Interpreted Graphs and *ETPR*(k) Graph Grammar Parsing for Syntactic Pattern Recognition

Mariusz Flasiński
IT Systems Department, Jagiellonian University
ul. St. Łojasiewicza 4, 30-384 Cracow, Poland

Abstract. Further results of research into graph grammar parsing for syntactic pattern recognition (*Pattern Recognit.* 21:623–629, 1988; 23:765–774, 1990; 24:1223–1224, 1991; 26:1–16, 1993; 43:249–2264, 2010; *Comput. Vision Graph. Image Process.* 47:1–21, 1989; *Fundam. Inform.* 80:379–413, 2007; *Theoret. Comp. Sci.* 201:189–231, 1998) are presented in the paper. The notion of interpreted graphs based on Tarski's model theory is introduced. The bottom-up parsing algorithm for *ETPR*(k) graph grammars is defined.

Key words: syntactic pattern recognition, graph grammar, parsing, interpreted graph, model theory

1. Introduction

Syntactic pattern recognition consists in representing patterns with string, tree or graph structures, defining generative grammars for sets of such structures, and using corresponding automata/syntax analyzers for classifying unknown structural patterns [1, 2, 3, 4]. Graph grammars are the strongest formalism generating structural patterns and they are used in machine graphics and vision for this purpose for more than 40 years [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19].

Although the extensive research into parsing of graph languages for syntactic pattern recognition has been carried out, only a few syntax analyzers for graph-based patterns have been developed. The efficient parser for expansive graph grammars was defined by Fu and Shi in 1983 [20]. In 1990 parsing algorithms for plex grammars were constructed by Bunke and Haller [21], and Peng, Yamamoto and Aoki [22]. Then, syntax analyzers for relational grammars were proposed by: Wittenburg, Weitzman and Talley in 1991 [23], and Ferruci, Tortora, Tucci and Vitiello in 1994 [24]. The parsing algorithm for context-sensitive layered grammars was presented by Rekers and Schürr in [25]. The parsing method for reserved graph grammars was defined by Zhang, Zhang and Cao in 2001 [26].

Till the first half of 1990s three efficient parsing algorithms, $O(n^2)$, were defined for subclasses of classical *Node Label Controlled* (*NLC*) graph grammars introduced in [27]. Firstly, the syntax analyzer for the regular *ETL*(1) subclass of *edNLC* languages was proposed in [28, 29], then its error-correcting extension was defined [30, 31], and finally the parser for the context-free *ETPL*(k) subclass of *edNLC* languages was constructed [32]. Moreover, formal power properties of *ETPL*(k) graph languages were characterized in [33] and the inference algorithm for these languages was defined [34]. The *ETPL*(k) parser was successfully used in a variety of practical applications like: scene analysis in robotics [28], software allocation in distributed systems [35], CAD/CAM integration [36, 37], reasoning in real-time expert system [38], mesh refinement (finite element method, FEM) in CAE system [39], reasoning with semantic networks in AI systems [40], sign language recognition [41, 42].

The successful use of the *ETPL*(k) model in aforementioned applications results, among others, from the linear ordering of graphs representing visual objects, components of networks, etc. In turn, the effective definition of this ordering is the result of constructing the graphs on the basis of semantic features of represented phenomena. Formulating general preconditions allowing one to construct such graphs, called here *interpreted graphs* is the first goal of the paper. The complete formalization of the bottom-up parsable version of the deterministic subclass of *edNLC* graph grammars analogous to *ETPL*(k) grammars[1] and presenting the *ETPR*(k) parsing algorithm analogous to the one introduced in [32] is the second goal of the paper.

Definitions relating to *edNLC* graph grammars are presented in Section 2. Notions of interpreted graphs and (reversely) indexed edge-unambiguous graphs that allow us to introduce linear ordering on graphs used for representing patterns are included in Section 3. Definitions of bottom-up parsable *ETPR*(k) graph grammars are contained in Section 4, whereas in Section 5 the *ETPR*(k) parsing algorithm is presented. The concluding remarks are included in the last section.

2. Preliminaries

In this section we present basic definitions of: *EDG* graph, *edNLC* graph grammar and *edNLC* graph language [27].

Definition 2.1. A *directed node- and edge-labelled graph*, *EDG* graph, over Σ and Γ is a quintuple

$$H = (V, E, \Sigma, \Gamma, \phi)\ \text{, where}$$

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), where $v, w \in V, \lambda \in \Gamma$,
$\phi : V \longrightarrow \Sigma$ is a node-labelling function.

The *family of all 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.

[1]The preliminary formalization was presented in [37].

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 $A \overset{\text{isom}}{=} B$.
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 2.2. An *edge-labelled directed Node Label Controlled*, *edNLC*, *graph grammar* is a quintuple

$$G = (\Sigma, \Delta, \Gamma, P, Z), \text{where}$$

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
$l \in \Sigma \backslash \Delta, D \in EDG_{\Sigma,\Gamma}, C : \Gamma \times \{\text{in}, \text{out}\} \longrightarrow 2^{\Sigma \times \Sigma \times \Gamma \times \{\text{in},\text{out}\}}$ is the embedding transformation,
$Z \in EDG_{\Sigma,\Gamma}$ is the starting graph called the *axiom*.

Definition 2.3. 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 $v \in V_H$ 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

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

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

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 is shown in Fig. 1. The graph h which will be transformed is shown in Fig. 1a. 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 in the following way.

$$\begin{aligned} &\textbf{(i)} \quad C(p,\text{in}) \;\; = \{(D,B,v,\text{out})\} \,, \\ &\textbf{(ii)} \quad C(v,\text{out}) = \{(b,a,v,\text{out})\} \,. \end{aligned}$$

The derivation step is performed in two stages. Firstly, the node labelled with A of the graph h is removed and the graph of the right-hand side D is placed instead of this node. The transformed graph after removing the node is called the rest graph. During the second stage the embedding transformation is used in order to connect certain nodes of the graph D with the rest graph. The item 2 is interpreted in the following way.

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

So, the item 2 generates the edge of the graph $\overline{h}$, shown in Fig. 1c, which is labelled with v and connects nodes labelled with D and B on the basis of the edge of the graph h labelled with p and connecting nodes labelled with A and B. Let us notice that the application of the item 2 preserves the edge labelled with v of the graph h.

3. Interpreted graphs and indexed edge-unambiguous graphs

As it has been discussed in [33], there are two main reasons of difficulties with constructing efficient parsing algorithms for graph languages (comparing with such algorithms for string and tree languages): the lack of ordering of graph structure and the complexity of embedding transformation. In this section we consider the first problem.

Let is notice that the main idea of any syntax analysis algorithm consists in repetitive tearing off succeeding subphrases/substructures (*handles*) from an analyzed sentence/structure and matching them against phrases/structures which are defined on the basis of right-hand sides of productions. In case of a graph structure it means looking for a subgraph (a handle) which is isomorphic to a given graph, i.e. resolving the subgraph isomorphism problem known to be NP-complete. To resolve this problem we have introduced the subclass of *EDG* graphs called *indexed edge-unambiguous graphs, IE graphs* [28, 33] in which the linear order has been defined. In spite of the fact that

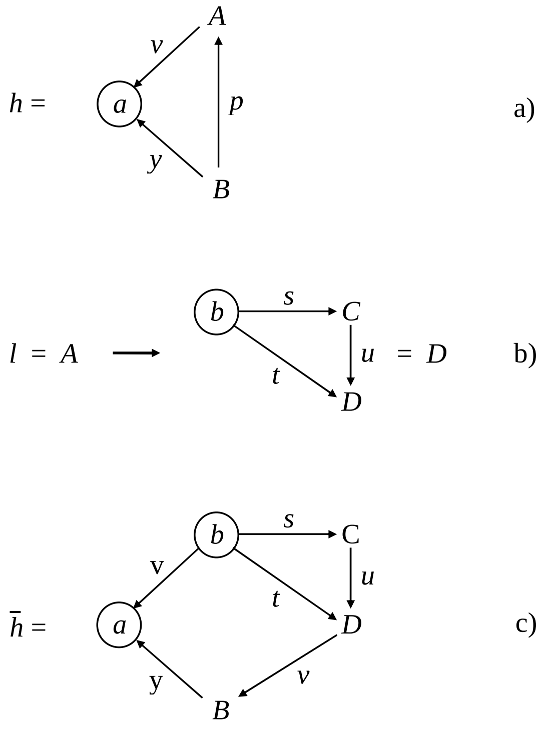

Fig. 1. An example of a derivation step in an *edNLC* graph grammar.

formal restrictions have been imposed on the class of *IE* graphs, it has turned out that they do not limit its descriptive power in practice. The *IE* graphs have been used as a descriptive formalism for representing: combinations of objects of scenes analyzed by industrial robots [28], configurations of hardware/software components analyzed by distributed software allocators [35], structures consisting of geometrical/topological features of machined parts in CAD/CAM integration systems [36, 37], frames in real-time expert systems [38], grids analyzed with FEA methods in CAE systems [39], semantic networks in AI systems [40] hand postures analyzed by sign language recognition systems [41, 42].

Although specific preconditions which have to be fulfilled for the effective use of *IE* graphs have been determined and discussed for each of these applications, they have not been defined formally in a general case till now. Only in [28, 33] it has been pointed out intuitively that one has to refer to a *semantic aspect* of a graph representation. Indeed, in order to formulate general conditions for the effective use of the class of *IE* graphs we have to refer to *Tarski's (semantic) model theory.* We will use the model theory approach to define the class of *interpreted EDG graphs.*

Graphs are used in computer science for representing *structures* which consist of objects and relations among them. These objects, corresponding to *individual objects* in logic, can represent physical entities/phenomena (e.g. Albert Einstein, Hurricane Katrina), sets (groups) of entities (e.g. my family), social/cultural/political constructs (e.g. UNESCO, USA), (theoretical) concepts (e.g. the set of natural numbers, triangle, animal). Individual objects are represented with graph nodes, whereas relations among these objects are represented with graph edges. Hereinafter structures represented with graphs will be called *relational structures* (in order to distinguish them from (abstract) structures defined in model theory)[2] Then we will assume that a graph node is characterized with *node attributes* and a *node label* (which in fact is a kind of the distinguished attribute). Usually for a graph node representing a *unique object* (e.g. (this) Albert Einstein) the node label corresponds to the unique "identifier" of the object, and for a graph node representing a *concept object (class, category)* (e.g. tree) the node label corresponds to the name of the concept. A graph edge between nodes v and w is characterized with an *edge label* which defines the kind of the relation between objects (of a relational structure) which are represented with the nodes v and w. Since we construct our formalism for *EDG* graphs we assume that relations are binary and there are no multiple relations between objects (cf. [27]). For graph edges, similarly as for nodes, attributes can be also defined.

Let us formalize our considerations. Firstly, we introduce the definition of *relational structure.*

Definition 3.1. Let $\mathcal{U}$ be a finite set of individual objects called *universe*, $\mathcal{N}_{\mathcal{U}}$ be a set of their names, $\mathcal{A}_{\mathcal{U}}$ be a set of their attributes.

[2]That is, in our terminology *graphs* (treated as a representative formalism) represent *relational structures* that are constituted by objects and relations among them.

Let each object $o^k, k = 1, \dots K$ of $\mathcal{U}$ be represented by its name $n_u^k \in \mathcal{N}_\mathcal{U}$ and the set of its attributes[3] $a_u^k \in 2^{\mathcal{A}_\mathcal{U}}$.
Let $\mathcal{R} \subset 2^{\mathcal{U}\times\mathcal{U}}$ be a set of binary relations such that for a pair of objects at most one relation is established, $\mathcal{N}_\mathcal{R}$ be a set of their names, $\mathcal{A}_\mathcal{R}$ be a set of their attributes, $\mathcal{R} = \{(n_r, a_r) \mid n_r \in \mathcal{N}_\mathcal{R}, a_r \in 2^{\mathcal{A}_\mathcal{R}}\}$.
A *relational structure* is a sextuple $\mathfrak{S} = (\mathcal{U}, \mathcal{R}, \mathcal{N}_\mathcal{U}, \mathcal{N}_\mathcal{R}, \mathcal{A}_\mathcal{U}, \mathcal{A}_\mathcal{R})$.

Now we can define the *interpretation of EDG graph over relational structure* and the *interpreted EDG graph*.

Definition 3.2. Let $H = (V, E, \Sigma, \Gamma, \phi)$ be an *EDG* graph over Σ and Γ, $\mathfrak{S}$ be a relational structure defined as in Definition 3.1, $\Sigma \subset \mathcal{N}_\mathcal{U}, \Gamma \subset \mathcal{N}_\mathcal{R}$.
An *interpretation* $\mathcal{I}$ of the graph H over the structure $\mathfrak{S}$ is a pair

$$\mathcal{I} = (\mathfrak{S}, \mathcal{F})\ ,\ \text{where}$$

$\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2)$ is the *denotation function* defined in the following way.

- $\mathcal{F}_1$ assigns an object $u \in \mathcal{U}$ having a name $a \in \mathcal{N}_\mathcal{U}$ to each graph node $v \in V, \phi(v) = a, a \ \in \Sigma$,
- $\mathcal{F}_2$ assigns a pair of objects $(u', u'') \in r, r \in \mathcal{R}$ to each graph edge $(v, \lambda, w) \in E,\ v, w \in V, \lambda \in \Gamma$ such that $\mathcal{F}_1(v) = u'$, $\mathcal{F}_1(w) = u''$ and r has the name λ.

Definition 3.3. Let H be an *EDG* graph over Σ and Γ, $\mathfrak{S}$ be a relational structure, $\mathcal{I}$ be the interpretation of H over $\mathfrak{S}$ defined as in Definition 3.2. An *interpreted EDG graph* is a triple $H^\mathcal{I} = (\mathfrak{S}, H, \mathcal{I})$.

The *family of all the EDG graphs over* Σ *and* Γ *interpreted by* $\mathcal{I}$, shortly *interpreted EDG graphs*, is denoted by $EDG^\mathcal{I}_{\Sigma,\Gamma}$.

The examples of defining interpreted graphs for representing machined parts in the vision subsystem of the CAD/CAM system [36] and hand gestures in the Polish Sign Languages recognition system [41] are shown in Figs. 2a and 2b, respectively.

As we have mentioned above, we have been able to define the linear order for *EDG* graphs in various application areas because, in fact, we have considered *interpreted EDG* graphs. In all mentioned applications of *edNLC* grammars, the linear order has been introduced on the basis of semantics (i.e. attributes) of graphs representing relational structures.

Before we introduce the family of indexed edge-unambiguous graphs (defined for top-down parsable *edNLC* languages) and the family of reversely indexed edge-unambiguous graphs (defined for bottom-up parsable *edNLC* languages), let us define the string-like graph representation of *EDG* graphs as in [28, 32]. (This kind of the representation was originally defined for Ω graphs in [20].)

Definition 3.4. Let $k \in V$ be the node having the index k of the *EDG* graph $H = (V, E, \Sigma, \Gamma, \phi)$. A *characteristic description* of the node k is the quadruple

[3]The set of attributes for an object can be the empty set.

$$n(k), r, (e_1 \dots e_r), (i_1 \dots i_r)\ , \text{ where}$$

n is the label of the node k, i.e. $\phi(k) = n$,
r is the out-degree of k (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 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.

If nodes of the graph h from Fig. 1a labelled with: a, B, A are indexed with: $1, 2, 3$, respectively, then:

$$B(2),\quad 2,\quad (y\ p),\quad (1\ 3)$$

is the characteristic description of the node indexed with 2.

Definition 3.5. Let $H = (V, E, \Sigma, \Gamma, \phi)$ be an *IE* graph, where $V = \{1, \dots, k\}$ is the set of its nodes, $I(i), i = 1, \dots, k$ is the characteristic description of the form of the node i. A string $I(1) \dots I(k)$ is called a *characteristic description* of the graph H.

Assuming a way of indexing of the graph h from Fig. 1a as it has been defined above, we receive the following characteristic description of this graph

$$\begin{array}{ccc} a(1) & B(2) & A(3) \\ 0 & 2 & 1 \\ - & y\ p & v \\ - & 1\ 3 & 1 \end{array} .$$

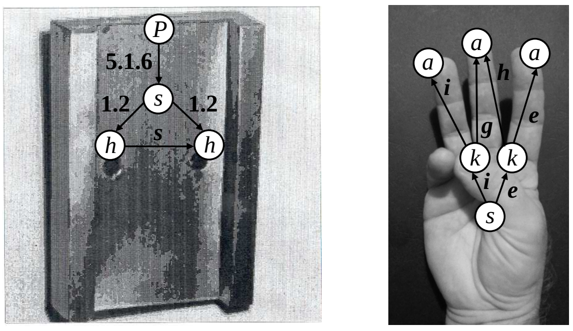

Fig. 2. The application of interpreted graphs for representing (a) machined parts in the vision subsystem of the CAD/CAM system [36] and (b) hand gestures in the Polish Sign Language recognition system [41].

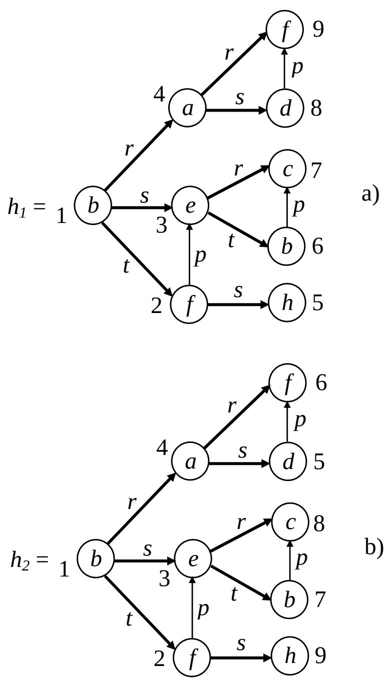

Fig. 3. An example of an *IE* graph (a) and an *rIE* graph (b).

On the basis of semantic features of *EDG* graphs we have constructed the so-called *IE* graphs used in the top-down *ETPL*(k) parsing scheme [28, 32]. Let us define them formally on the basis of the concept of *interpreted graphs* introduced above in Definition 3.3.

Definition 3.6. Let $H^{\mathcal{I}} = (\mathfrak{S}, H, \mathcal{I})$ be an interpreted *EDG* graph over Σ and Γ. An *indexed edge-unambiguous graph*, *IE* graph, over Σ and Γ defined on the basis of the graph $H^{\mathcal{I}}$ is an *EDG* graph $G = (V, E, \Sigma, \Gamma, \phi)$ which is isomorphic to H up to direction of edges[4], 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)[5].
2. Nodes of G are indexed in the same way as nodes of T.

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

[5]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.

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 $IE_{\Sigma,\Gamma}$.
The exemplary IE graph h_1 is shown in Fig. 3a. The indices are ascribed to the graph nodes according to LOTT. The edges of the spanning tree T are thickened.

Since in the paper we characterize formally bottom-up parsable *ETPR*(k) graph grammars, we will introduce the class of graphs which are generated by these grammars. These graphs should be indexed according to the scheme allowing us to apply a reductive parsing. During such a parsing a syntax analyzer produces the rightmost derivation in reverse. (As it is performed for Knuth's (string) *LR*(k) parsers [43].) The class of such graphs has been introduced informally in [37]. We define them on the basis of interpreted graphs using the new traversal scheme called the *Reverse Level Order Tree Traversal (RLOTT).* This scheme is analogous to the LOTT scheme used above for *IE* graphs, however it uses the LIFO queue, i.e. the stack, instead of the FIFO queue.

Definition 3.7. Let $H^{\mathcal{I}} = (\mathfrak{S}, H, \mathcal{I})$ be an interpreted *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 $H^{\mathcal{I}}$ is an *EDG* graph $G = (V, E, \Sigma, \Gamma, \phi)$ which is isomorphic to H up to direction of 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 $rIE_{\Sigma,\Gamma}$.

The exemplary IE graph h_2 is shown in Fig. 3b.

At the end of this section we introduce the notion of node level. We say that the node v of the *IE* (*rIE*) graph is of the n level, if v is at the n level of the spanned tree T constructed as in Definition 3.6 (Definition 3.7).

4. Formal properties of *ETPR*(k) graph grammars

The formal properties of the *ETPR*(k) bottom-up parsable subclass of *edNLC* graph grammars are presented in this section. Some of them are analogous to those imposed of the *ETPL*(k) top-down parsable graph grammars [28, 29, 32].

Firstly, let us impose the constraint on the form of the right-hand side graphs in order to reduce the computational complexity of a single derivation step.

Definition 4.1. 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) $l \in \Sigma$,
   (b) D is the *rIE* graph having the characteristic description:

$$\begin{array}{cccccccc} n_1(1) & n_2(2) & \ldots & n_m(m) & \text{or} & n_1(1), & \text{where} & n_i(i) \\ r_1 & r_2 & \ldots & r_m & & 0 & & r_i \\ E_1 & E_2 & \ldots & E_m & & - & & E_i \\ I_1 & I_2 & \ldots & I_m & & - & & I_i \end{array}$$

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

Let us require a derivation process to be performed according to the linear ordering imposed on *rIE* graphs.

Definition 4.2. A *TLP* graph grammar G is called a *closed rTLP graph grammar* G if for each derivation of this grammar

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

a graph $g_i, i = 0, \ldots, n$ is an *rIE* graph.

Definition 4.3. Let there be given a derivation of a closed *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 right-hand side derivation*, denoted $\underset{\text{rr}(G)}{\Longrightarrow}$ if:

1. for each $i = 0, \ldots, n-1$ we apply a production for a node having the greatest) index in a graph g_i,
2. node indices do not change during a derivation.

A closed *rTLP* graph grammar rewriting graphs according to the regular right-hand side derivation is called a *closed rTLPO graph grammar*, abbrev. from *reverse Two-Level Production-Ordered.*

Now we introduce notions used for extracting handles in analyzed graphs which are matched against right-hand sides of productions during graph parsing.

Definition 4.4. Let g be an *rIE* graph, l some node of g defined by a characteristic description $n(l), r, e_1 \ldots e_r, i_1 \ldots i_r$. A subgraph h of the graph g consisting of node l, nodes having indices $i_{a+1}, i_{a+2}, \ldots, i_{a+m}, a \geq 0, a + m \leq r$, and edges connecting the nodes: $l, i_{a+1}, i_{a+2}, \ldots, i_{a+m}$ is called an *m-successors two-level graph originated in the node l and beginning with the* (i_{a+1})th *successor.* The subgraph h is denoted $h = m - TL(g, l, i_{a+1})$. By $0 - TL(g, l, -)$ we denote the subgraph of g consisting only of node l.

Definition 4.5. Let g be an *rIE* graph, l some node defined by a characteristic description $n(l), r, e_1 \dots e_r, i_1 \dots i_r$. A subgraph h of graph g consisting of node l, nodes having indices $i_{a+1}, i_{a+2}, \dots, i_r, a \geq 0$, and edges connecting the nodes $l, i_{a+1}, i_{a+2}, \dots, i_r$ is called a *complete two-level graph originated in node* l *and beginning with the* (i_{a+1})th *successor*. The subgraph h is denoted

$$h = CTL(g, l, i_{a+1}).$$

Let us impose the fundamental constraint which is analogous to that used in the definition of string Knuth's $LR(k)$ grammars [43]. It allows us to construct the efficient non-backtracking bottom-up parsing scheme.

Definition 4.6. Let $G = (\Sigma, \Delta, \Gamma, P, Z)$ be a closed *rTLPO* 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

$$Z \underset{\mathrm{rr}(G)}{\overset{*}{\Longrightarrow}} X_1 A X_2 \underset{\mathrm{rr}(G)}{\Longrightarrow} X_1 g X_2 \ ,$$

$$Z \underset{\mathrm{rr}(G)}{\overset{*}{\Longrightarrow}} X_3 B X_4 \underset{\mathrm{rr}(G)}{\Longrightarrow} X_1 g X_5 \ ,$$

and

$$k - TL(X_2, 1, 2) \overset{\mathrm{isom}}{=} k - TL(X_5, 1, 2) \ ,$$

where $\underset{\mathrm{rr}(G)}{\overset{*}{\Longrightarrow}}$ is the transitive and reflexive closure of $\underset{\mathrm{rr}(G)}{\Longrightarrow}$, A, B are characteristic descriptions of certain nodes, X_1, X_2, X_3, X_4, X_5 are substrings of characteristic descriptions, 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 concerns the embedding transformation. The *edNLC* embedding transformation operates at the border between the production and its context. So, we do not have the context freeness property stated that reordering of derivation steps does not influence the result of a derivation. The lack of the order independence property, related to the finite Church-Rosser, fCR, property, results in the intractability of parsing. Thus, we have to restrict the power of the embedding transformation in order to obtain fCR and to guarantee the parsing efficiency. We make it by preserving the part of the production context unchanged during a derivation step.

Definition 4.7. 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 $a \in \Sigma$, if there exists the *rIE* graph $g = (V, E, \Sigma, \Gamma, \phi)$ belonging to a certain regular left-hand (right-hand) side derivation in G that: $(k, x, l) \in E, \phi(k) = b$, and $\phi(l) = a$.

Definition 4.8. A $PR(k)$ graph grammar G is called an $ETPR(k)$, abbrev. from *Embedding Transformation-preserving Production-ordered k-Right nodes unambiguous, graph grammar*, if: for each production of the form

$$(l)\quad A \longrightarrow \begin{array}{cccc} X_1(1) & X_2(2) & \dots & X_m(m) \\ r_1 & r_2 & \dots & r_m \\ E_1 & E_2 & \dots & E_m \\ I_1 & I_2 & \dots & I_m \end{array}$$

$$\text{where } X_a \neq X_b, a, b = 1, \dots, m\ .$$

If (b, y) is a potential previous context for A, then there exists only one $(X_i, b, z, in) \in C_l(y, in), i \in \{1, \dots, m\}$, where C_l is the embedding transformation of the lth production. If $i = 1$, then $z = y$, i.e. $(X_1, b, y, in) \in C_l(y, in)$.

Let $G = (\Sigma, \Delta, \Gamma, P, Z)$ be the *ETPR*(k) graph grammar. The language of G denoted $L(G)$ is the set

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

5. Parsing algorithm of *ETPR*(k) graph languages

The general scheme of parsing for *ETPR*(k) graph grammars [37] is a slight modification of the parsing schemes for *ETL*(1) [28] and *ETPL*(k) [32] graph grammars. It consists in a succeeding identification of a handle constructed on the basis of the property of an unambiguous choice of a production in the regular right-hand side derivation[6] according to Definition 4.6.

Let us introduce the following denotations and functions.

- G – an *rIE* graph to be analyzed represented by its characteristic description.
- H – the *rIE* graph represented by its characteristic description, which is being constructed (with succeeding reductions) during parsing on the basis of the graph G.
- *give_index*() – the function gives the succeeding index from the stack of node indices constructed according to Definition 3.7.
- *nonempty_indices_stack*() – the Boolean function gives *true* if the stack of node indices is nonempty.
- *give_handle*(H, i) – the function extracts the handle (in the form of its characteristic description) originated in the node indexed with i from the graph H according to Definition 4.6.
- *choose_production*(*handle*) – the function, on the basis of *handle*, identifies the proper production to be used for a reduction according to Definition 4.6.
- *reduction*(H, i, k) – the function performs the reduction to the node indexed with i in the graph H according to the production k.

We assume that the right-hand side graphs of the grammar are stored in the form of their characteristic descriptions.

Now, we can define the parsing algorithm 5.1.

[6]In *ETL*(1) and *ETPL*(k) graph grammars the corresponding property concerns the regular left-hand side derivation.

Algorithm 5.1 The parsing algorithm for *ETPR*(k) graph grammar

```
$H := G$;
$err := 0$;
while $err = 0$ and $nonempty\_indices\_stack()$ do
begin
        $i := give\_index()$;
        $handle := give\_handle(H, i)$;
        $k := choose\_production(handle)$;
        if $k = 0$ then $err := 1$ else $reduction(H, i, k)$;
end;
```

In order to evaluate the time complexity of Algorithm 5.1 let us analyze the running times of its functions. Let n be the number of nodes of the graph G. The running time of the function *give_index*() operating on the stack of indices is $\sim n$. Extracting the handle with the help of the function *give_handle*(H, i), if we assume that H is represented with its characteristic description, is $\sim n$, as well. The running time of the function *choose_production*(*handle*) is bounded by the constant c which depends on the size of the grammar, i.e. the number of its productions and the maximum size of the right-hand size graph. (Thus, c does not depend on the size of the input graph G.) The function *reduction*(H, i, k) is analogous to the function *production*(H, i, k) of the *ETPL*(k) parser introduced in [32]. Its running time is $\sim n$.

Now, we can formulate the following theorem.

Theorem 5.1. The running time of the parsing algorithm for *ETPR*(k) graph grammar (Algorithm 5.1) is $O(n^2)$, where n is the number of the nodes of the analyzed *rIE* graph.

Proof. The `while` loop of Algorithm 5.1 is performed at most n times. Since the running times of all the functions inside the loop are bounded either by a constant or by n, the running time of the algorithm is $O(n^2)$.

6. Concluding remarks

The deterministic subclasses of *Node Label Controlled (NLC)* graph grammars for syntactic pattern recognition and computer vision have been studied for thirty years. They have been used in a variety of the real-world applications. The possibility of imposing the linear ordering on *EDG* graphs is one of two crucial factors resulting in the high efficiency of the model. (The balanced restrictions imposed on the embedding transformation of *edNLC* grammars is the second key factor.) In all the aforementioned applications of the model the linear ordering has been defined on the basis of semantic features of patterns considered. However, the theoretical foundations of the scheme allowing us to index graph nodes based on pattern semantics have not been formulated till now. We have introduced them in the paper defining concepts of: relational structure, interpretation of *EDG* graphs over relational structure and interpreted *EDG* graph. These notions allow us to introduce the concept of (reverse) indexed edge-unambiguous graphs (*IE* and *rIE* graphs) in a formalized way.

The presentation of formal properties of $ETPR(k)$ graph grammars introduced preliminarily in [37] and their parsing algorithm has been the second goal of the paper. The $ETPR(k)$ parsing scheme is analogous to the $ETPL(k)$ one [32]. However, our experience with practical applications has revealed that there are some graph languages that cannot be generated by $ETPL(k)$ grammars and can be generated by $ETPR(k)$. This problem is worth further studying. Therefore, similarly as in case of $ETPL(k)$ grammars which are characterized from the point of view of descriptive power [33], the research into power properties of $ETPR(k)$ graph grammars will be carried out and the results obtained will be the subject of further publications.

References

[1] T. Pavlidis. *Structural Pattern Recognition*. Springer, New York, 1977.

[2] R.C. Gonzales and M.G. Thomason. *Syntactic Pattern Recognition: An Introduction*. Addison-Wesley, Reading, 1978.

[3] K.S. Fu. *Syntactic Pattern Recognition and Applications*. Prentice Hall, Englewood Cliffs, 1982.

[4] H. Bunke and A. Sanfeliu (eds.). *Syntactic and Structural Pattern Recognition – Theory and Applications*. World Scientific, Singapore, 1990.

[5] V. Claus, H. Ehrig and G. Rozenberg (eds.). Graph Grammars and Their Application to Computer Science and Biology. *Lecture Notes in Computer Science* 73, 1979. `doi:10.1007/BFb0025713`.

[6] H. Ehrig, M. Nagl and G. Rozenberg (eds.). Graph Grammars and Their Application to Computer Science. *Lecture Notes in Computer Science* 153, 1983. `doi:10.1007/BFb0000094`.

[7] H. Ehrig, M. Nagl and G. Rozenberg (eds.). Graph Grammars and Their Application to Computer Science. *Lecture Notes in Computer Science* 291, 1987. `doi:10.1007/3-540-18771-5`.

[8] H. Ehrig, H.-J. Kreowski and G. Rozenberg (eds.). Graph Grammars and Their Application to Computer Science. *Lecture Notes in Computer Science* 532, 1991. `doi:10.1007/BFb0017372`.

[9] J. Cuny, H. Ehrig, G. Engels and G. Rozenberg (eds.). Graph Grammars and Their Application to Computer Science. *Lecture Notes in Computer Science* 1073, 1996. `doi:10.1007/3-540-61228-9`.

[10] H. Ehrig, G. Engels, H.-J. Kreowski and G. Rozenberg (eds.). Theory and Application of Graph Transformations. *Lecture Notes in Computer Science* 1764, 2000. `doi:10.1007/b75045`.

[11] A. Corradini, H. Ehrig, H.-J. Kreowski and G. Rozenberg (eds.). Graph Transformations. *Lecture Notes in Computer Science* 2505, 2002. `doi:10.1007/3-540-45832-8`.

[12] H. Ehrig, G. Engels, F. Parisi-Presicce and G. Rozenberg (eds.). Graph Transformations. *Lecture Notes in Computer Science* 3256, 2004. `doi:10.1007/b100934`.

[13] A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro and G. Rozenberg (eds.). Graph Transformations. *Lecture Notes in Computer Science* 4178, 2006. `doi:10.1007/11841883`.

[14] H. Ehrig, R. Heckel, G. Rozenberg and G. Taentzer (eds.). Graph Transformations. *Lecture Notes in Computer Science* 5214, 2008. doi:10.1007/978-3-540-87405-8.

[15] H. Ehrig, A. Rensink, G. Rozenberg and A. Schürr (eds.). Graph Transformations. *Lecture Notes in Computer Science* 6372, 2010. doi:10.1007/978-3-642-15928-2.

[16] H. Ehrig, G. Engels, H.-J. Kreowski and G. Rozenberg (eds.). Graph Transformations. *Lecture Notes in Computer Science* 7562, 2012. doi:10.1007/978-3-642-33654-6.

[17] H. Giese and B. König (eds.). Graph Transformations. *Lecture Notes in Computer Science* 8571, 2014. doi:10.1007/978-3-319-09108-2.

[18] F. Parisi-Presicce and B. Westfechtel (eds.). Graph Transformations. *Lecture Notes in Computer Science* 9151, 2015. doi:10.1007/978-3-319-21145-9.

[19] R. Echahed and M. Minas (eds.). Graph Transformations. *Lecture Notes in Computer Science* 9761, 2016. doi:10.1007/978-3-319-40530-8.

[20] Q.Y. Shi and K.S. Fu. Parsing and translation of attributed expansive graph languages for scene analysis. *IEEE Trans. Pattern Analysis Mach. Intell.*, 5:472–485, 1983. doi:10.1109/TPAMI.1983.4767426.

[21] H.O. Bunke and B. Haller. A parser for context free plex grammars. *Lecture Notes in Computer Science*, 411:136–150, 1990. doi:10.1007/3-540-52292-1_10.

[22] K.J. Peng, T. Yamamoto and Y. Aoki. A new parsing scheme for plex grammars. *Pattern Recognition*, 23:393–402, 1990. doi:10.1016/0031-3203(90)90026-H.

[23] K. Wittenburg, L. Weitzman and J. Talley J. Unification-based grammars and tabular parsing for graphical languages. *Journal Visual Languages Computing*, 2:347–370, 1991. doi:10.1016/S1045-926X(05)80004-7.

[24] F. Ferruci, G. Tortora, M. Tucci and G. Vitiello. A predictive parser for visual languages specified by relational grammars. In *Proc. IEEE Symp. Visual Lang. VL'94*, 245-252. doi:10.1109/VL.1994.363611.

[25] J. Rekers and A. Schürr. Defining and parsing visual languages with layered graph grammars. *Journal Visual Languages Computing*, 8:27–55, 1997. doi:10.1006/jvlc.1996.0027.

[26] D.Q. Zhang, K. Zhang and J. Cao. A context-sensitive graph grammar formalism for the specification of visual languages. *The Computer Jornal*, 44:186–200, 2001. doi:10.1093/comjnl/44.3.186.

[27] D. Janssens and G. Rozenberg. On the structure of node-label-controlled graph languages. *Information Sciences*, 20:191–216, 1980. doi:10.1016/0020-0255(80)90038-9.

[28] M. Flasiński. Parsing of edNLC-graph grammars for scene analysis. *Pattern Recognition*, 21:623–629, 1988. doi:10.1016/0031-3203(88)90034-9.

[29] M. Flasiński. Characteristics of edNLC-graph grammars for syntactic pattern recognition. *Computer Vision Graphics Image Processing*, 47:1–21, 1989. doi:10.1016/0734-189X(89)90050-9.

[30] M. Flasiński. Distorted pattern analysis with the help of Nodel Label Controlled graph languages. *Pattern Recognition*, 23:765–774, 1990. doi:10.1016/0031-3203(90)90099-7.

[31] M. Flasiński. Some notes on a problem of constructing the best matched graph. *Pattern Recognition*, 24:1223–1224, 1991. doi:10.1016/0031-3203(91)90147-W.

[32] M. Flasiński. On the parsing of deterministic graph languages for syntactic pattern recognition. *Pattern Recognition*, 26:1–16, 1993. doi:10.1016/0031-3203(93)90083-9.

[33] M. Flasiński. Power properties of NLC graph grammars with a polynomial membership problem. *Theoretical Computer Science*, 201:189–231, 1998. doi:10.1016/S0304-3975(97)00212-0.

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

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

[36] M. Flasiński. Use of graph grammars for the description of mechanical parts. *Computer-Aided Design*, 27:403–433, 1995. `doi:10.1016/0010-4485(94)00015-6`.

[37] M. Flasiński and Z. Flasińska. Characteristics of bottom-up parsable edNLC graph languages for syntactic pattern recognition. In L.J. Chmielewski et al., editors, *Computer Vision and Graphics: Proc. Int. Conf. ICCVG 2014*, volume 8671 of *Lecture Notes in Computer Science*, pages 195–202, Warsaw, Poland, September 2014. Springer, Heidelberg. `doi:10.1007/978-3-319-11331-9_24`.

[38] U. Behrens, M. Flasiński, L. Hagge and K. Ohrenberg. ZEX – an expert system for ZEUS. *IEEE Trans. Nuclear Science*, 41:152–156, 1994. `doi:10.1109/23.281478`.

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

[40] M. Flasiński. *Introduction to Artificial Intelligence*. Springer International, Switzerland, 2016. `doi:10.1007/978-3-319-40022-8`.

[41] M. Flasiński and S. Myśliński. On the use of graph parsing for recognition of isolated hand postures of Polish Sign Language. *Pattern Recognition* 43:2249–2264, 2010. `doi:10.1016/j.patcog.2010.01.004`.

[42] M. Flasiński. Syntactic pattern recognition: paradigm issues and open problems. In C.H. Chen (ed.), Handbook of Pattern Recognition and Computer Vision, World Scientific, New Jersey – London – Singapore, 2016, Chapt. 1, pp. 3-25.

[43] D.E. Knuth. On the translation of languages from left to right. *Information Control* 8:607–639, 1965. `doi:10.1016/S0019-9958(65)90426-2`.