Generative Power of Reduction-based Parsable ETPR(k) Graph Grammars for Syntactic Pattern Recognition

Submitted: 30th April 2018; accepted: 25th June 2018

Mariusz Flasiński

DOI: 10.14313/JAMRIS_2-2018/12

Abstract:

*Further results of research into parsable graph grammars used for syntactic pattern recognition (Pattern Recognition: **21**, 623-629 (1988); **23**, 765-774 (1990); **24**, 12-23 (1991); **26**, 1-16 (1993); **43**, 2249-2264 (2010), Comput. Vision Graph. Image Process. **47**, 1-21 (1989), Computer-Aided Design **27**, 403-433 (1995), Theoret. Comp. Sci. **201**, 189-231 (1998), Pattern Analysis Applications bf 17, 465-480 (2014)) are presented in the paper. The generative power of reduction-based parsable ETPR(k) graph grammars is investigated. The analogy between the triad of CF - LL(k) - LR(k) string languages and the triad of NLC - ETPL(k) - ETPR(k) graph languages is discussed.*

Keywords: *syntactic pattern recognition, graph grammar, parsing*

1. Introduction

Graph grammars are the strongest descriptive/generative formalism in the theory of formal languages and automata, if compared with string or tree grammars. They are used for the synthesis of formal representations in various important areas of computer science such as: software engineering, (syntactic) pattern recognition, database design, programming languages and compiler design, computer networks, distributed and concurrent computing, logic programming, computer vision, IT systems for chemistry and biology, artificial intelligence (natural language processing, knowledge representation and rule-based systems) [10, 11, 45]. However, the use of graph automata/parsers as tools for the analysis of graph representations in these application areas is strongly limited because the membership problem for graph languages is PSPACE- or NP- complete. Research into this problem has been undertaken for 50 years. The first graph automata were defined in the 1960s by Blum and Hewitt [3]. For Pfaltz-Rosenfeld web grammars generating node-labelled graphs with embedding transformations that specify inheriting edges by pointing out proper nodes of right-hand side graphs [40], the web automata were defined by Rosenfeld and Milgram [43] in 1972 and in 1977 the web parser by Brayer [6]. In 1978 Franck constructed the precedence relations-based syntax analyzer, $O(n^2)$, n is the number of nodes, [28] for *NLC*-like grammars [30, 32][1] with restricted embedding transformations. (Later, the complexity is stated with respect to the number of nodes n.) In the same year Della Vigna and Ghezzi [8] proposed the parser, $O(n^2)$, for grammars based on the Pratt model, in which the embedding transformation is defined by determining input (output) nodes of right-hand side graphs which inherit the edges of left-hand sides [41]. The precedence relations-based parser, $O(n^2)$, was constructed by Kaul [33] for *NLC*-like grammars. In the early 1980s, subclasses of graph grammars with polynomial membership problem were studied by Brandenburg [4], Slisenko [49], and Turan [51]. Subsequently, the parsing algorithm, $O(n^2)$, for expansive graph grammars was formulated by Fu and Shi in 1983 [48]. In 1986 the polynomial parsing algorithm for boundary *NLC* languages was defined by Rozenberg and Welzl [46]. During the first half of 1990s three parsing algorithms, $O(n^2)$, based on the analogy to *LL*(k) grammars [36, 44] were defined: for the regular *ETL*(1) subclass of *edNLC* languages [13, 14], the error-correcting parser [15], and for the context-free *ETPL*(k) subclass of *edNLC* languages [16]. The first (polynomial) parser for Habel-Kreowski/Bauderon-Courcelle hyperedge replacement grammars, *HR grammars*, [1, 29] was constructed by Lautemann in 1988 [35]. The succeeding parsers for this class of graph grammars were proposed by Vogler in 1991 (the Cocke-Kasami-Younger-based parser), $O(n^3)$, [52], by Seifert and Fischer in 2004 (the Earley-based parser), $O(n^3)$, [47], by Mazanek and Minas in 2008 (a method based on polynomial graph parser combinators) [37], and in 2015 by Drewes, Hoffmann and Minas for the predictively top-down parsable subclass of *HR grammars*, $O(n^2)$, [9]. Two polynomial syntax analyzers for Feder plex grammars, which generate graph-like structures (called plexes) consisting of nodes with pre-defined attaching points (called napes), [12] were constructed independently by Bunke and Haller [7] and by Peng, Yamamoto and Aoki [39] in 1990. For relational grammars; in which the right-hand sides are structures defined with relations between labelled objects and embedding is performed in an analogous way - as with plex attaching points; parsing algorithms were proposed by Wittenburg, Weitzman and Talley in 1991 (exponential) [54] and in 1994 by Tucci, Vitiello and Costagliola (polynomial) [50]. In 1996 Wills published a paper on exponential Earley-based parsing for attributed flow graph grammars [53], which can be treated as plex grammars with attributes generating directed acyclic graphs. The exponential parser for layered graph grammars was constructed by Rekers and Schürr in 1997 [42]. Layered graph grammars

are context-sensitive grammars with decomposing node and edge alphabets into more than two layers (i.e. terminal and nonterminal layers) and imposing a kind of lexicographical order on graphs based on layers. The polynomial syntax analyzer for reserved graph grammars, which are layered grammars with reversed productions (to make parsing efficient), was defined by Zhang, Zhang and Cao in 2001 [55]. The automata for Janssens-Rozenberg *NCE* graph languages [31] were defined by Brandenburg and Skodinis in 2005 [5].

Graph grammars can be divided into two large families according to the embedding mechanism: *grammars with connecting embedding* (the set theoretic approach, the algorithmic approach) and *grammars with gluing embedding* (the algebraic approach) [45]. Within each of these families two standard classes of graph grammars, which are interesting for defining practical parsing algorithms, are distinguished [45]. The grammars with connecting embedding are *VR* (vertex replacement) grammars, mainly *NCE*-like (Neighbourhood Controlled Embedding) grammars [31] and *NLC*-like (Node Label Controlled) grammars [30,32]. The grammars with gluing embedding are *HR* (hyperedge replacement) grammars [1, 29]. Research into defining the subclasses with polynomial membership problem of *NLC*-like grammars and their applications has been carried out for the last 30 years.

The previously mentioned parsable *ETPL*(k) subclass of *edNLC* graph grammars has been successfully used for practical applications (see below). Moreover, the inference algorithm for *ETPL*(k) graph languages has been defined [20] and its descriptive power was characterized [19]. Nevertheless, in some cases its power limitations have been revealed. These limitations result from constraints imposed on the definition of *ETPL*(k) grammar in order to make it parsable in a *top-down manner*. *ETPL*(k) grammars have been defined analogously to top-down parsable (string) *LL*(k) grammars [36, 44]. It is also known that Knuth's reduction-based (bottom-up) parsable (string) *LR*(k) grammars [34] have a greater generative power than *LL*(k) grammars. Therefore, the *reduction-based parsable ETPR*(k) subclass of *edNLC* graph grammars has been defined [23, 24]. Both classes, i.e. *ETPL*(k) and *ETPR*(k) have been applied successfully for scene analysis in robotics [13], software allocation in distributed systems [25], CAD/CAM integration [18, 22], reasoning in real-time expert systems [2, 17], mesh refinement (finite element method, FEM) in CAE systems [27], sign language recognition [21, 26], and computer vision [24]. However, to date the formal properties of *ETPR*(k) graph grammars have not been presented.

The generative power of *ETPR*(k) graph grammars with polynomial membership problem is presented and the analogies between parsable subfamilies of *CF* string and *edNLC* graph languages are discussed in this paper. The definitions pertaining to *edNLC* graph grammars are given in Section 2. Notions of indexed and reversely indexed edge-unambiguous graphs that enable linear ordering on *EDG* graphs [32] to be introduced are presented in Section 3. The definitions concerning *edNLC* graph languages with polynomial membership problem are included in Section 4. The generative power of the reduction-based parsable *ETPR*(k) subclass of *edNLC* graph languages is investigated in Section 5. The discussion on the analogy between the triad of *CF* - *LL*(k) - *LR*(k) string languages and the triad of *NLC* - *ETPL*(k) - *ETPR*(k) graph languages is presented in Section 6 and the final section consists of concluding remarks.

2. Preliminaries

In this section the basic definitions of *EDG* graph, *edNLC* graph grammar and *edNLC* graph language are introduced [30, 32].

Definition 1. A *directed node- and edge-labelled graph*, *EDG graph*, over Σ and Γ is a quintuple

$$H = (V, E, \Sigma, \Gamma, \phi),$$

where V is a *finite, non-empty set of nodes*,
Σ is a *finite, non-empty set of node labels*,
Γ is a *finite, non-empty set of edge labels*,
E is a *set of edges* of the form (v, λ, w), in which
$v, w \in V, \lambda \in \Gamma$, and
$\phi : V \longrightarrow \Sigma$ is a *node-labelling function*.

The *family of the EDG graphs over* Σ *and* Γ is denoted by $EDG_{\Sigma,\Gamma}$. The components V, E, ϕ of a graph H are sometimes denoted with V_H, E_H, ϕ_H.

Let $A = (V_A, E_A, \Sigma, \Gamma, \phi_A)$, $B = (V_B, E_B, \Sigma, \Gamma,$, $\phi_B)$ and $C = (V_C, E_C, \Sigma, \Gamma, \phi_C)$ be *EDG* graphs. An *isomorphism from* A *onto* B is a bijective function h from V_A onto V_B such that

$$\phi_B \circ h = \phi_A \text{ and } E_B = \{(h(v), \lambda, h(w)) : (v, \lambda, w) \in E_A\}.$$

We say that A *is isomorphic to* B, and denote it with $A \cong B$.
A graph C is a *(full) subgraph* of B iff $V_C \subseteq V_B, E_C = \{(v, \lambda, w) \in E_B : v, w \in V_C\}$ and ϕ_C is the restriction to V_C of ϕ_B.

Definition 2. An *edge-labelled directed Node Label Controlled*, *edNLC*, *graph grammar* is a quintuple

$$G = (\Sigma, \Delta, \Gamma, P, Z),$$

where Σ is a *finite, non-empty set of node labels*,
$\Delta \subseteq \Sigma$ is a *set of terminal node labels*,
Γ is a *finite, non-empty set of edge labels*,
P is a *finite set of productions* of the form (l, D, C), in which $l \in \Sigma \backslash \Delta, D \in EDG_{\Sigma,\Gamma}, C : \Gamma \times \{in, out\} \longrightarrow 2^{\Sigma \times \Sigma \times \Gamma \times \{in,out\}}$ is the *embedding transformation*,
$Z \in EDG_{\Sigma,\Gamma}$ is the *starting graph* called the *axiom*.

Definition 3. Let $G = (\Sigma, \Delta, \Gamma, P, Z)$ be an *edNLC* graph grammar.
(1) Let $H, \overline{H} \in EDG_{\Sigma,\Gamma}$. Then H *directly derives* $\overline{H}$ in G, denoted by $H \underset{G}{\Longrightarrow} \overline{H}$, if there exists a node $v \in V_H$

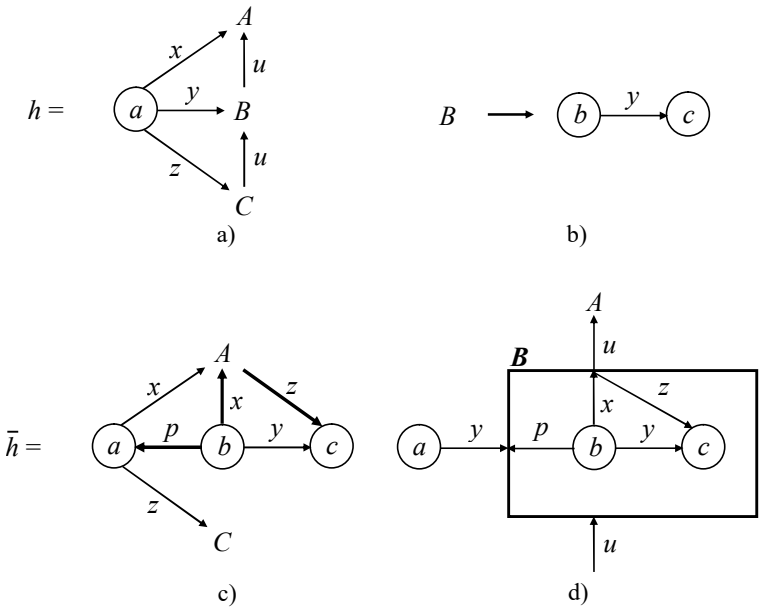

Fig. 1. An example of a derivation step in an** $edNLC$ **graph grammar.

and a production (l, D, C) in P such that the following holds.

(a) $l = \phi_H(v)$.

(b) There exists an isomorphism from $\overline{H}$ onto the graph X in $EDG_{\Sigma,\Gamma}$ constructed as follows. Let $\overline{D}$ be a graph isomorphic to D such that $V_H \cap V_{\overline{D}} = \emptyset$ and let h be an isomorphism from D onto $\overline{D}$. Then

$$X = (V_X, E_X, \Sigma, \Gamma, \phi_X),$$

where

$$V_X = (V_H \setminus \{v\}) \cup V_{\overline{D}},$$

$$\phi_X(y) = \begin{cases} \phi_H(y) & \text{if } y \in V_H \setminus \{v\} \\ \phi_{\overline{D}}(y) & \text{if } y \in V_{\overline{D}}, \end{cases}$$

$$\begin{gathered} E_X = (E_H \setminus \{(n,\gamma,m) : n = v \text{ or } m = v\}) \cup E_{\overline{D}} \\ \cup\{(n,\gamma,m) : n \in V_{\overline{D}}, m \in V_{X\setminus\overline{D}} \text{ and there exists an} \\ \text{edge } (m,\lambda,v) \in E_H \text{ such that} \\ (\phi_X(n), \phi_X(m), \gamma, out) \in C(\lambda, in)\} \cup \\ \cup\{(m,\gamma,n) : n \in V_{\overline{D}}, m \in V_{X\setminus\overline{D}} \text{ and there exists an} \\ \text{edge } (m,\lambda,v) \in E_H \text{ such that} \\ (\phi_X(n), \phi_X(m), \gamma, in) \in C(\lambda, in)\} \cup \\ \cup\{(n,\gamma,m) : n \in V_{\overline{D}}, m \in V_{X\setminus\overline{D}} \text{ and there exists an} \\ \text{edge } (v,\lambda,m) \in E_H \text{ such that} \\ (\phi_X(n), \phi_X(m), \gamma, out) \in C(\lambda, out)\} \cup \\ \cup\{(m,\gamma,n) : n \in V_{\overline{D}}, m \in V_{X\setminus\overline{D}} \text{ and there exists an} \\ \text{edge } (v,\lambda,m) \in E_H \text{ such that} \\ (\phi_X(n), \phi_X(m), \gamma, in) \in C(\lambda, out)\}. \end{gathered}$$

(2) By $\overset{*}{\underset{G}{\Longrightarrow}}$ we denote the transitive and reflexive closure of $\underset{G}{\Longrightarrow}$.

(3) The language of G, denoted $L(G)$, is the set

$$L(G) = \{H : Z \overset{*}{\underset{G}{\Longrightarrow}} H \ \ and \ H \in EDG_{\Delta,\Gamma}\}.$$

An example of a derivation step of an *edNLC* grammar is shown in Fig. 1.

The graph h which will be transformed is shown in Fig. 1a., whereas the left-hand side $l = A$ and the right-hand side D of a production are shown in Fig. 1b. Let us assume that the embedding transformation is defined as follows:

(i) $C(y, in) = \{(b, a, p, out)\}$,

(ii) $C(u, out) = \{(b, A, x, out), (c, A, z, in)\}$,

(iii) $C(u, in) = \emptyset$.

The derivation step is performed in two parts. During the first stage the node labelled with B of the graph h (corresponding to the left-hand side of the production) is removed, and the graph of the right-hand side replaces the removed node. The transformed graph obtained by removing the node (cf. $V_H \setminus \{v\}$ in Definition 3) and its adjacent edges (cf. $E_H \setminus \{(n,\gamma,m) : n = v \text{ or } m = v\}$ in Definition 3) is called the rest graph. During the second stage, the embedding transformation is used in order to connect certain nodes of the right-hand side graph with the rest graph. The item (i) is interpreted as follows:

1) Each edge labelled with y and coming in the node corresponding to the left-hand side of the production, i.e. B, shall be replaced by
2) the edge:
   a) connecting the node of the graph of the right-hand side of the production and labelled with b with the node of the rest graph and labelled with a,
   b) labelled with p,
   c) and going out from the node b.

Thus the item (i) of the embedding transformation generates the edge of the graph $\overline{h}$, shown in Fig. 1c, which is labelled with p and connects the nodes labelled b and a on the basis of the edge of the graph h labelled y and connecting the nodes labelled B and a (redirection and relabelling). The item (ii) duplicates an edge, and the item (iii) deletes an edge.

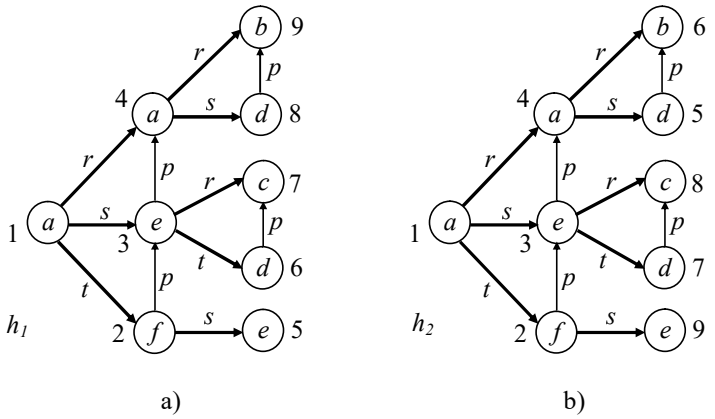

Fig. 2. An example of an IE graph (a) and an rIE graph (b).

In this paper *edNLC* productions are depicted according to the diagrammatical convention used in [45] (see Fig. 1d for the example production). The left-hand side is depicted with a box carrying its label in the upper left corner. The box contains the right-hand side graph. The area outside the box represents the environment of the right-hand side graph. The labelled arrows pointing to/from the box to the outside specify the domain of the embedding transformation. The labelled arrows which continue an outside arrow inside the box specify the embedding of this (outside) edge. Thus, the outside arrow can be re-established (with possible redirection/relabelling), duplicated (if continued by more than one arrow) or deleted (if not continued).

3. edNLC Graph Languages with Polynomial Membership Problem

As discussed in [19], there are two main reasons for the problems with constructing efficient parsing algorithms for graph languages (compared to the algorithms for string and tree languages) the lack of ordering of the graph structure and the complexity of the embedding transformation. Firstly, consider the ordering problem.

Note that the main concept of a reduction-based syntax analysis consists of analyzing the sentence/structure in order to identify consecutive subphrases/sub-structures (*handles*) that correspond to right-hand sides of the productions. Once a handle is identified, it is consumed, i.e. it is reduced to the left-hand side of the appropriate production. (In a top-down parse, handles have to be identified as well in order to find the appropriate production to be applied.) In the case of a graph structure, this means to look for a subgraph (a handle) that is isomorphic to a given graph, i.e. resolving the subgraph isomorphism problem, which is known to be NP-complete.

To resolve this problem we have introduced two subclasses of *EDG* graphs called *indexed edge-unambiguous graphs, IE graphs* [13, 15] and *reverse indexed edge-unambiguous graphs, rIE graphs* [23] in which a linear order on a set of nodes is defined. A transformation of an *EDG* graph into an *(r)IE* graph can be performed, if the former is an *interpreted graph* [23], i.e. it represents some real-world structure[2].

Now, let us introduce the way of indexing graph nodes, which has been used for defining the top-down parsable *ETPL*(k) graph grammars [19]. Let $G = (V, E, \Sigma, \Gamma, \phi)$ be an *EDG* graph, $V = \{v_1, v_2, \ldots, v_n\}$. We define a set of indices *Ind* $= \{1, 2, \ldots, n\}$ for V. G is called an indexed graph if indices of *Ind* have been ascribed to nodes of V with a bijective function.

Definition 4. Let H be an *EDG* graph over Σ and Γ. An *indexed edge-unambiguous graph*, *IE* graph, over Σ and Γ defined on the basis of the graph H is an *EDG* graph $G = (V, E, \Sigma, \Gamma, \phi)$ which is isomorphic to H up to the direction of the edges[3], such that the following conditions are fulfilled.
1. G contains a directed spanning tree T such that nodes of T have been indexed due to the Level Order Tree Traversal (LOTT)[4].
2. Nodes of G are indexed in the same way as nodes of T.
3. Every edge in G is directed from the node having a smaller index to the node having a greater index.

The *family of all the IE graphs over* Σ *and* Γ is denoted by $IE_{\Sigma,\Gamma}$.

An example of an IE graph h_1 is shown in Fig. 2a. The indices are ascribed to the graph nodes according to LOTT. The edges of the spanning tree T are thickened.

The way of indexing nodes according to LOTT is convenient if one uses a top-down parsing scheme [19]. In this paper reduction-based (bottom-up) parsable *ETPR*(k) graph grammars are characterized. The graphs generated by these grammars should be indexed according to a scheme that allows one to apply a reduction-based parsing scheme, i.e. the parser produces the rightmost derivation in reverse. (As it is made for Knuth's (string) *LR*(k) parsers [34].) Thus, we have to define a new traversal scheme for the tree spanned on an *EDG* graph. Such a scheme has been introduced in [23]. It is analogous to the LOTT (BFS) scheme, however it uses a LIFO queue (i.e. a stack) instead of a FIFO queue. We call it the *Reverse*

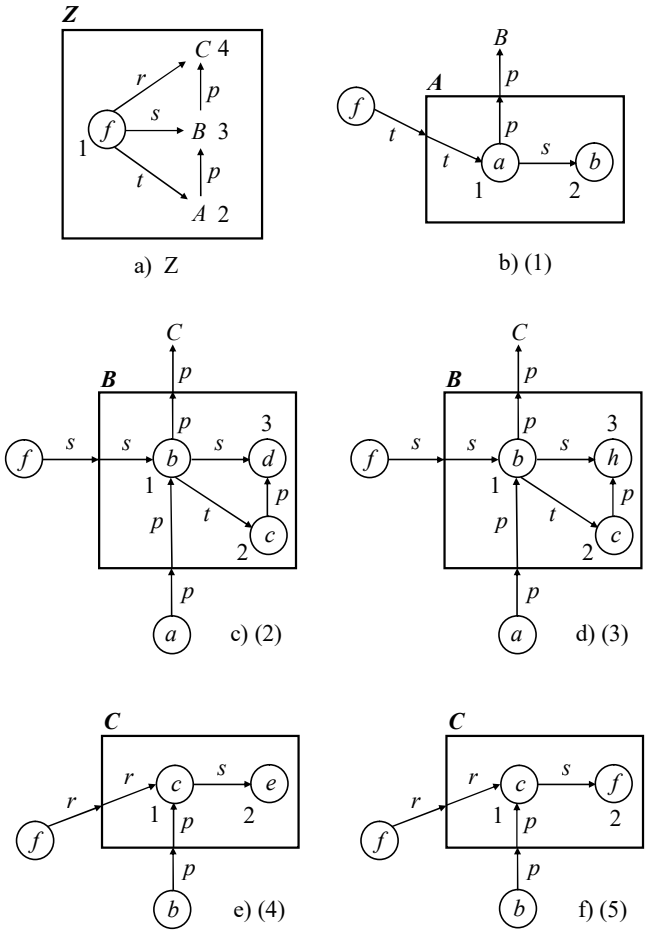

Fig. 3. An example of an** ETPL(k)-ETPR(k) **graph grammar

Level Order Tree Traversal (RLOTT). Now reversely indexed edge-unambiguous graphs can be defined.

Definition 5. Let H be an *EDG* graph over Σ and Γ. A *reversely indexed edge-unambiguous graph*, *rIE* graph, over Σ and Γ defined on the basis of the graph H is an *EDG* graph $G = (V, E, \Sigma, \Gamma, \phi)$ which is isomorphic to H up to the direction of the edges, such that the following conditions are fulfilled.
1. G contains a directed spanning tree T such that nodes of T have been indexed due to the Reverse Level Order Tree Traversal (RLOTT).
2. Nodes of G are indexed in the same way as nodes of T.
3. Every edge in G is directed from the node having a smaller index to the node having a greater index.

The *family of all the rIE graphs over* Σ *and* Γ is denoted by $rIE_{\Sigma,\Gamma}$.

An example of an rIE graph h_2 is shown in Fig. 2b. The indices are ascribed to the graph nodes according to RLOTT. The edges of the spanning tree T are thickened.

Let us introduce the notion of node level. We say that a node v of the *IE* (*rIE*) graph is on level n, if v is on level n of the spanning tree[5] T constructed as in Definition 4 (Definition 5).

We define the string-like graph representation of *IE* (*rIE*) graphs as in [19]. (This form of representation was originally defined for Ω graphs in [48].)

Definition 6. Let $v_k \in V$ be the node of an *IE* (*rIE*) graph $H = (V, E, \Sigma, \Gamma, \phi)$. A *characteristic description* of v_k is the quadruple $(\ a,\ r,\ (e_1 \dots e_r),\ (i_1 \dots i_r)\ )$, where a is the label of the node v_k, i.e. $\phi(v_k) = a$, r is the out-degree of v_k (the out-degree of the node designates the number of edges going out from this node), $(i_1 \dots i_r)$ is the string of node indices to which edges going out from v_k come (in increasing order), $(e_1 \dots e_r)$ is the string of edge labels ordered in such a way that the edge having the label e_x comes into the node having the index i_x.

For example,

$$(\ e,\ 3,\ (p\ t\ r),\ (4\ 6\ 7)\ )$$

is the characteristic description of the node indexed with 3 in the graph h_1 shown in Fig. 2a.

Definition 7. Let $H = (V, E, \Sigma, \Gamma, \phi)$ be an *IE* (*rIE*) graph, where $V = \{v_1, \dots, v_k\}$ is the set of nodes indexed such that v_i is indexed with i, $I(i), i = 1, \dots, k$ is the characteristic description of the node v_i. The string $I(1) \dots I(k)$ is called the *characteristic description* of the graph H.

Assuming a way of indexing of the graph h_1 from Fig. 2a as it has been defined above, we obtain the fol-

lowing characteristic description of this graph.

a	f	e	a	e	d	c	d	b
3	2	3	2	0	1	0	1	0
$(t\,s\,r)$	$(p\,s)$	$(p\,t\,r)$	$(s\,r)$	–	(p)	–	(p)	–
$(2\,3\,4)$	$(3\,5)$	$(4\,6\,7)$	$(8\,9)$	–	(7)	–	(9)	–

Now, the formal properties of the *ETPR*(k) reduction-based parsable subclass of *edNLC* languages can be presented. As this subclass will be compared with the *ETPL*(k) top-down parsable subclass of *edNLC* languages[6] in the next section, definitions for both classes must be introduced. Fortunately, most corresponding notions for both classes differ only slightly, so they may be formalized by single definitions with modifications. (The modifications for the *ETPR*(k) class are written in brackets in the definitions.)

Firstly, to reduce the computational complexity of a single step of the parsing algorithm the following constraint is imposed on the form of the right-hand side graphs of the productions.

Definition 8. Let $G = (\Sigma, \Delta, \Gamma, P, Z)$ be an *edNLC* graph grammar. The grammar G is called a *TLP graph grammar*, abbrev. from *Two-Level Production*, if the following conditions are fulfilled.
1. P is a finite set of productions of the form (l, D, C), where : (a) $l \in \Sigma \backslash \Delta$,
(b) D is the *IE* (*rIE* for the *ETPR*(k) class) graph having the characteristic description :

$n_1(1)$	$n_2(2)$	$\ldots$	$n_m(m)$	*or*	$n_1(1)$,	$n_i(i)$
r_1	r_2	$\ldots$	r_m		0	r_i
E_1	E_2	$\ldots$	E_m		–	E_i
I_1	I_2	$\ldots$	I_m		–	I_i

is a characteristic description of the node $i,\ i = 1, \ldots, m, n_1 \in \Delta$ (i.e. n_1 is a terminal label) and $i,\ i = 2, \ldots, m$ are nodes on level 2,
(c) $C : \Gamma \times \{in, out\} \longrightarrow 2^{\Sigma \times \Sigma \times \Gamma \times \{in,out\}}$ is the embedding transformation.
2. Z is an *IE* (*rIE*) graph such that its characteristic description satisfies the condition defined in point 1(b).

An example of a *TLP* grammar is shown in Fig. 3.

Now, we will introduce restrictions on the derivation process, i.e. on the embedding transformation. The *NLC*-like embedding transformation operates at the border between the left- and right-hand sides of a production and their context. Thus, we do not have the important *context freeness* property stated that reordering of the derivation steps does not influence the result of the derivation. The lack of the order-independence property, related to the *finite Church-Rosser, fCR,* property (non-overlapping steps can be done in any order), results in the intractability of the parsing. Therefore, the power of the *NLC*-like embedding transformation must be limited in order to obtain the *fCR* property and to guarantee efficiency of parsing. For example, in *boundary NLC* graph grammars, defined by Rozenberg and Welzl [46], nonterminal nodes cannot be adjacent (in right-hand side graphs and in the axiom). In our model [13, 16] we limit the power of the embedding transformation in the following way. Firstly, we require that all graphs in a derivation are (r)IE graphs. In fact, this requirements restricts the embedding transformation, which cannot redirect edges. Secondly, we require that a derivation process is performed according to the linear ordering imposed on *IE* (*rIE*) graphs[7]. It is also assumed [13, 14, 16] that during a derivation step, the root of the right-hand side inherits the index from the replaced node (corresponding to the left-hand side) and the remaining nodes of the right-hand side get the next available indices.

Definition 9. A TLP graph grammar G is called a *closed TLP* (*rTLP*) *graph grammar* G if for each derivation of this grammar

$$Z = g_0 \underset{G}{\Longrightarrow} g_1 \underset{G}{\Longrightarrow} \ldots \underset{G}{\Longrightarrow} g_n$$

each graph $g_i, i = 0, \ldots, n$ is an *IE* (*rIE*) graph.

Definition 10. Let there be given a derivation of a closed *TLP* (*rTLP*) graph grammar G:

$$Z = g_0 \underset{G}{\Longrightarrow} g_1 \underset{G}{\Longrightarrow} \ldots \underset{G}{\Longrightarrow} g_n.$$

This derivation is called a *regular left-hand* (*right-hand*) *side derivation*, denoted $\underset{rl(G)}{\Longrightarrow}$ ($\underset{rr(G)}{\Longrightarrow}$) if :
(1) for each $i = 0, \ldots, n-1$ a production for a nonterminal node having the least (greatest) index in a graph g_i is applied,
(2) node indices do not change during a derivation.
A closed TLP (rTLP) graph grammar which rewrites graphs according to the regular left-hand (right-hand) side derivation is called a *closed TLPO* (*rTLPO*) *graph grammar*, abbrev. from (*reverse*) *Two-Level Production-Ordered*.

In order to achieve the requirements imposed by Definitions 9 and 10, the embedding transformation C of each production (l, D, C) should fulfil the following conditions.
1. C has to re-introduce (without re-directing) the incoming edge belonging to the spanning tree T of the derived *(r)IE* graph (cf. Definitions 4 and 5).
2. Any other incoming edge can be re-introduced and duplicated without re-directing. It can also be deleted.
3. An outcoming edge can be:
(a) deleted,
(b) re-introduced without re-directing,
(c) used for generating new edges coming into nodes of level 2 of the right-hand side[8].

Let us define the concepts used for extracting handles in the analyzed graphs which are matched against the right-hand sides of productions during the graph parsing. These concepts will be used for the *ETPL*(k) class as well as for the *ETPR*(k).

Definition 11. Let g be an *IE* (*rIE*) graph, l the index of some node of g defined by a characteristic description

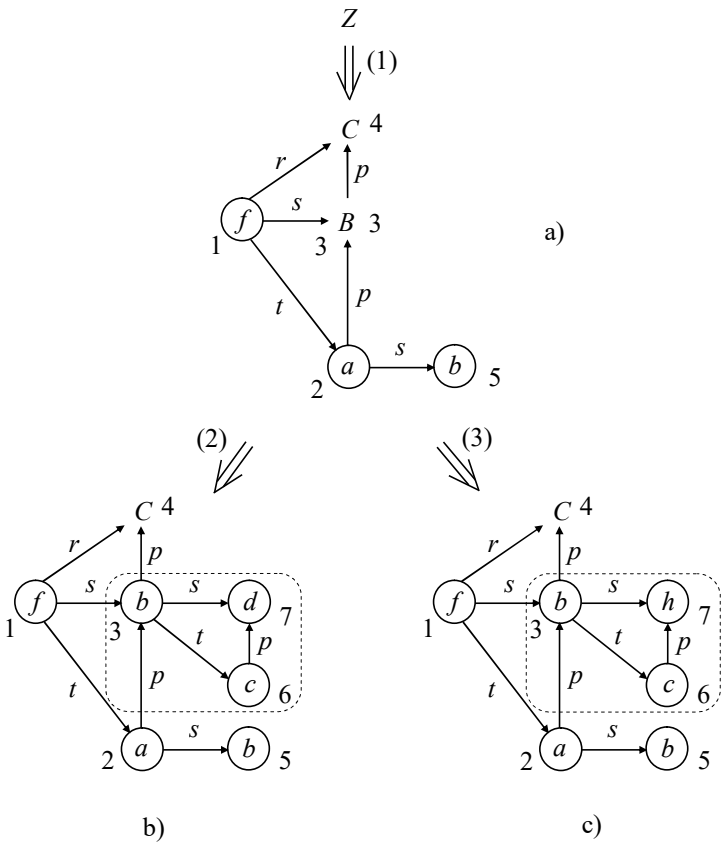

Fig. 4. An example of an** $ETPL(k)$ **derivation

$(n, r, e_1 \dots e_r, i_1 \dots i_r)$. A subgraph h of the graph g consisting of the node indexed with l, nodes having indices $i_{a+1}, i_{a+2}, \dots, i_{a+m}, a \geq 0, a + m \leq r$, and edges connecting the nodes indexed with: $l, i_{a+1}, i_{a+2}, \dots, i_{a+m}$ is called an *m-successors handle*, denoted $h = m - TL(g, l, i_{a+1})$. By $0 - TL(g, l, -)$ we denote the subgraph of g consisting only of the node indexed with l.

If the subgraph h of the graph g from Definition 11 consists of the node indexed with l, nodes having indices $i_{a+1}, i_{a+2}, \dots, i_r, a \geq 0$, and edges connecting the nodes indexed with: $l, i_{a+1}, i_{a+2}, \dots, i_r$, then it is denoted $h = CTL(g, l, i_{a+1})$.

Now, the fundamental constraint which is analogous to that used in a definition of string $LL(k)$ grammars can be imposed. This constraint allows an efficient, non-backtracking, *top-down* parsing scheme for *edNLC* grammars to be constructed. In order to introduce the idea of this scheme in an intuitive way, an $LL(k)$ grammar [36, 44] is defined.

Let $G = (\Sigma, \Delta, P, S)$, Σ a set of symbols, Δ a set of terminal symbols, P a set of productions and S the starting symbol, be a context-free grammar. Let $\eta \in \Sigma^*$, and $|x|$ denote the length of the string $x \in \Sigma^*$. $FIRST_k(\eta)$ denotes a set of all the terminal prefixes of strings of length k (or less than k, if a terminal string shorter than k is derived from α) that can be derived from η in the grammar G[9], i.e.

$$FIRST_k(\eta) = \{x \in \Delta^* : (\eta \underset{G}{\overset{*}{\Longrightarrow}} x\beta \wedge |x| = k) \vee (\eta \underset{G}{\overset{*}{\Longrightarrow}} x \wedge |x| < k),\ \beta \in \Sigma^*\}.$$

Let $\underset{l(G)}{\overset{*}{\Longrightarrow}}$ denote a *leftmost derivation* in G, that is a derivation such that a production is always applied to the leftmost nonterminal[10], $N = \Sigma \setminus \Delta$ be a set of nonterminal symbols.

Definition 12. Let $G = (\Sigma, \Delta, P, S)$ be a context-free grammar. The grammar G is called an $LL(k)$ *grammar* if for every two leftmost derivations

$$S \underset{l(G)}{\overset{*}{\Longrightarrow}} \alpha A \delta \underset{l(G)}{\Longrightarrow} \alpha\beta\delta \underset{l(G)}{\overset{*}{\Longrightarrow}} \alpha x$$
$$S \underset{l(G)}{\overset{*}{\Longrightarrow}} \alpha A \delta \underset{l(G)}{\Longrightarrow} \alpha\gamma\delta \underset{l(G)}{\overset{*}{\Longrightarrow}} \alpha y,$$

where $\alpha, x, y \in \Delta^*,\ \beta, \gamma, \delta \in \Sigma^*,\ A \in N,$ the following condition holds.

$$\text{If } FIRST_k(x) = FIRST_k(y) \text{ then } \beta = \gamma.$$

The $LL(k)$ condition means that for any step during a derivation of a string $w \in \Delta$ in G, we can choose a production in an unambiguous way on the basis of an analysis of some part of w which is of length at most k. We can say that an $LL(k)$ grammar has the property of an unambiguous choice of a production given the k-length prefix in the leftmost derivation. Now, by analogy, we define a $PL(k)$ graph grammar which has the property of an unambiguous choice of a production given the $k - TL$ graph in the regular left-hand side derivation.

Definition 13. Let $G = (\Sigma, \Delta, \Gamma, P, Z)$ be a closed TLPO graph grammar. The grammar G is called a

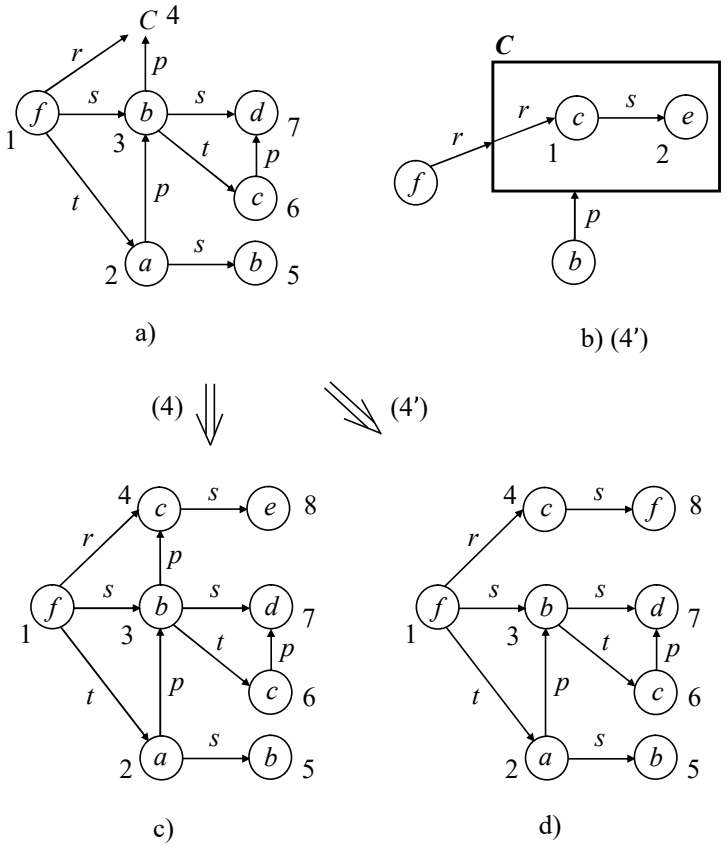

Fig. 5. An example of preserving a potential previous context.

PL(k), abbrev. *Production-ordered k-Left nodes unambiguous, graph grammar* if the following condition is fulfilled. Let

$$Z \underset{rl(G)}{\overset{*}{\Longrightarrow}} X_1AX_2 \underset{rl(G)}{\Longrightarrow} g_1 \underset{rl(G)}{\overset{*}{\Longrightarrow}} h_1$$

and

$$Z \underset{rl(G)}{\overset{*}{\Longrightarrow}} X_1AX_2 \underset{rl(G)}{\Longrightarrow} g_2 \underset{rl(G)}{\overset{*}{\Longrightarrow}} h_2\,,$$

where $\underset{rl(G)}{\overset{*}{\Longrightarrow}}$ is the transitive and reflexive closure of $\underset{rl(G)}{\Longrightarrow}$, be two regular left-hand side derivations, such that A is a characteristic description of a node indexed with l, and X_1 and X_2 are characteristic descriptions of subgraphs. Let max be a number of nodes of the graph X_1AX_2. If

$$k - TL(h_1, l, max + 1) \cong k - TL(h_2, l, max + 1)$$

then

$$CTL(g_1, l, max + 1) \cong CTL(g_2, l, max + 1).$$

For example, our graph grammar shown in Fig. 3 is *PL*(2). As we can see in Fig. 4 in order to identify which production has been applied to a node indexed with 3, we have to analyze $2 - TL$ graphs originated in this node. ($1 - TL$ graphs for productions 2 and 3 are the same.)

For defining reduction-based (bottom-up) parsable graph grammars we have used the same methodology as in the case of top-down parsable grammars. That is, we have imposed a constraint which is analogous to that used in the definition of Knuth's string *LR*(k) grammars [34] allowing us to construct an efficient, non-backtracking, *bottom-up* parsing scheme for *edNLC* grammars. Therefore, we firstly define an *LR*(k) grammar.

Let $\underset{r(G)}{\overset{*}{\Longrightarrow}}$ denote a *rightmost derivation* in G, that is a derivation such that a production is always applied to the rightmost nonterminal[11]. A string which occurs in the rightmost derivation of some sentence is called a *right-sentential form*.

Definition 14. Let $G = (\Sigma, \Delta, P, S)$ be a context-free grammar. The grammar G is called an *LR*(k) *grammar* if for every two rightmost derivations

$$S \underset{r(G)}{\overset{*}{\Longrightarrow}} \alpha Aw \underset{r(G)}{\Longrightarrow} \alpha\beta w$$
$$S \underset{r(G)}{\overset{*}{\Longrightarrow}} \gamma By \underset{r(G)}{\Longrightarrow} \alpha\beta x,$$

where $w, x, y \in \Delta^*$, $\alpha, \beta, \gamma \in \Sigma^*$, $A, B \in N$, the following condition holds.

If $FIRST_k(w) = FIRST_k(x)$ then $\alpha = \gamma$, $A = B$, $x = y$.

The *LR*(k) condition means that for each right-sentential form we can identify a handle (i.e. the right-hand side of some production) and we can choose a production in an unambiguous way[12] by looking at most k symbols beyond the handle. We can say that an *LR*(k) grammar has a property of both the identification of a handle and an unambiguous choice of a production given k symbols ahead in a right-sentential form. Now, by analogy, we define a *PR*(k) graph grammar, which has the property of both

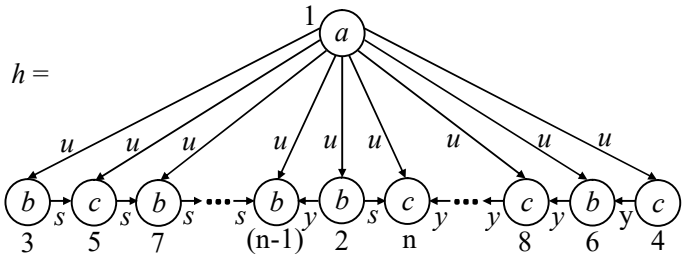

Fig. 6. A graph h of the language L_0 that cannot be generated by any $ETPR(k)$ grammar

an identification of a handle and an unambiguous choice of a production given a $k-TL$ graph beyond the handle in a regular right-hand side derivation.

Definition 15. Let $G=(\Sigma,\Delta,\Gamma,P,Z)$ be a closed rT-LPO graph grammar. The grammar G is called a *PR(k)*, abbrev. *Production-ordered k-Right nodes unambiguous, graph grammar* if the following condition is fulfilled. Let

$$Z \underset{rr(G)}{\overset{*}{\Longrightarrow}} X_1AX_2 \underset{rr(G)}{\Longrightarrow} X_1gX_2\,,$$
$$Z \underset{rr(G)}{\overset{*}{\Longrightarrow}} X_3BX_4 \underset{rr(G)}{\Longrightarrow} X_1gX_5\,,$$

and

$$k-TL(X_2,1,2)\cong k-TL(X_5,1,2)\,,$$

where $\underset{rr(G)}{\overset{*}{\Longrightarrow}}$ is the transitive and reflexive closure of $\underset{rr(G)}{\Longrightarrow}$, A, B are characteristic descriptions of certain nodes, X_1, X_2, X_3, X_4, X_5 are characteristic descriptions of subgraphs, g is the right-hand side of a production: $A\longrightarrow g$.
Then:

$$X_1=X_3\,,\ A=B\,,\ X_4=X_5\,.$$

The last restriction that has to be imposed concerns the embedding transformation. We have already introduced limitations for the embedding transformation which guarantee that all graphs during a derivation are *IE* (*rIE*) graphs and that node indices do not change during a derivation (Definitions 9 and 10). Nevertheless, these conditions do not guarantee that during parsing the characteristic description of a node does not change (e.g. after its analysis by a parser). Of course, it is an unwanted effect. For example, let us modify the definition of production 4 of our grammar shown in Fig. 3e. A modified production (4') is shown in Fig. 5b. The results of applying productions 4 and 4' to a graph shown in Fig. 5a are shown in figures 5c and 5d, respectively. One can easily notice that during parsing with the modified grammar, after analyzing a node indexed with 3, its characteristic description changes, because the embedding transformation of production 4' does not re-introduce an edge labelled with p. We will claim such edges need to be re-introduced. Let us also notice that the issue concerns only edges incoming to the root of the right-hand side, since they have already been analyzed by the parser. (If the embedding transformation for the root node v does not re-introduce an edge outgoing from v, then the parser, analyzing the handle originated at v, "sees" such a situation.)

Definition 16. Let $G=(\Sigma,\Delta,\Gamma,P,Z)$ be a PL(k) (PR(k)) graph grammar. A pair $(b,x), b\in\Delta, x\in\Gamma$, is called a *potential previous context* for a node label $a\in\Sigma\backslash\Delta$, if there exists the *IE* (*rIE*) graph $g=(V,E,\Sigma,\Gamma,\phi)$ belonging to a certain regular left-hand (right-hand) side derivation in G such that : $(k,x,l)\in E, \phi(k)=b$ and $\phi(l)=a$.

Definition 17. A PL(k) (PR(k)) graph grammar $G=(\Sigma,\Delta,\Gamma,P,Z)$ is called an *ETPL*(k) (*ETPR*(k)), abbrev. from *Embedding Transformation - preserving Production-ordered k-Left (k-Right) nodes unambiguous, graph grammar* if for each production $(l,D,C)\in P$ the following condition is fulfilled.
Let $l=A, X_1, X_2, \ldots, X_m$, where $X_i\neq X_j, i,j=1,\ldots,m$, be labels of nodes indexed with $1,2,\ldots,m$ of the right-hand side graph D. For each potential previous context (b,y) for A, there exists $(X_i,b,z,in)\in C(y,in), i\in\{1,\ldots,m\}$. If $i=1$, then $z=y$, i.e. $(X_1,b,y,in)\in C(y,in)$.

A parsing algorithm, $O(n^2)$, for *ETPR*(k) graph grammar was defined in [23]. It is a slight modification of the parsing scheme for *ETPL*(k) graph grammar [16].

4. Generative power of ETPR(k) graph languages

In this section the generative power of *ETPR*(k) graph languages is characterized in an analogous way as was made for *ETPL*(k) graph languages in [19]. Finally, two theorems concerning both classes of languages are proved.

Let X denote a class of graph grammars. Then $\mathcal{L}(X)$ denotes a set of graph languages such that there exists an X grammar G and $L=L(G)$.
Additionally, we say that a language L is *ETPL*(k) (*ETPR*(k)), if there exists an *ETPL*(k) (*ETPR*(k)) grammar G such that $L=L(G)$.

Firstly, we will show that the class of *ETPR*(k) languages is a proper subclass of the class of *edNLC* languages. Comparing the generative power of both classes, we are interested in their intrinsic properties, which do not result from assuming the *specific* indexing for graphs as in the case of *ETPR*(k) languages. (Since, obviously, any "ordered" version of a class of

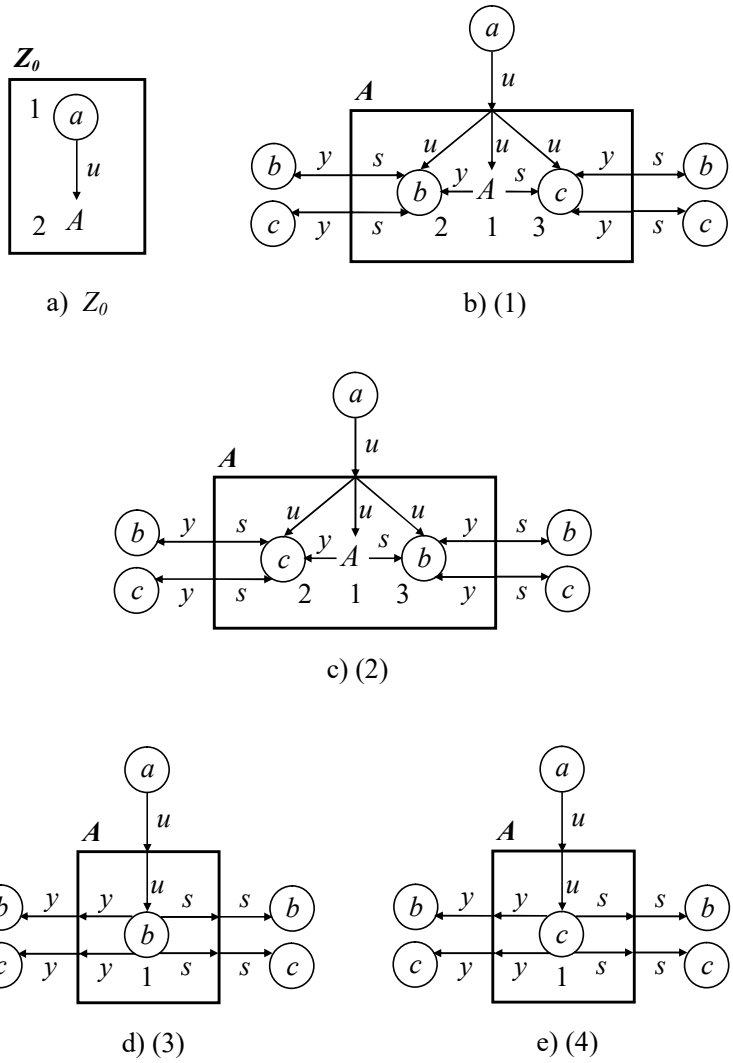

Fig. 7. The axiom and the productions of the $edNLC_o$ ***grammar*** G_0***.***

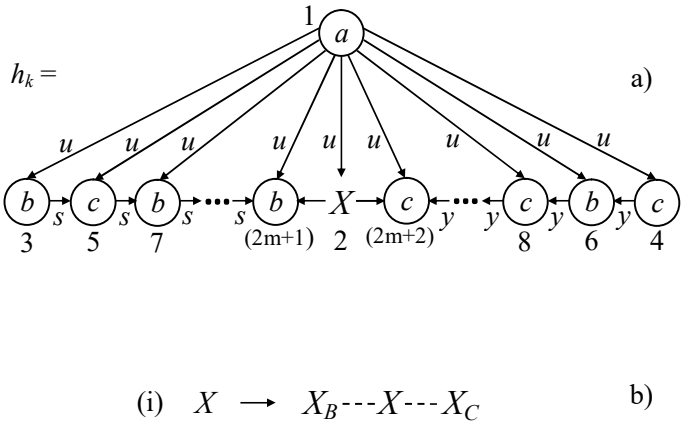

Fig. 8. The hypothetical graph and the production of an $ETPR(k)$ ***grammar.***

graph languages is its "subfamily".) Therefore, to compare the essential generative power of both families an ordered version of "pure" *edNLC* languages will be defined.

Let $H \in EDG_{\Sigma,\Gamma}$. Then $H_{[o]}$ denotes a graph obtained from H by indexing its nodes and re-directing (if necessary) its edges in such a way that $H_{[o]}$ fulfils the conditions of Definition 5, i.e. $H_{[o]}$ is an *rIE* graph. A grammar $\overline{G} = (\Sigma, \Delta, \Gamma, P, Z)$ is called an ordered *edNLC* grammar corresponding to G, denoted *edNLC*$_o$, if

$L(\overline{G}) = \{H_{[o]} : Z_{[o]} \underset{rr(G)}{\Longrightarrow} H^1_{[o]} \underset{rr(G)}{\Longrightarrow} H^2_{[o]} \overset{*}{\underset{rr(G)}{\Longrightarrow}} H^m_{[o]} = H_{[o]}$ and $H \in EDG_{\Delta,\Gamma}\}$.

Theorem 1. *For any* $k \geq 0$

$$\mathcal{L}(ETPR(k)) \subsetneq \mathcal{L}(edNLC_o) .$$

Proof. PART 1: $\mathcal{L}(ETPR(k)) \subseteq \mathcal{L}(edNLC_o)$.

Let $L \in \mathcal{L}(ETPR(k))$, i.e. there exists an $ETPR(k)$ grammar G such that $L = L(G)$. We should show that $L \in \mathcal{L}(edNLC_o)$, i.e. that there exists an $edNLC_o$ grammar $\overline{G}$ such that $L = L(G) = L(\overline{G})$.

One can easily note that setting $\overline{G} := G$ is sufficient, because any $ETPR(k)$ grammar is also an $edNLC_o$ grammar.

PART 2: $\mathcal{L}(ETPR(k)) \neq \mathcal{L}(edNLC_o)$.

We will define a language $L_0 \in \mathcal{L}(edNLC_o)$ that cannot be generated by any $ETPR(k)$ graph grammar. Let us introduce a language which is of the complementary palindromic form. In case of strings, a complementary palindrome is a sequence of symbols which reads in reverse as the complement of the forward sequence. It means that for each symbol its complementary symbol has to be defined. For example, in DNA a symbol A is complementary to T, and C is complementary to G. Thus, for example the DNA sequence $GGCATGCC$ is a complementary palindrome.

Let L_0 consist of *rIE* graphs of the complementary palindromic form as the graph h shown in Fig. 6. Let us assume that a node label c is complementary to a node label b. The graph h is "divided" with the edge $(1, u, 2)$. A string of node labels of a "path" on the right-hand side of this edge (without a node indexed with 2) is a complementary palindrome of a string of node labels of the left-hand side "path" (also without a node indexed with 2). That is, for any n-node graph $h = (V, E, \Sigma, \Gamma, \phi) \in L$, n - an even number, the following holds. $\phi(1) = a$; $\phi(2) = b$ or $\phi(2) = c$; for an odd index $k = 3, 5, \ldots, (n-1)$: if $\phi(k) = b$ then $\phi(k+1) = c$, and if $\phi(k) = c$ then $\phi(k+1) = b$. $E = \{(1, u, i), i = 2, \ldots, n\} \cup \{(2, y, (n-1)\}, (2, s, n)\} \cup \{(k, s, (k+2)), k = 3, 5, 7, \ldots, (n-3)\} \cup \{(k, y, (k+2)), k = 4, 6, 8, \ldots, (n-2)\}$. We will call L_0 the *complementary palindromic graph language*.

Now, we define an $edNLC_o$ grammar $G_0 = (\Sigma_0, \Delta_0, \Gamma_0, P_0, Z_0)$ generating the language L_0. (Without loss of generality we assume that during derivation all nodes indexed with $k = 3, 4, \ldots, n$ are generated directly as terminal nodes, i.e. not *via* nonterminal nodes.)

$\Sigma_0 = \{a, b, c, A\}$,
$\Delta_0 = \{a, b, c\}$,
$\Gamma_0 = \{s, u, y\}$,
P_0 and Z_0 are shown in Fig. 7.

Now, we will show that L_0 cannot be generated by any $ETPR(k)$ grammar. Let us assume, proving by contradiction, that there exists an $ETPR(k)$ grammar G_1 which generates L_0. Then, let us assume that we generate the n-node graph h, $n \geq 6$, belonging to L_0. Let $\mathcal{D}$ be the following derivation of h:

$$Z_o \underset{rr(G)}{\Longrightarrow} h_1 \overset{*}{\underset{rr(G)}{\Longrightarrow}} h_k \underset{rr(G)}{\Longrightarrow} h_{k+1} \overset{*}{\underset{rr(G)}{\Longrightarrow}} h_r = h.$$

Let us assume that the graph h_k has $(2m+2)$ nodes and $n \geq 2m+6$, i.e. we have still to generate at least four nodes. Let us assume also that h_{k+1} has more than $(2m+2)$ nodes, i.e. new nodes are generated during $h_k \underset{rr(G)}{\Longrightarrow} h_{k+1}$ of $\mathcal{D}$.

Note that the graph h_k has to be of the form shown in Fig. 8a. The form of h_k results from the following facts. All the nodes of h_k, except for the node indexed with 2, have to be labelled with terminals. (According to Definition 10 we apply a production to a nonterminal node having the greatest index.) The graph h_k has to be of the proper, i.e. complementary palindromic, form. That is, the right-hand side "path" has to be a complementary palindrome of the left-hand side "path", because we use a context-free graph grammar that does not possess the mechanisms allowing one to take into account previous derivation steps (and the terminal "context" already derived) in a further derivation process. It means that a graph derived cannot be rectified later, if it does not conform to the complementary palindromic form. Ascribing indices to nodes of h_k has also to be definitive since according to Definition 10 node indices do not change during a derivation. The edges have to be directed as in Fig. 8a according to Definition 5. Obviously, the labels of edges connecting terminal nodes have to be definitive.

At the step $h_k \underset{rr(G)}{\Longrightarrow} h_{k+1}$ of $\mathcal{D}$ we have to generate two new nodes simultaneously because of a palindromic-like structure of h. Let us assume that the node indexed with $(2m+3)$ of h is labelled with b and the node indexed with $(2m+4)$ of h is labelled with c. (For the opposite labelling reasoning is analogous.) The production of G_1 which is to be applied for generating the succeeding pair of complementary nodes has to be of the form shown in Fig. 8b, where

- X is a nonterminal node used for generating the succeeding pairs of complementary nodes in further derivation steps,
- X_B is a terminal node labelled with b or X_B is a nonterminal node and the production replacing X_B with a terminal node labelled with b belongs to G_1, and
- X_C is a terminal node labelled with c or X_C is a nonterminal node and the production replacing X_C with a terminal node labelled with c belongs to G_1.

Let us note that according to Definition 8 the right-hand side graph of the production **(i)** has to be a two-level graph. Moreover, the root of the right-hand side has to inherit the index from the replaced node. Thus, the node X has to be the root of the right-hand side. However, it is contrary to the condition of Definition 8 saying that the root of the right-hand side has to be a terminal node. *Q.E.D.*

The parameter k in definitions of both $ETPR(k)$ and $ETPL(k)$ graph grammars has shown to be very useful from a practical point of view in many applications of these grammars e.g.: robotics [13], distributed systems [25], CAD/CAM [18, 22], industrial-like control [2, 17], CAE (FEM computing) [27], sign language recognition [21, 26], computer vision [24]. In [19] we have proved that by increasing this parameter we strengthen the generative power of $ETPL(k)$ grammars. By proving the following theorem, which establishes the hierarchy of $ETPR(k)$ grammars, we show that the same holds for the $ETPR(k)$ class.

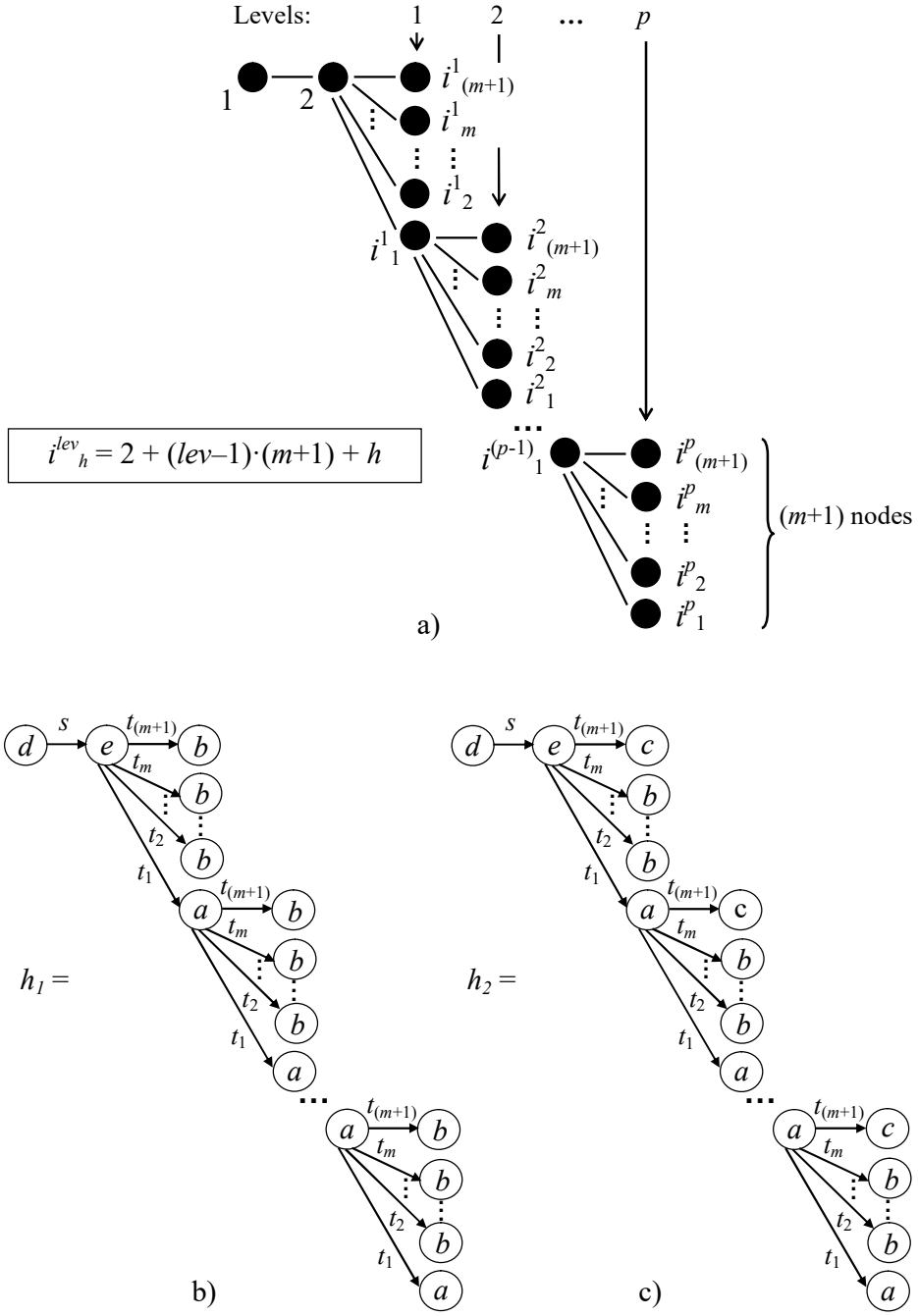

Fig. 9. The language that cannot be generated by any $ETPR(m)$ ***grammar***

Theorem 2. *For a given* $k \geq 0$

$$\mathcal{L}(ETPR(k)) \subsetneq \mathcal{L}(ETPR(k+1)) \ .$$

Proof. PART 1: $\mathcal{L}(ETPR(k)) \subseteq \mathcal{L}(ETPR(k+1))$.
Let G be an $ETPR(k)$ grammar. One should define such an $ETPR(k+1)$ grammar $\overline{G}$ that $L(G) = L(\overline{G})$.

Let us note that it is sufficient to set : $\overline{G} = G$.

PART 2. $\mathcal{L}(ETPR(k)) \neq \mathcal{L}(ETPR(k+1))$.
Let us take any $k = m$. We define an $ETPR(m+1)$ language L which cannot be generated by any $ETPR(m)$ grammar. Let $L = L_1 \cup L_2$. The *rIE* graphs belonging to both L_1 and L_2 are of the cascade-like form shown in Fig. 9a. Firstly, let us define this cascade-like structure. Each n-node graph $h = (V, E, \Sigma, \Gamma, \phi) \in L$, consists of: two nodes indexed with 1 and 2 such that $\phi(1) = d$, $\phi(2) = e$, $(1, s, 2) \in E$, and p levels, $p = 1, 2, \ldots$, assuming that it has at least two levels. Each level consists of $(m+1)$ nodes indexed as shown in Fig. 9a. If we denote the hth node of the level lev with v^{lev}_h, then its index $i^{lev}_h = 2+(lev-1)\cdot(m+1)+h$. A set of edges E contains additionally the following edges (cf. Fig. 9):
- $(2, t_1, i^1_1), (2, t_2, i^1_2), \ldots, (2, t_{(m+1)}, i^1_{(m+1)})$,
- for each level $l = 2, \ldots, p$:
$(i^{(l-1)}_1, t_1, i^l_1), (i^{(l-1)}_1, t_2, i^l_2), \ldots, (i^{(l-1)}_1, t_{(m+1)}, i^l_{(m+1)})$.

Now, we define the way of labelling the graph nodes belonging to levels $l = 1, \ldots, p$ (cf. Figs. 9b and c.)
- If $h_1 \in L_1$, then $\phi(i^l_1) = a$, for $h = 2, \ldots, m : \phi(i^l_h) = b$, and $\phi(i^l_{(m+1)}) = b$.
- If $h_2 \in L_2$, then $\phi(i^l_1) = a$, for $h = 2, \ldots, m : \phi(i^l_h) = b$, and $\phi(i^l_{(m+1)}) = c$.

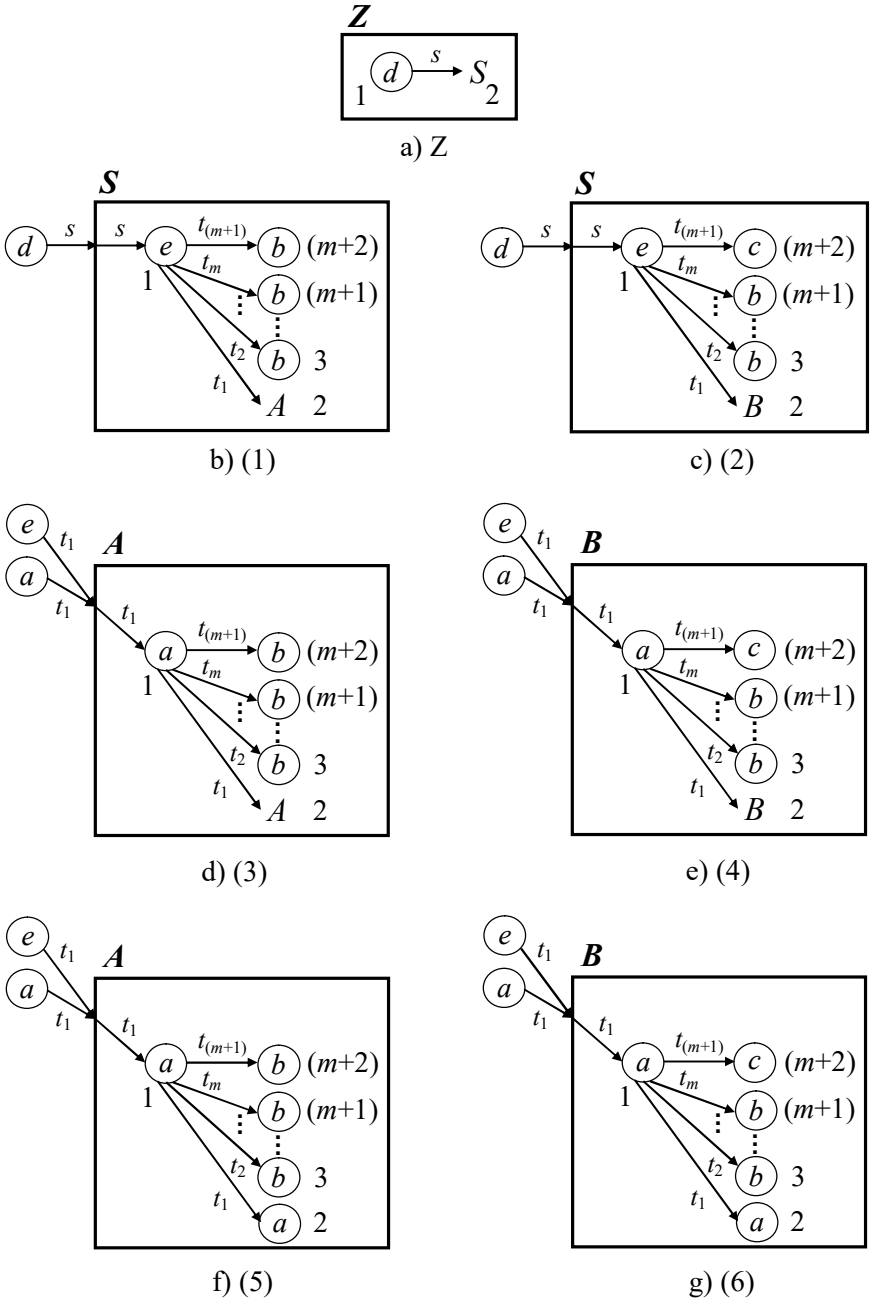

Fig. 10. The axiom and the productions of the $ETPR(m+1)$ ***grammar*** G

We will call L the *(m+1)-height-step cascade graph language*.

Now, let us define an $ETPR(m+1)$ grammar $G = (\Sigma, \Delta, \Gamma, P, Z)$ which generates a language L. $\Sigma = \{a, b, c, d, e, S, A, B\}, \Delta = \{a, b, c, d, e\}, \Gamma = \{s, t_1, t_2, \ldots, t_{(m+1)}\}$, P and Z are shown in Fig. 10.

It can be easily noted that to generate a graph $h_1 \in L_1$ having l levels one has to apply production 1 once, production 3 l times, and production 5 once, and to generate a graph $h_2 \in L_2$ having l levels one has to apply production 2 once,production 4 l times, and production 6 once. The grammar G obviously does not generate any graphs not belonging to L. Thus, $L = L(G)$.

Now, we show that G is the only grammar of the *ETPR* class which generates L.

Firstly, let us note that graphs belonging to L are, in fact, trees.

Secondly, according to Definition 8 the right-hand side graph of any production in an *ETPR* grammar has to be a graph of level at most 2. A node of the right-hand side graph indexed with 1 has to be terminal. Then, a node of the right-hand side graph indexed with 2 of the productions used for developing succeeding levels[13] has to be nonterminal[14].

Thirdly, let us note that a higher level of any graph belonging to L has to be generated at one derivation step, i.e. the right-hand sides of productions used for developing succeeding levels have to be two-level trees having $(m+1)$ children[15]. To show it, let us assume, proving indirectly, that some level p can be generated in stages. It means that at the first stage we generate a subtree having k children, $k < m+1$, indexed with: $i_1^p, \ldots, i_k^p$ (cf. Fig. 9a). Now, on the basis of a node indexed with i_k^p we have to generate the

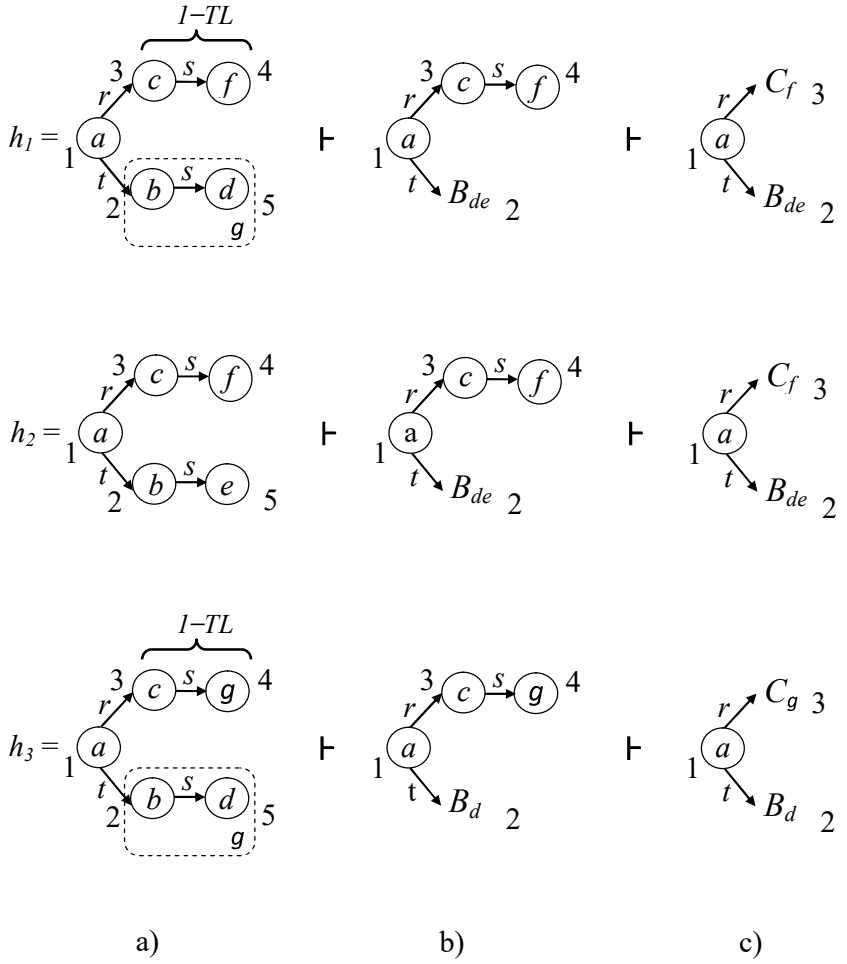

Fig. 11. The language that cannot be generated by any $ETPL(k)$ ***grammar***

next child, i.e. $i^p_{(k+1)}$ with some production $\overline{p}$. We have to use the embedding transformation of $\overline{p}$ to connect the newly-generated child with the parent, i.e. to generate an edge $(i_1^{(p-1)}, t_{(k+1)}, i^p_{(k+1)})$ (cf. Fig. 9). However, let us note that we will also have to destroy an edge connecting nodes i^p_k and $i^p_{(k+1)}$ in a further derivation, which is forbidden by the principle of preserving a potential previous context (cf. Definition 17).

On the other hand, G is not an *ETPR*(m) grammar. During a derivation of any $h \in L$, in spite of the fact that $m - TL$ graphs described by Definition 15 are isomorphic, the corresponding right-hand side graph (the handle) can be reduced to various left-hand side nonterminals. For example, $m - TL$ right-hand side graphs of productions 5 and 6 are isomorphic, however, these productions reduce to different nonterminals A and B[16]. *Q.E.D.*

At the end of this section we show that both classes *ETPL* and *ETPR* are incomparable.

Theorem 3. *There exists*

$$L \in \mathcal{L}(ETPR(1))$$

such that for any $k \geq 0$

$$L \notin \mathcal{L}(ETPL(k)).$$

Proof. In [19] (cf. Theorem 4, [19]) we have defined a language L which cannot be generated by any *ETPL*(k), $k \geq 0$ grammar. The language L consists of three graphs h_1, h_2, and h_3 shown in Fig. 11a. If the upper path finishes with a node labelled f, then the lower path can finish with a node labelled either d or e. However, if the upper path finishes with a node labelled g, then the lower path can finish only with a node labelled d. We will call L a *third-level contextual graph language*, since contextual dependencies between pairs of node labels occur at the third level of a graph.

We will define an *ETPR*(1) grammar G which generates the language L. In Figs. 11b and c we have shown the proper reductions during the bottom-up parsing. They help us to define the following grammar $G = (\Sigma, \Delta, \Gamma, P, Z)$.
$\Sigma = \{a, b, c, d, e, f, g, B_{de}, B_d, C_f, C_g, S\}$, $\Delta = \{a, b, c, d, e, f, g\}$, $\Gamma = \{r, s, t\}$, the axiom Z consists of the one-node graph labelled with S, P is shown in Fig. 12.
One can easily note that $L = L(G)$. *Q.E.D.*

Theorem 4. *There exists*

$$L \in \mathcal{L}(ETPL(1))$$

such that for any $k \geq 0$

$$L \notin \mathcal{L}(ETPR(k)).$$

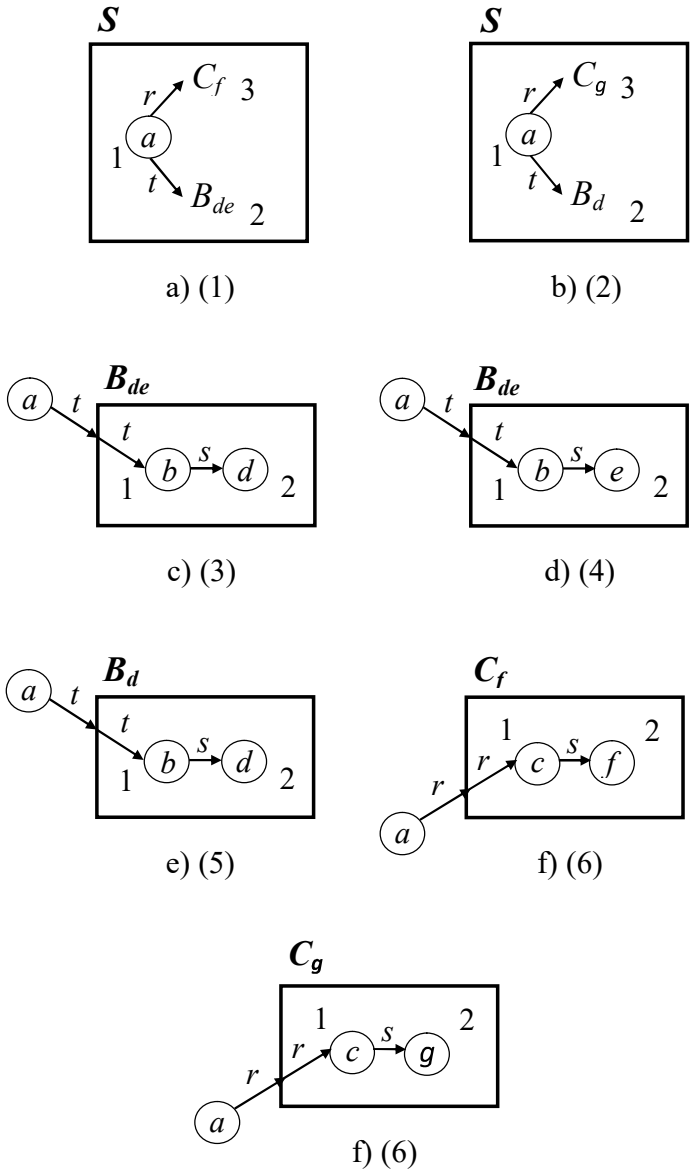

Fig. 12. The productions of the $ETPR(1)$ ***grammar*** G

Proof. Firstly, we define an *ETPL*(1) language L. Let $L = L_1 \cup L_2$. Each n-node graph, $n \geq 7$, $h_1 = (V_1, E_1, \Sigma_1, \Gamma_1, \phi_1) \in L_1$ is of the form shown in Fig. 13a. The graph consists of a node having the characteristic description $(a(1), 2, (tr), (23))$ and two paths. A sequence of nodes in each path is connected with edges labelled s. The lower path consists of:
- a subsequence of nodes indexed with: $2, \ldots, (2k-2), k \geq 2$ and labelled b,
- a (distinguished) node indexed with $2k, k \geq 2$ and labelled d, and
- a subsequence of nodes indexed with: $(2k+2), \ldots, k \geq 2$ and labelled f.

The upper path consists of:
- a subsequence of nodes indexed with: $2, \ldots, (2k-1), k \geq 2$ and labelled c,
- a (distinguished) node indexed with $(2k+1), k \geq 2$ and labelled e, and
- a subsequence of nodes indexed with: $(2k+3), \ldots, k \geq 2$ and labelled g.

The lengths of both paths (defined as a number of nodes in a sequence) can be various.

Additionally, there exists a $bridge = ((2k+1), u, (2k+2)) \in E_1$. In other words, let $dist(a, d)$ denote the number of nodes between the node labelled a and the node labelled d, and $dist(a, e)$ denotes a number of nodes between the node labelled a and the node labelled e. Then, $dist(a, d) = dist(a, e)$ and there exists a $bridge \in E_1$.

L_2 consists of graphs analogous to the graphs of L_1. However, $dist(a, d) \neq dist(a, e)$ and, there is no edge ($bridge$) connecting both paths. Summing up, an edge called a $bridge$ occurs in a graph $h \in L$ iff $dist(a, d) = dist(a, e)$. We will call L the *contextually-conditioned-bridge graph language*.

Let us the define an *ETPL*(1) grammar $G = (\Sigma, \Delta, \Gamma, P, Z)$ generating the language L.
$\Sigma = \{a, b, c, d, e, f, g, B, C, D, E, F, G\}$, $\Delta = \{a, b, c, d, e, f, g\}$, $\Gamma = \{p, r, s, t, u, x, y\}$, P and Z are shown in Fig. 14.

An example of generating a $bridge$ in case $dist(a, d) = dist(a, e)$ is shown in Fig. 13b. One can

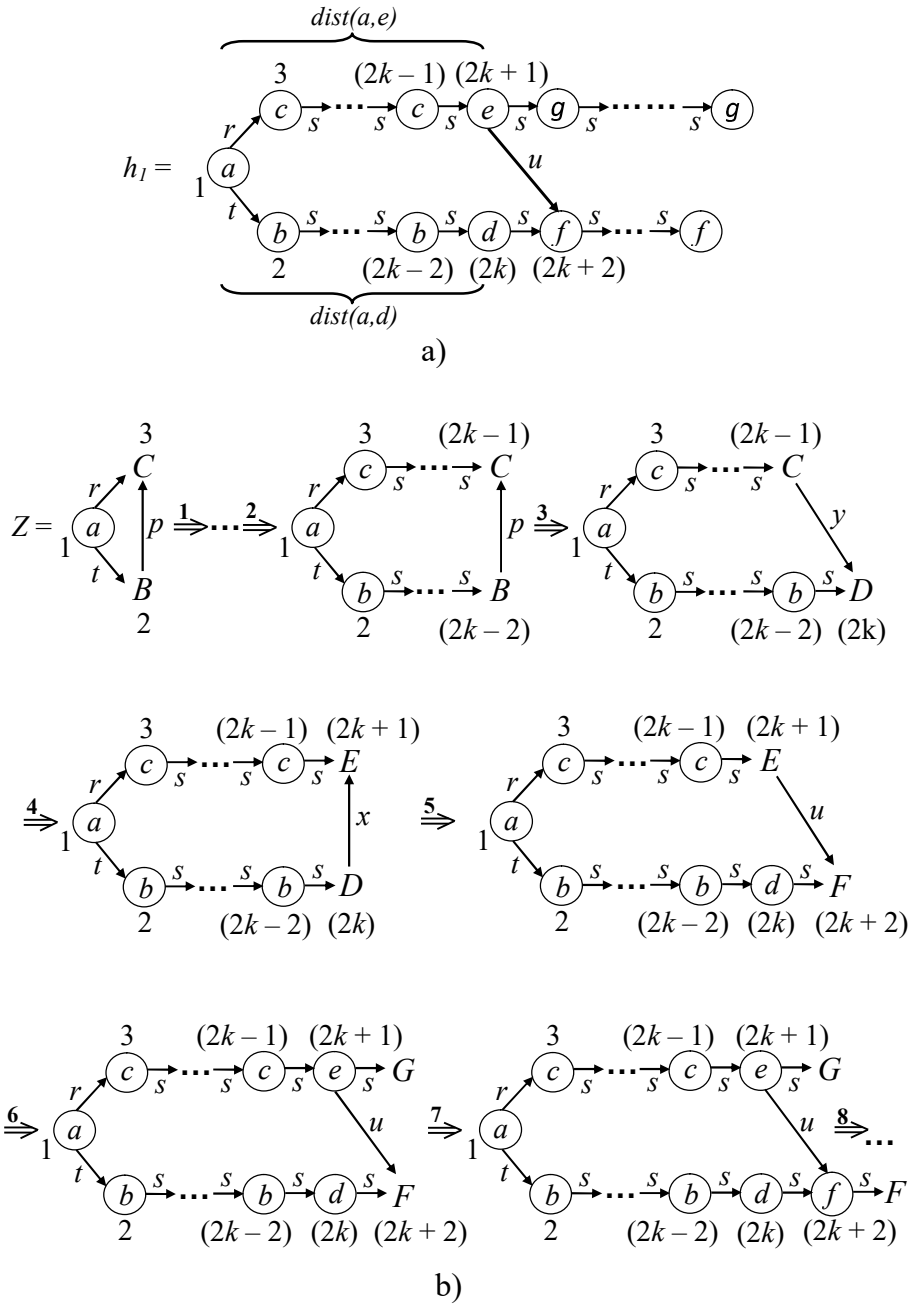

Fig. 13. The $ETPL(1)$ ***language that cannot be generated by any*** $ETPR(k)$ ***grammar***

easily see that $L = L(G)$.

Now, we show that L cannot be generated by any *ETPR*(k) grammar. First of all, let us note that any *rIE* graph $h_1 \in L_1$ has to be indexed as shown in Fig. 15a. If h_1 has to belong to L_1, then $dist(a, d) = dist(a, e)$. Assuming indexing defined as in Fig. 15a (i.e. a node labelled e is indexed with k), it means that a node labelled d has to be indexed with $(i + k - 3)$.

According to the Definition 10 of *ETPR*(k) grammars the upper path of h_1 has to be generated firstly, as we can see in Fig. 15b. (An edge $(2, r, i)$ is to be established in order to generate a $bridge$.) However, one can easily see that we do not know whether an index j of a node labelled d fulfils the condition: $j = (i + k - 3)$[17]. In consequence, we do not know whether to establish a $bridge$ (in case $h_1 \in L_1$) or not (in case $h_1 \in L_2$). *Q.E.D.*

5. CF and NLC Languages with Polynomial Membership Problem

As stated in the introduction, research into the theory of parsing for *NLC* graph grammars has been conducted for thirty years. The graph grammars of the *edNLC* class [30] were chosen as the basis for this research from the outset, which has proved to be appropriate. On the one hand, the *edNLC* class has been revealed as descriptively strong enough to be successfully applied for solving the previously mentioned real-world problems [2, 13, 17, 18, 21, 22, 24–27][18]. On the other hand, the *edNLC* class has turned out to be flexible enough to enable us to define the deterministic subclasses with polynomial membership problem, and in consequence - the efficient parsing algorithms. Moreover, the way of defining *edNLC* graph grammars has enabled to define these deterministic subclasses,

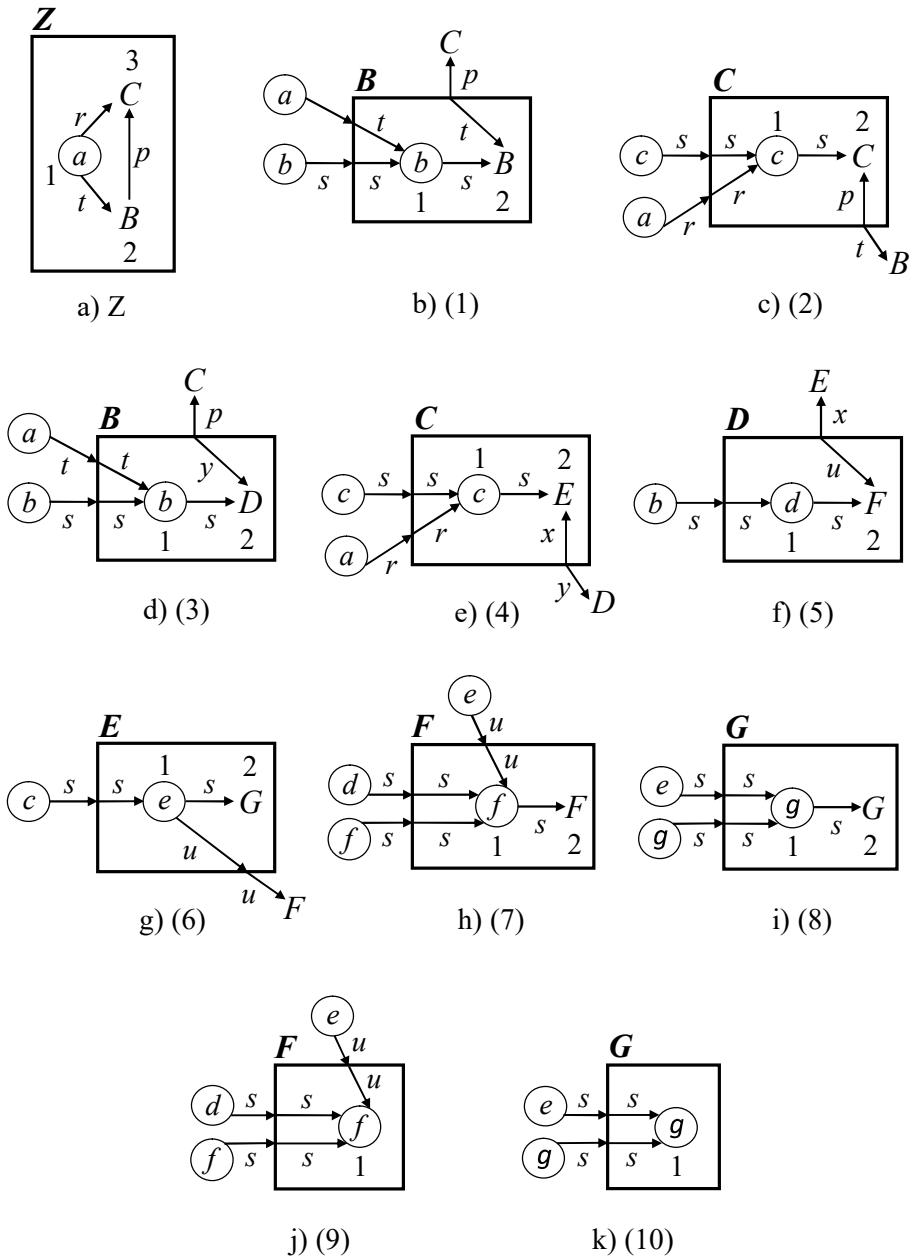

Fig. 14. The $ETPL(1)$ ***grammar generating the contextually-conditioned-bridge graph language.***

using constructs and mechanisms analogous to those used in the theory of parsing of string languages. This analogy is especially noteworthy, since from the methodological/paradigmatic point of view, analogies of this kind are highly desirable [21].

First of all, let us note that the essential properties of both subclasses of *edNLC* graph grammars, namely top-down parsable *ETPL*(k) and reduction-based (bottom-up) parsable *ETPR*(k), expressed by Definitions 13 and 15 are analogous to the definitions of theirs counterparts, namely subclasses of top-down parsable *LL*(k) [36, 44] and bottom-up parsable *LR*(k) [34] *CF* (*context-free*) grammars in the parsing theory of string languages.

Secondly, these analogies have proved to be useful when studying the formal properties of *ETPR*(k) languages presented in a previous section. For example, the well-known fact that the (string) *CF* language of palindromes cannot be generated by any *LR*(k) grammar has inspired us to construct the *edNLC complementary palindromic graph language* in order to show that it cannot be generated by any *ETPR*(k) grammar (the proof of Theorem 1). On the other hand, investigating whether *ETPR*(k) languages constitute a hierarchy, we have analyzed the Mickunas-Lancaster-Schneider stratification-based method used for proving that *LR*(k) languages do not constitute a hierarchy [38]. The study has revealed that the *stratification trick* [38] cannot be made in case of graph structures. Knowing why this is impossible, we have been able to define the *ETPR (m+1)-height-step cascade graph language* in order to show that *ETPR*(k) languages constitute a hierarchy (the proof of Theorem 2).

In our previous paper concerning the generative power of *ETPL*(k) languages [19] we have proved, among others, the following two theorems.

Theorem (1. in [19]) *For a given* $k \geq 0$

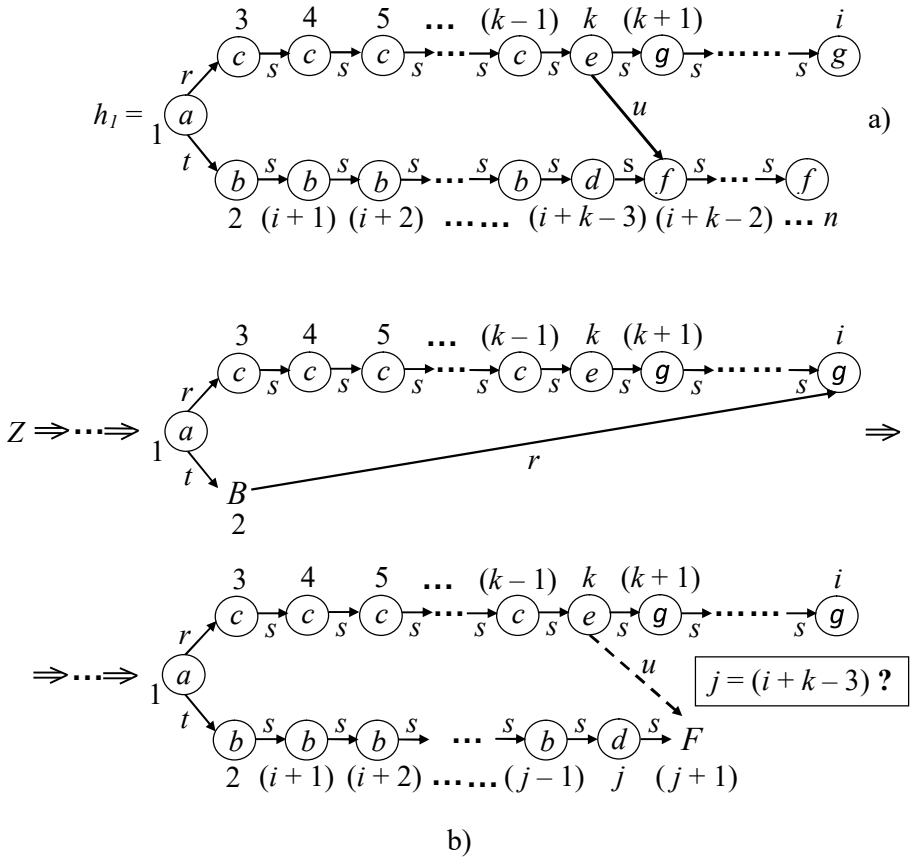

Fig. 15. The hypothetical derivation of the graph $h_1 \in L_1$ ***with an*** $ETPR(k)$ ***grammar.***

$$\mathcal{L}(ETPL(k)) \subsetneq \mathcal{L}(ETPL(k+1)) \ .$$

Theorem (2. in [19]) *For any* $k \geq 0$

$$\mathcal{L}(ETPL(k)) \subsetneq \mathcal{L}(edNLC_o) \ .$$

These two theorems together with the ones proved in a previous section allow us to establish a diagram presenting the relationships among the families of parsable *edNLC* languages shown in Fig. 16b. An analogous diagram for (string) *CF* languages is shown in Fig. 16a. The analogy between both basic classes of languages, i.e. (string) *CF* and (graph) *edNLC*, can be easily noted. However, there are also some essential differences. The first one consists of the lack of a hierarchy in the case of a bottom-up parsable subclass of the *edNLC* class. The second difference is crucial from an application point of view. Whereas the family of *LL*-type languages is strictly contained in the family of *LR*-type languages, the classes *ETPL* and *ETPR* are not comparable. Although the insufficient descriptive power of *ETPL*-type languages for solving certain real-world application problems was the original motivation of the author for conducting research into a bottom-up parsable subclass of *edNLC* grammars, finally it was shown that both parsable subclasses are needed and that they complement each other.

6. Concluding Remarks

The following two goals were the focus of our research into *Node Label Controlled* (*NLC*) graph grammars formulated in [30].

- To establish a theory of parsing for *NLC* graph languages (the theoretical-oriented research area).
- To apply this theory to various real-world problems, which require efficient algorithmic schemes of graph (sets of graphs) processing (the application-oriented research area), for their solution.

Two generic types of parsable subclasses of languages with polynomial membership problem are considered in the theory of parsing: the top-down parsable languages and the reduction-based (bottom-up) parsable ones. These two generic subclasses have both their pros and cons. Therefore, within the theoretical-oriented area of our research two subclasses of *NLC* graph grammars have been developed, namely top-down parsable $ETPL(k)$ (analogous to $LL(k)$ grammars [36, 44]) and bottom-up parsable $ETPR(k)$ (analogous to $LR(k)$ grammars [34]). The generative power of the former has been presented in [19] and the latter - in this paper. Additionally, we have compared generative power of both subclasses as well. Finally, we have discussed the analogy between the triad of *CF* - $LL(k)$ - $LR(k)$ string languages and the triad of *NLC* - $ETPL(k)$ - $ETPR(k)$ graph languages.

Apart from the previously discussed theoretical results, both parsable subclasses of *NLC* graph grammars have been successfully used in a variety of applications such as: scene analysis in robotics [13], software allocation in distributed systems [25], CAD/CAM integration [18, 22], reasoning in real-time expert systems [2, 17], mesh refinement (finite element met-

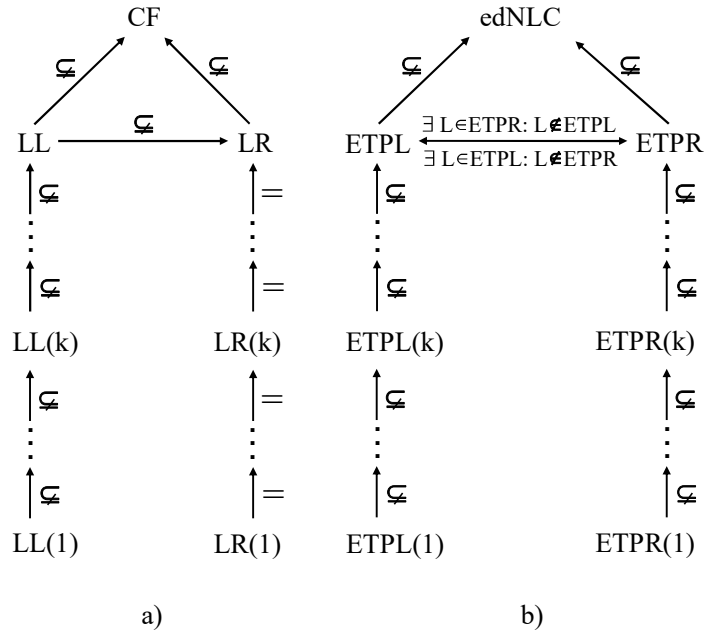

Fig. 16. The analogy between** CF **and** edNLC**: a) relationships among families of parsable** CF **languages, b) relationships among families of parsable** edNLC **languages

hod, FEM) in CAE systems [27], sign language recognition [21, 26] and computer vision [24] within the application-oriented area of our research .

Summing up, the class of *NLC* graph grammars has proved to be an attractive theoretical model because of its well-balanced properties. That is, on one hand, due to its simplicity, formal elegance and strong descriptive power, and on the other hand because of its flexibility allowing it to be used in a variety of real-world applications. In our opinion, *NLC* graph grammars provide an attractive reference model for the theory of parsing of graph languages and that they will play a key role in the further development of this theory.

Notes

[1] *NLC* grammars are introduced below.

[2] This condition concerning *(r)IE* graphs can be fulfilled easily in practice. *(r)IE* graphs have been used as a descriptive formalism for representing: combinations of objects of scenes analyzed by industrial robots [13], configurations of hardware/software components analyzed by distributed software allocators [25], structures consisting of geometrical/topological features of machine parts in CAD/CAM integration systems [18, 21], semantic networks/frames in real-time expert systems [2, 17], grids analyzed with Finite Element Analysis (FEA) methods in Computer Aided Engineering (CAE) systems [27], hand postures analyzed by sign language recognition systems [21, 26].

[3] That is, (some) edges of G can be re-directed with respect to their counterparts in H.

[4] Let us recall that LOTT means that for each node firstly the node is visited, then its child nodes are put into the FIFO queue. This type of a tree traversal is also known as the Breadth First Search (BFS) scheme.

[5] We assume that the root is on level 1, its children are on level 2, etc.

[6] Formal properties of the $ETPL(k)$ class have been presented in [19].

[7] Analogously, as for parsable $LL(k)/LR(k)$ string grammars a leftmost/rightmost derivation is required.

[8] The formalization of these conditions is contained in the paper on inferencing $ETPL(k)$ graph grammars [20].

[9] Both notions: $k-TL$ graph and $FIRST_k$ prefix play an analogous role in considered models.

[10] A *regular left-hand side derivation* in our model is analogous to a *leftmost derivation* for *CF* grammars.

[11] A *regular right-hand side derivation* in our model is analogous to a *rightmost derivation* for *CF* grammars.

[12] That is, we can choose a proper left-hand side.

[13] That is, productions 1, 2, 3, 4 in G.

[14] In [19] (Lemma 1, p. 207) we have proved that the index of a replaced node is always preserved in our model.

[15] As it is in productions 1, 2, 3, 4.

[16] To determine a proper reduction (unambiguously) one has to analyze $(m+1)-TL$ graphs instead.

[17] Obviously, at any derivation step we do not know how many nodes labelled with b have been generated till this step.

[18] Let us note that although syntactic pattern recognition problems have been the main motivation for the application-oriented part of this research, it was not limited to this area and has included e.g. distributed systems, reasoning over ontologies in expert systems.

AUTHOR

Mariusz Flasiński* – Information Technology Systems Department, Jagiellonian University, ul. prof. St. Łojasiewicza 4, Cracow 30-348, Poland, e-mail: Mariusz.Flasinski@uj.edu.pl, www: www.ksi.uj.edu.pl/~mfl.

*Corresponding author

REFERENCES

[1] M. Bauderon and B. Courcelle, "Graph expressions and graph rewritings", *Mathematical Systems Theory*, vol. 20, 1987, 83–127.

[2] U. Behrens, M. Flasiński, L. Hagge, and K. Ohrenberg, "ZEX - An expert system for ZEUS", *IEEE Trans. Nuclear Science*, vol. 41, 1994, 152–156.

[3] M. Blum and C. Hewitt, "Automata on a two-dimensional tape". In: *Proc. 8th IEEE Annual Symposium Switching and Automata Theory*, Austin, TX, USA, 1967, 155–160.

[4] F. J. Brandenburg, "On the complexity of the membership problem of graph grammars". In: *Proc. Int. Workshop on Graphtheoretic Concepts in Computer Sci. (WG'83)*, Osnabrück, Germany, 1983, 40–49.

[5] F. J. Brandenburg and K. Skodinis, "Finite graph automata for linear and boundary graph languages", *Theoretical Computer Science*, vol. 332, 2005, 199–232.

[6] J. M. Brayer, "Parsing of web grammars". In: *Proc. IEEE Workshop Picture Data Description and Management*, Chicago, IL, USA, 1977.

[7] H. O. Bunke and B. Haller, "A parser for context free plex grammars", *Lecture Notes in Computer Science*, vol. 411, 1990, 136–150.

[8] P. Della Vigna and C. Ghezzi, "Context-free graph grammars", *Information and Control*, vol. 37, 1978, 207–233.

[9] F. Drewes, B. Hoffmann, and M. Minas, "Predictive top-down parsing for hyperedge replacement grammars", *Lecture Notes in Computer Science*, vol. 9151, 2015, 19–34.

[10] H. Ehrig, , H. J. Kreowski, U. Montanari, and G. Rozenberg, eds., *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 3: Concurrency, Parallelism, and Distribution*, World Scientific: Singapore, 1999.

[11] H. Ehrig, G. Engels, H. J. Kreowski, and G. Rozenberg, eds., *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and Tools*, World Scientific: Singapore, 1999.

[12] J. Feder, "Plex languages", *Information Sciences*, vol. 3, 1971, 225–241.

[13] M. Flasiński, "Parsing of edNLC-graph grammars for scene analysis", *Pattern Recognition*, vol. 21, 1988, 623–629.

[14] M. Flasiński, "Characteristics of edNLC-graph grammars for syntactic pattern recognition", *Computer Vision, Graphics and Image Processing*, vol. 47, 1989, 1–21.

[15] M. Flasiński, "Distorted pattern analysis with the help of Node Label Controlled graph languages", *Pattern Recognition*, vol. 23, 1990, 765–774.

[16] M. Flasiński, "On the parsing of deterministic graph languages for syntactic pattern recognition", *Pattern Recognition*, vol. 26, 1993, 1–16.

[17] M. Flasiński. "Further development of the ZEUS Expert System: Computer science foundations of design". Technical Report 94-048, Deutsches Eletronen-Synchrotron, Hamburg, Germany, 1994.

[18] M. Flasiński, "Use of graph grammars for the description of mechanical parts", *Computer-Aided Design*, vol. 27, 1995, 403–433.

[19] M. Flasiński, "Power properties of NLC graph grammars with a polynomial membership problem", *Theoretical Computer Science*, vol. 201, 1998, 189–231.

[20] M. Flasiński, "Inference of parsable graph grammars for syntactic pattern recognition", *Fundamenta Informaticae*, vol. 80, 2007, 379–413.

[21] M. Flasiński, *Introduction to Artificial Intelligence*, Springer International: Switzerland, 2016.

[22] M. Flasiński. "Syntactic pattern recognition: paradigm issues and open problems". In: C. H. Chen, ed., *Handbook of Pattern Recognition and Computer Vision*, 3–25. World Scientific, New Jersey-London-Singapore, 2016.

[23] M. Flasiński, "Interpreted graphs and ETPR(k) graph grammar parsing for syntactic pattern recognition", *Machine Graphics and Vision*, vol. 27, 2018, 3–19.

[24] M. Flasiński, *Syntactic Pattern Recognition*, World Scientific: New Jersey - London - Singapore, 2019.

[25] M. Flasiński and L. Kotulski, "On the use of graph grammars for the control of a distributed software allocation", *The Computer Journal*, vol. 35, 1992, A165–A175.

[26] M. Flasiński and S. Myśliński, "On the use of graph parsing for recognition of isolated hand postures of Polish Sign Language", *Pattern Recognition*, vol. 43, 2010, 2249–2264.

[27] M. Flasiński, R. Schaefer, and W. Toporkiewicz, "Supporting CAE parallel computations with IE-graph solid representation", *J. Geometry Graphics*, vol. 1, 1997, 23–29.

[28] R. Franck, "A class of linearly parsable graph grammars", *Acta Informatica*, vol. 10, 1978, 175–201.

[29] A. Habel and H. J. Kreowski, "May we introduce to you: hyperedge replacement", *Lecture Notes in Computer Science*, vol. 291, 1987, 15–26.

[30] D. Janssens and G. Rozenberg, "On the structure of node-label-controlled graph languages", *Information Sciences*, vol. 20, 1980, 191–216.

[31] D. Janssens and G. Rozenberg, "Graph grammars with neighbourhood-controlled embedding", *Theoretical Computer Science*, vol. 21, 1982, 55–74.

[32] D. Janssens, G. Rozenberg, and R. Verraedt, "On sequential and parallel node-rewriting graph grammars", *Computer Graphics and Image Processing*, vol. 18, 1982, 279–304.

[33] M. Kaul, "Parsing of graphs in linear time", *Lecture Notes in Computer Science*, vol. 153, 1983, 206–218.

[34] D. Knuth, "On the translation of languages from left to right", *Information and Control*, vol. 8, 1965, 607–639.

[35] C. Lautemann, "Efficient algorithms on context-free graph languages", *Lecture Notes in Computer Science*, vol. 317, 1988, 362–378.

[36] P. M. Lewis II and R. E. Stearns, "Syntax-ditected transduction", *Journal of the Association for Computing Machinery*, vol. 15, 1968, 465–488.

[37] S. Mazanek and M. Minas, "Functional-logic graph parser combinators", *Lecture Notes in Computer Science*, vol. 5117, 2008, 261–275.

[38] M. D. Mickunas, R. L. Lancaster, and V. B. Schneider, "Transforming LR(k) grammars to LR(1), SLR(1), and (1,1) bounded right-context grammars", *Journal of the Association for Computing Machinery*, vol. 23, 1976, 511–533.

[39] K. J. Peng, T. Yamamoto, and Y. Aoki, "A new parsing scheme for plex grammars", *Pattern Recognition*, vol. 23, 1990, 393–402.

[40] J. L. Pfaltz and A. Rosenfeld, "Web grammars". In: *Proceedings First Int. Conf. Artificial Intelligence*, 1969, 609–619.

[41] T. W. Pratt, "Pair grammars, graph languages and string-to-graph translations", *Journal of Computer and System Sciences*, vol. 5, 1971, 560–595.

[42] J. Rekers and A. Schürr, "Defining and parsing visual languages with layered graph grammars", *Journal of Visual Languages and Computing*, vol. 8, 1997, 27–55.

[43] A. Rosenfeld and D. L. Milgram. "Web automata and web grammars". In: B. Meltzer and E. Michie, eds., *Machine Intelligence 7*. Edinburgh University Press, Edinburgh, 1972.

[44] D. J. Rosenkrantz and R. E. Stearns, "Properties of deterministic top-down grammars", *Information and Control*, vol. 17, 1970, 226–256.

[45] G. Rozenberg, ed., *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations*, World Scientific: Singapore, 1997.

[46] G. Rozenberg and E. Welzl, "Boundary NLC graph grammars - basic definitions, normal forms, and complexity", *Information and Control*, vol. 69, 1986, 136–167.

[47] S. Seifert and I. Fischer, "Parsing string generating hypergraph grammars", *Lecture Notes in Computer Science*, vol. 3256, 2004, 352–367.

[48] Q. Y. Shi and K. S. Fu, "Parsing and translation of attributed expansive graph languages for scene analysis", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 5, 1983, 472–485.

[49] A. O. Slisenko, "Context-free grammars as a tool for describing polynomial-time subclasses of hard problems", *Information Processing Letters*, vol. 14, 1982, 52–56.

[50] M. Tucci, G. Vitiello, and G. Costagliola, "Parsing nonlinear languages", *IEEE Trans. Software Engineering*, vol. 20, 1994, 720–739.

[51] G. Turan. "On the complexity of graph grammars". Rep. automata theory research group, Szeged, Hungary, 1982.

[52] W. Vogler, "Recognizing edge replacement graph languages in cubic time", *Lecture Notes in Computer Science*, vol. 532, 1991, 676–687.

[53] L. M. Wills, "Using attributed graph parsing to recognize clichés in programs", *Lecture Notes in Computer Science*, vol. 1073, 1996, 170–184.

[54] K. Wittenburg, L. Weitzman, and J. Talley, "Unification-based grammars and tabular parsing for graphical languages", *Journal of Visual Languages and Computing*, vol. 2, 1991, 347–370.

[55] D. Q. Zhang, K. Zhang, and J. Cao, "A context-sensitive graph grammar formalism for the specification of visual languages", *The Computer Journal*, vol. 44, 2001, 186–200.