibach; One way finite visit automata, Theor. Comput. Sci 6 (1978), 175-221

[GS] F.Gécseg, M.Steinby; *Tree automata*, Akadémiai Kiadó, Budapest, 1984

[Hab] A.Habel; *Hyperedge Replacement: Grammars and Languages*, Lecture Notes in Computer Science 643, Springer-Verlag, 1992

[HK] A.Habel, H.-J.Kreowski; May we introduce to you: hyperedge replacement, in [ENRR], pp.15-26

[HU] J.E.Hopcroft, J.D.Ullman; *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Mass., 1979

[Kau] M.Kaul; *Syntaxanalyse von Graphen bei Präzedenz-Graph-Grammatiken*, Dissertation, Universität Osnabrück, 1985

[KL] C.Kim, D.H.Lee; Separating $k$-separated eNCE graph languages, Theor. Comput. Sci. 120 (1993), 247-259

[Nag1] M.Nagl; Formal languages of labelled graphs, Computing 16 (1976), 113-137

[Nag2] M.Nagl; A tutorial and bibliographical survey of graph grammars, in *Graph Grammars* (V.Claus, H.Ehrig, G.Rozenberg, eds.); Lecture Notes in Computer Science 73, Springer-Verlag, Berlin, 1980, pp.70-126

[Nag3] M.Nagl; *Graph-Grammatiken*, Vieweg, Braunschweig, 1979

[Oos] V.van Oostrom; Graph grammars and 2nd order logic (in Dutch), M. Sc. Thesis, Leiden University, January 1989

[RW] G.Rozenberg, E.Welzl; Boundary NLC graph grammars - basic definitions, normal forms, and complexity, Inform. and Control 69 (1986), 136-167

[Schu] R.Schuster; Graphgrammatiken und Grapheinbettungen: Algorithmen und Komplexität, Technical Report MIP-8711, Universität Passau, 1987

[SW1] K.Skodinis, E.Wanke; Exponential time analysis of confluent and boundary eNCE graph languages, in *Graph-Theoretic Concepts in Computer Science*, WG'94 (E.W.Mayr, G.Schmidt, G.Tinhofer, eds.), Lecture Notes in Computer Science 903, Springer-Verlag, Berlin, 1995, pp.180-192

[SW2] K.Skodinis, E.Wanke; The bounded degree problem for eNCE graph grammars, 5th International Workshop on Graph-Grammars and their Application to Computer Science, Williamsburg, Virginia, USA, November 1994

[Wel] E.Welzl; Boundary NLC and partition controlled graph grammars, in [ENRR], 593-609

This article was processed using the LaTeX macro package with LLNCS style