Knowledge representation using graph grammar rewriting system

Jiří Zuzaňák, Pavel Zemčík
Graph@FIT
Department of Computer Graphics and Multimedia
Faculty of Information Technology
Brno University of Technology, Brno 612 66, Czech Republic
{izuzanak,zemcik}@fit.vutbr.cz

ABSTRACT

Graph rewriting systems are applicable to vast majority of problems that are being solved in computer science. From problems concerning program optimization, software verification, description, and parsing of structured information to graph programming languages and layout algorithms. Graph rewriting systems are often represented as sets of rules describing transformations on graphs. The graph rewriting rule encapsulates complete information about applicable graph modification. In context of the described graph rewriting system, rules represent atomic modification of graph. A novel approach to graph rewriting and criteria for rule application enabling development of exhaustive graph rewrite system are introduced. The presented approach is derived from the well known double pushout approach (DPO). This paper concentrates on discussion of knowledge formalization representation for modeling concepts and on application of these concepts using the proposed programmed graph rewriting system.

Keywords: Graph rewriting, Knowledge representation, Graph grammars, Image processing, Computer vision

1 INTRODUCTION

Most of the computing techniques can be simulated by graph rule based modifications performed on models. Systems from Chomsky hierarchy of grammars based on string transformations are used for modelling of various languages; similarly, graph rewrite systems (GRS) based on graph transformations can be used to transform models based on graphs.

This way, the proper set of graph transformations (graph rewriting rules) can be used to describe semantics of the simulated process. Such process can be e.g. program optimization, software verification, parsing of complex structured information, layout algorithms, or modification of complex networks. Traffic networks, simulation of chemical reactions, construction of artificial neural networks, etc. represents example of networks which can be modeled and transformed by graph rewriting systems. Follows few examples (described in more details) of application of graph rewriting systems.

In [11] traffic, networks are modelled by graph rewriting system. Result of graph rewriting process is Time Transition Petri Net (TTPN). Generated Petri Nets can be directly used for simulation of particular traffic situations. The graph grammars in this work define semantics of a given start model as all the reachable models that results from the application of rules.

Graph transformations are used also for simulation of chemical reactions. In [15] reactions are modelled by set of edge relabeling graph transformations. Reaction is based on transformation of substrate chemical graph to a product chemical graph. Transformation of is performed by breaking existing bonds and creating new bonds between molecule atoms. Atoms of chemical molecule are represented by vertices, while bonds between atoms are represented by graph edges.

Graph rewriting systems can be also exploited for rewriting of specialized patterns such are terms ([6, 2, 3]) and already mentioned strings. The theory of term graph rewriting studies the issue of representing finite terms as directed, acyclic graphs, and of modeling term rewriting by graph rewriting. Main advantage of this approach is sharing of common subterms explicitly, avoiding to copy subterm when applying rewrite rule with more than one occurrence of a variable in its right-hand side.

Graph rewriting is also used for performing basic computations (atomic steps) in graph programming (GP) languages [14]. GP languages are based on modification of input graphs, and related transformation of vertex and edge labels. Transformations are determined by set of rewriting rules, where rules are part of graph algorithm. Resulted graph programming language is suitable for solving graph problems at a high level of abstraction, freeing programmers from handling low-level data structures.

Graph rewriting systems can be used as computational model for more general class of languages than just GP languages. Functional and logical programming languages for example. Whilst implementation of functional languages by graph rewriting is simple and intuitive, the implementation of logic programming languages is less direct and thus is more limited in practice. In [13] is proposed approach applying graph rewriting system as computational model for logic programming language. The achieved results are demonstrated on implementation of Prolog, and more novel logic programming language PLL.

In this paper, description of a novel approach for graph rewriting and transformation based on programmed graph rewriting system is presented. The graph rewriting system is designed mainly for parsing and interpreting of knowledge retrieved from image or video by various image processing and computer vision algorithms. However, presented graph rewriting system is designed in generally for use in any system which can benefit from graph transformations. The presented system and algorithms are designed especially for efficiency of execution.

This paper is organized as follows. In Section 2, basic definitions and notations are presented. Section 3 contains description of graph rewrite system basics (graph rewriting rules, left-side matching and rule application). Section 4 briefly describes proposed graph rewriting system. Implementation of described graph rewriting system is discussed in Section 5. Finally in Section 6 the discussion of the results and proposed ideas for future work concludes the paper.

2 BASIC DEFINITIONS

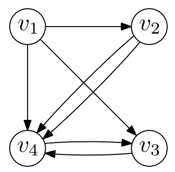
In this section, definitions and notations further required for description of graph rewriting system will be introduced. Basic concepts of graph representation and graph properties and also relations among graphs will be described in details.

2.1 Basic Definitions

For graph rewriting system and graphs itself to be usable, it is important to introduce a possibility to evaluate vertices and edges of graph by labels. Let labels be set of arbitrary objects of same class. Also, for further definitions, basic concepts of deterministic finite state machine will be needed.

Definition 1 (Labels) *Let $\mathcal{L}$ be a set of values, the* labels. *The relation equal $\subseteq \mathcal{L} \times \mathcal{L}$ determines equality of elements of $\mathcal{L}$. We will denote $(x,y) \in equal$ by $x =_{equal} y$*

Relation equal should be relation of equivalence (reflexive, symmetric and transitive).



$\sigma_{t_G} \circ \sigma_{s_G}^{-1}$	v_1	v_2	v_3	v_4
v_1	0	1	1	1
v_2	0	0	0	1
v_3	0	0	0	1
v_4	0	0	1	0

Figure 1: Example of directed multigraph

Definition 2 (Deterministic finite state machine) *The* deterministic finite state machine *(DFSM) is quintuple $M = (\Sigma, S, s_0, \delta, F)$, where Σ is input alphabet, S is non-empty set of states, $s_0 \in S$ is machine initial state, $\delta : S \times \Sigma \to S$ is deterministic state transition function, and $F \subseteq S$ is set of final states.*

2.2 Graph Representation

Most important structure in graph rewriting systems is the graph itself. From now on, reference to a graph will refer to directed graph with multi-edges and loops enabled. (Directed multigraph with loops)

Definition 3 (Directed multigraph) *A* directed multigraph *G is tuple $G = (V, E, \sigma_s, \sigma_t, \mu_v, \mu_e)$, where V is set of vertices, E is set of edges, $\sigma_s : E \to V$, and $\sigma_t : E \to V$ are functions mapping edges to theirs source and target vertices, and $\mu_v : V \to \mathcal{L}$, and $\mu_e : E \to \mathcal{L}$ are functions mapping vertices and edges to set of labels $\mathcal{L}$*

For text simplification we will write $G = (V_G, E_G, \dots)$ for graph $G = (V_G, E_G, \sigma_{s_G}, \sigma_{t_G}, \mu_{v_G}, \mu_{e_G})$.

Example 1 *Follows example of simple directed multigraph $G = (V_G, E_G, \sigma_{s_G}, \sigma_{t_G}, \mu_{v_G}, \mu_{e_G})$, where $V_G = \{v_1, \dots, v_4\}$, $E_G = \{e_1, \dots, e_7\}$, $\sigma_{s_G} = \{(e_1, v_1), (e_2, v_1), (e_3, v_1), (e_4, v_2), (e_5, v_2), (e_6, v_3), (e_7, v_4)\}$, and $\sigma_{t_G} = \{(e_1, v_2), (e_2, v_3), (e_3, v_4), (e_4, v_4), (e_5, v_4), (e_6, v_4), (e_7, v_3)\}$, and $\mu_{v_G} = \mu_{e_G} = \emptyset$. Example of graph is displayed in Fig. 1.*

Denotation 1 *Several functions are implicitly associated with graph G*

We say that edge e is incident *to $\sigma_s(e)$ and $\sigma_t(e)$, and if $\sigma_s(e) \neq \sigma_t(e)$ then vertices $\sigma_s(e)$ and $\sigma_t(e)$ are* adjacent.

Set of all edges incident *to vertex v is defined by function $inc(v) = \{e \mid v = \sigma_s(e) \vee v = \sigma_t(e)\}$; similarly, set of all vertices* adjacent *to vertex v is defined by function $adj(v) = \{u \mid (u = \sigma_s(e) \wedge v = \sigma_t(e)) \vee (u = \sigma_t(e) \wedge v = \sigma_s(e))\}$*

The function $in : V \to \mathbb{N}$ determines the number of incoming incident edges of a vertex: $in(v) = |\{e \mid v = \sigma_t(e)\}|$. Similarly, the function out determines the number of outgoing incident edges: $out(v) = |\{e \mid v = \sigma_s(e)\}|$. A vertex with $in(v) = 0$ is called root. *A vertex with $out(v) = 0$ is called* sink

The set of all graphs over set of labels $\mathcal{L}$ is denoted as $\mathcal{G}_{\mathcal{L}}$

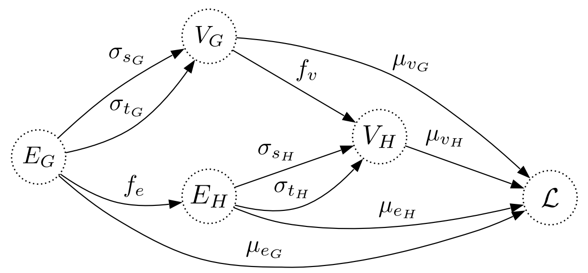


Figure 2: Graphs G and H and morphism $f : G \to H$

Several relations among graphs are defined. From basic graph unions and intersections to subgraph relation and graph isomorphisms, that are essential for graph rewriting systems.

Definition 4 (Graph union and intersection) *A graph $I = G \cup H$ is called* union *of graphs G and H, if $I = (V_I, E_I, \ldots)$, where $V_I = V_G \cup V_H$, $E_I = E_G \cup E_H$, $\sigma_{s_I} = \sigma_{s_G} \cup \sigma_{s_H}$, $\sigma_{t_I} = \sigma_{t_G} \cup \sigma_{t_H}$, $\mu_{v_I} = \mu_{v_G} \cup \mu_{v_H}$, and $\mu_{e_I} = \mu_{e_G} \cup \mu_{e_H}$. Graph* intersection *$I = G \cap H$, and graph* difference *$I = G \setminus H$ are defined similarly*

Definition 5 (Subgraph) *A graph $H = (V_H, E_H, \ldots)$ is* subgraph *of graph G, if $E_H \subseteq E_G$, $V_H = \{v \mid e \in E_H \wedge (v = \sigma_{sG}(e) \vee v = \sigma_{tG}(e))\} \cup (V_{arb} \subseteq V_G)$, $\sigma_{s_H} = \sigma_{s_G} \cap (E_H \times V_H)$, $\sigma_{t_H} = \sigma_{t_G} \cap (E_H \times V_H)$, $\mu_{v_H} = \mu_{v_G} \cap (V_H \times \mathcal{L})$, and $\mu_{e_H} = \mu_{e_G} \cap (E_H \times \mathcal{L})$, where V_{arb} is arbitrary subset of V_G. Subgraph is denoted by $H \subseteq G$*

Definition 6 (Graph morphism and isomorphism) *A* graph morphism *$f : G \to H$ between two graphs G and H consists of two functions $f_v : V_G \to V_H$ and $f_e : E_G \to E_H$ that preserve labels and attachment to vertices, that is, $\mu_{v_H} \circ f_v = \mu_{v_G}$, $\mu_{e_H} \circ f_e = \mu_{e_G}$, $\mu_{v_H} \circ \sigma_{s_H} \circ f_e = \mu_{v_G} \circ \sigma_{s_G}$, and $\mu_{v_H} \circ \sigma_{t_H} \circ f_e = \mu_{v_G} \circ \sigma_{t_G}$*

Functions illustrating connections of vertices and edges of graphs G and H, and morphism between them $f : G \to H$ are depicted in Fig. 2.

The graph morphism f is *injective* (*surjective*) if f_v and f_e are. If f is both injective and surjective, then it is an *isomorphism*. In this case graphs G and H are isomorphic, which is denoted by $G \cong H$

Definition 7 (Subgraph isomorphism) *A* subgraph isomorphism *f from graph H to graph G is graph isomorphism $f : H \to S$, where $S \subseteq G$. Denoted by $H \cong (S \subseteq G)$, or $f : H \to (S \subseteq G)$*

3 GRAPH REWRITING

Basic building block of graph rewriting system is graph rewriting rule, describing one possible modification (transformation) of the target host graph. In vast majority of literature ([11, 15, 14]) graph rule is represented by its left-hand side, right-hand side, and set of connections. Left-hand side of such rule can be represented by node, edge, or graph, from which rewriting system get its name: node-, edge-, graph-replacement systems. Right-hand side of rule is in most cases represented by graph. The connections describe relation between left-hand and right-hand side of rule. In many cases, connections also describe how to compute (find) labels of vertices and edges newly inserted into the host graph.

This paper is restricted to graph-rewriting systems, thus node and edge-rewriting systems are not considered. Most common approach for graph transformation is *Double Pushout Approach* (DPO) ([4, 7, 8]) which has rewriting rules of form:

$$r : L \leftarrow^{l} K \rightarrow^{r} R \tag{1}$$

where L is left-hand side, K is interface graph, R is right-hand side, and l and r are morphisms. K represents interface that is common for L and R. Transformation of graph G to graph H by rule r describe diagram in Equation 2.

$$\begin{array}{ccccc} L & \leftarrow^{l} & K & \rightarrow^{r} & R \\ \downarrow^{m} & & \downarrow^{d} & & \downarrow^{m^*} \\ G & \leftarrow^{l^*} & D & \rightarrow^{r^*} & H \end{array} \tag{2}$$

In order to apply rule to graph G the match m should be found between L and G. In next step are from G deleted all elements $L \setminus K$, thus producing graph D. Finally to produce graph H elements from $R \setminus K$ are added to graph D.

Definition 8 (Reducible expression) *Reducible expression (redex) represents subgraph of host graph, to which is left side of rule mapped.*

Approach to graph rewriting presented in this paper is inspired by the DPO approach. But in contrary to DPO, the proposed approach works as follows: the first step of a rule rewrite, once a redex has been located, is to glue into the host graph new structure (represented by right-side); then change the shape of the graph by redirecting edges. Finally redundant structure (the garbage) is removed.

Beside the connections in existing implementations, also negative application conditions (NAC), are used, that are left-hand side context information restricting rule from application. This technique adds dependency on contextual information to process of left-hand side matching. Such extension of this process boosts expressive power of graph rewriting system.

In the presented approach left-hand side of rule is represented by general graph. This graph should be directed multigraph with loops, and must be weakly connected (in graph exist non oriented path from any vertex to each other vertex). Above mentioned constraint is requirement of algorithm for subgraph isomorphism search.

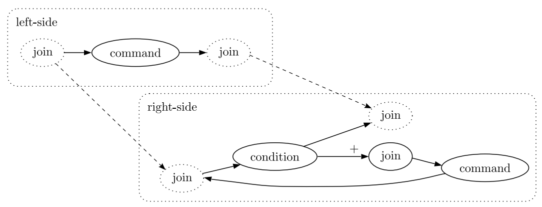


Figure 3: Example of simple graph rewriting rule

Definition 9 (Graph rewrite rule) *The quadruplet* $r = (L, R, E_{ex}, join)$ *is* graph rewrite rule*, which consist of* left-hand side $L \in \mathcal{G}_{\mathcal{L}}$*,* right-hand side $R \in \mathcal{G}_{\mathcal{L}}$*, set of* excluded edges $E_{ex} \subseteq E_L$*, and function* $join : V_L \to V_R$

The set of excluded edges E_{ex} determines edges that must not be present in host graph in order to rule be applicable (NAC). Function *join* defines connection (interface) of graph rule in host graph. Example of simple graph rewriting rule is depicted on Fig. 3, where function *join* is expressed by dashed arcs, and $E_{ex} = \emptyset$.

Definition 10 (rule matching) *The graph rewriting rule* $r = (L, R, E_{ex}, join)$ *is applicable to host graph* H *if:*

1. *graph isomorphism* $f_r : L_w \to (S_L \subseteq H)$ *exists, where* $L_w = (V_L, E_L \setminus E_{ex}, \sigma_{s_L}, \sigma_{t_L}, \mu_{v_L}, \mu_{e_L})$.
2. *the following holds for every* $u_l \in V_L$ *and* $v \in V_H$*:* $v = f_{r_v}(u_l) \wedge (in_{L_w}(u_l) < in_H(v) \vee out_{L_w}(u_l) < out_H(v)) \Rightarrow u_r \in V_R \wedge (u_l, u_r) \in join$
3. *the following holds for every* $e_{ex} \in E_{ex}$ *and* $e \in E_H$*:* $\sigma_{s_H}(e) \neq f_{r_v}(\sigma_{s_L}(e_{ex})) \vee \sigma_{t_H}(e) \neq f_{r_v}(\sigma_{t_L}(e_{ex})) \vee \mu_{e_H}(e) \neq_{equal} \mu_{e_L}(e_{ex})$

Graph L_w is constructed by removing edges of set E_{ex} from graph L. Graph isomorphism f_r is called *occurrence*, and graph $S_L = (V_{S_L}, E_{S_L}, \ldots)$ is called *redex* (reducible expression). Statement 2 assures that all vertices of graph L that are not in *join* relation with some vertex in graph R have the same output and input degree as their image in graph H. Statement 3 assures that in graph H, edges that can be interpreted as images of edges from E_{ex} are not present.

Definition 11 (Graph rule match) *The* Graph rule match *is a triplet* $m = (r, H, f_r)$*, where* r *is matched graph rewriting rule,* H *is host graph, and* f_r *is matching morphism, which meet all conditions from Definition 10*

Evaluation function determines labels of new elements (vertices, edges) in host graph. This function is evaluated for every element $e \in V_R \cap E_R$ of right-side graph R. Every of these elements has associated one of evaluation heuristics.

Definition 12 (Evaluation function) *The method for determining labels of new elements of host graph is encapsulated in evaluation function* $eval : (e, m) \to \mathcal{L}$*, where* $e \in V_R \cup E_R$*, and* m *is graph rule match*

Rule application

Host graph H is transformed into graph I by rule r in following steps:

1. Construct graph $I_1 = (V_{I_1}, E_H, \sigma_{s_H}, \sigma_{t_H}, \mu_{v_{I_1}}, \mu_{e_H})$, where
   - $V_{I_1} = V_H \cup V_R$
     - let $f_{n_v} : V_R \to V_{I_1}$ is injective function mapping right-hand side vertices to new vertices in host graph
   - $\mu_{v_{I_1}} = \mu_{v_H} \cup \{(v, l) \mid v_r \in V_R \wedge v = f_{n_v}(v_r) \wedge l = eval(v_r, m)\}$

   Vertices from right-side of rule are inserted to host graph. Labels of vertices are evaluated, and function mapping original right-side vertices to new vertices of host graph is constructed.

2. Construct graph $I_2 = (V_{I_1}, E_{I_2}, \sigma_{s_{I_2}}, \sigma_{t_{I_2}}, \mu_{v_{I_1}}, \mu_{e_{I_2}})$, where
   - $E_{I_2} = E_H \cup E_R$
     - let $f_{n_e} : E_R \to E_{I_2}$ is injective function mapping right-hand side edges to new edges in host graph
   - $\sigma_{x_{I_2}} = \sigma_{x_H} \cup \{(e, v) \mid e_r \in E_R \wedge e = f_{n_e}(e_r) \wedge v = f_{n_v}(\sigma_{x_R}(e))\}$
   - $\mu_{e_{I_2}} = \mu_{e_H} \cup \{(e, l) \mid e_r \in E_R \wedge e = f_{n_e}(e_r) \wedge l = eval(e_r, m)\}$

   for $x \in \{s, t\}$, (hence σ_{x_G} stands for both σ_{s_G} and σ_{t_G}). Edges of right-side of rule are inserted to host graph (edges are defined on already inserted vertices). Labels of edges are evaluated, and function mapping edges of right-side to new edges of host graph is constructed. Graph I_2 is depicted at Fig. 4.b.

3. Construct graph $I_3 = (V_{I_1}, E_{I_2}, \sigma_{s_{I_3}}, \sigma_{t_{I_3}}, \mu_{v_{I_1}}, \mu_{e_{I_2}})$, where
   - $\sigma_{x_{I_3}} = (\sigma_{x_{I_2}} \cup \{(e, f_{n_v}(v)) \mid (u, v) \in join \wedge e = \sigma^{-1}_{x_H}(f_{r_v}(u))\}) \setminus \{(e, f_{r_v}(u)) \mid (u, v) \in join \wedge e = \sigma^{-1}_{x_H}(f_{r_v}(u))\}$

   for $x \in \{s, t\}$. Edges connecting join vertices of left-side of graph rewrite rule are redirected to join vertices of right-side of graph rewrite rule. Resulting graph I_3 is depicted at Fig. 4.c., where redirected edges are displayed as dashed arcs.

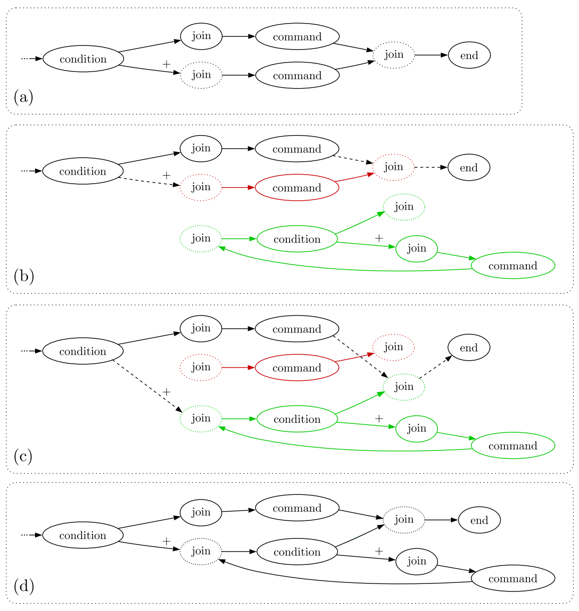


Figure 4: Illustration of host graph rewrite, driven by rule from Fig. 3

4. Construct graph $I_4 = (V_{I_1}, E_{I_4}, \sigma_{s_{I_4}}, \sigma_{t_{I_4}}, \mu_{v_{I_1}}, \mu_{e_{I_4}})$, where

   - $E_{I_4} = E_{I_2} \setminus \{e \mid e_l \in E_{L_w} \wedge e = f_{r_e}(e_l)\}$
   - $\sigma_{x_{I_4}} = \sigma_{x_{I_3}} \setminus \{(e,v) \mid e_l \in E_{L_w} \wedge e = f_{r_e}(e_l) \wedge v = \sigma_{x_H}(e)\}$
   - $\mu_{e_{I_4}} = \mu_{e_{I_2}} \setminus \{(e,l) \mid e_l \in E_{L_w} \wedge e = f_{r_e}(e_l) \wedge l = \mu_{e_H}(e)\}$

   for $x \in \{s,t\}$. Edges of rule left-side including their labels and mappings to source and target vertices are removed from host graph.

5. Construct graph $I = (V_I, E_I, \sigma_{s_I}, \sigma_{t_I}, \mu_{v_I}, \mu_{e_I})$, where

   - $V_I = V_{I_1} \setminus \{v \mid v_l \in V_L \wedge v = f_{r_v}(v_r)\}$
   - $\mu_{v_I} = \mu_{v_{I_4}} \setminus \{(v,l) \mid v_l \in V_L \wedge v = f_{r_v}(v_r) \wedge l = \mu_{v_H}(v)\}$
   - $E_I = E_{I_4}, \sigma_{s_I} = \sigma_{s_{I_4}}, \sigma_{t_I} = \sigma_{t_{I_4}}, \mu_{e_I} = \mu_{e_{I_4}}$

   Finally in last step vertices of rule left-side are removed from host graph. Graph I (after removing left-side vertices and edges) is depicted at Fig. 4.d.

Description of application of rule is tightly connected to implementation. Vertices and edges of right-hand side are inserted to host graph before any left-hand side vertices or edges are removed, so labels of new vertices and edges can be determined. Rewrite of host graph by rule from Fig. 3 is inllustrated in Fig. 4.

4 GRAPH REWRITE SYSTEM

The literature concerning graph rewriting reports on various methods of organizing a collection of graph rewriting rules. These can be unordered, ordered, or event-driven. Choice of rule organization system largely affects the number of rewrite rule applications that must be tested during graph rewriting system execution. Parsing by graph grammar normally (without any rule organisation system) requires frequent testing of inapplicable rules. In contrast, an ordered graph rewriting system can directly transform an input graph into required output graph. Event-driven graph rewriting systems are highly time-efficient, applied rules are used only as direct response to external action.

Unordered Graph-rewriting System

A set of graph rewriting rules. Rewrites the host graph by nondeterministically chosen rules until no further rule apply.

Graph Grammar

A set of graph-rewrite productions. A starting host graph. A designation of labels as terminal or nonterminal. The starting graph is transformed by graph-rewrite productions until terminal graph is obtained. The set of terminal graphs that can be generated by this process is called language of the grammar (generative use). Parsing given graph: find sequence of rewrite productions that derive given graph from start graph (recognition use).

Ordered graph rewriting system

A set of graph rewriting rules. A control specification (complete or partial ordering of rule-application). Rewrite the given host graph (choosing nondeterministically among applicable rules according to control specification) until a final state in control specification is reached.

Event-driven Graph-rewriting System

A set of graph-rewrite rules. A externally-arising sequence of events. Rewrite the initial host graph: rewrite rules are executed in response to events.

Presented approach is restricted to first three of introduced categories of graph rewriting (respective graph grammar) systems. It is possible to extend presented rewriting system by event-driven execution of graph rewriting rules, but it was not introduced so far. Definitions of graph rewriting systems follows.

Definition 13 (Graph rewrite system) *The* graph rewrite system *(GRS), is represented by set of graph rewriting rules. In short it can be denoted just by $\mathcal{R}$.*

Definition 14 (Graph grammar) *The* graph grammar *(GG) is triple $\mathcal{X}_G = (\mathcal{P}, G_s, \mathcal{L}_t)$, where $\mathcal{P}$ is set of graph rewriting productions, $G_s \in \mathcal{G}_{\mathcal{L}}$ is starting host graph, and $\mathcal{L}_t \subseteq \mathcal{L}$ is set of terminal labels.*

Rules of graph rewrite system $\mathcal{X}$ are applied to host graph in nondeterministic order. This can result (for non confluent graph rewrite systems) in nondeterministic results. Based on this fact need arise to introduce ordering for rule applications of graph rewriting system.

Definition 15 (GRS with priorities) *The* GRS with priorities *is pair $\mathcal{X}_p = (\mathcal{R}, p)$, where $\mathcal{R}$ is set of graph rewriting rules, and $p : \mathcal{R} \to \mathbb{N}$ is function assigning priority number to each rule. This GRS represent simple version of ordered graph rewriting system.*

Definition 16 (GRS driven by DFSM) *The* GRS driven by DFSM *is defined as pair $\mathcal{X}_M = (\mathcal{R}, M_r)$, where $\mathcal{R}$ is set of graph rewriting rules, and $M_r = (\mathcal{R}, S, s_0, \delta, \emptyset)$ is deterministic finite state machine. A string accepted by M_r defines string of the used graph rewriting rules. The set of applicable rules is in actual state s given as $\mathcal{R}_s = \{r \mid (s, r) \to s_x \in \delta\}$. GRS driven by DFSM stops when $\mathcal{R}_s = \emptyset$, it works while any applicable rules exist. As in previous case this GRS presents ordered graph rewriting system.*

According to categorization of graph rewriting systems no explicit stopping conditions are introduced. The rules of graph rewrite system are applied while at least one matching of applicable graph rule exists.

5 GRAPH REWRITING IMPLEMENTATION

In this section the implementation of approach which was theoretically described in above text is described. Implementation of the proposed approach corresponds to the introduced theory.

5.1 Graph Representation

The graph is represented by a dynamic structure that can be arbitrarily modified with no impact on performance. Graphs represented by proposed structure can be updated and modified using elementary steps consisting of edge and vertex removal and insertion.

In short, graph is represented by set of vertices and set of edges between these vertices. Each edge is described by two (source and target) indexes to the set of vertices and each vertex contains an array indexing all its incident edges (for faster computations). Each of graph elements (vertices, edges) has associated label which represents its evaluation. Described structure enables for simple execution of sophisticated graph algorithms, such as detection of spanning tree, testing bipartite graph, testing completeness, counting of graph components, search for shortest path, and other graph processing algorithms.

5.2 Vertex and Edge Evaluation

The possibility to evaluate vertices and edges of graph by labels is provided by dynamically linked libraries defining operations over labels, such as comparison, copying, serialization, de-serialization etc. These libraries are created by graph rewrite system user, thus enabling use of arbitrary type of labels of vertices and edges of graph. For some conventional data types (integers, strings, etc.), default set of functions is defined.

5.3 Graph Isomorphism Matching

From the definition of graph rewriting system, it can be seen that left-sides of graph rules are known a priory to their matching in host graph. This fact is exploited in system for detection of subgraph isomorphisms. Graph parsing automata is created representing subgraph isomorphism detector for left-side of each rule of graph rewriting system. Such automata is optimized for detection of common patterns in rules left-sides and exploits this knowledge in order to boost isomorphism detection speed.

The states of graph automata describes partial mapping of vertices and edges (spanning subtree) to a hypothetical host graph. The transitions between these states are described using edges, their properties, and by properties of target vertices (evaluation, in-out degree, directions, etc.). each of prototype subgraphs (left-sides of rules) has at least one final node, and by reaching of this node, the vertex mapping between this subgraph and host graph is decided.

Detection of subgraph and graph isomorphisms consists of two fundamental steps:

The first step is search for vertex permutations (mappings) of subgraph vertices to host graph vertices, thus describing set of functions F_{vd}. For which holds $F_v \subseteq F_{vd}$, where F_v is set of functions describing proper subgraph vertex mappings. Detection of set of vertex mapping functions F_{vd} is accomplished by search for a subgraph spanning tree.

The second step of subgraph (graph) isomorphism detection consist of search for edge permutations F_e. For each vertex mapping function $f_v \in F_{vd}$, possible permutations of edges are searched. Every found edge mapping function f_e is added to set F_e; if $f_v \notin F_v$ then f_v is inserted into set F_v, and the record $f_e \to f_v$ is inserted into the injective function π.

Result of this procedure is triple (F_v, F_e, π), where F_v is set of vertex mapping functions, F_e is set of edge mapping functions, and $\pi : F_e \to F_v$ is function associating edge mapping function to vertex mapping function.

After detection of isomorphism of subgraph L_w in host graph H, context conditions represented by set of excluded edges E_{ex} are verified.

5.4 Graph Rewriting System

Graph rewriting system itself is represented as set of graph rewriting rules. To set of graph rules are associated priorities or DFSM. Each GRS is described by one input text file. GRS file refers to rules represented in form of text file in graph format `.dot`, used by graph visualising library graphviz. Important part of graph rewriting system is represented by dynamic library, which defines vertices and edges evaluation functions.

6 CONCLUSION

In this paper, an approach for graph rewriting system implementation was proposed and also its real implementation was discussed. Graph rewriting system was designed for use in graph grammars, which are created for applications concerning parsing and describing of knowledge retrieved from image or video. In both forms of grammar use, either as generating tool, or for parsing of retrieved graph.

The advantage of proposed graph rewriting system is its ability to work with general directed multigraphs with loops. These graphs are represented by dynamic structure, which is not designed specifically for proposed task, but is designed more generally to enable to perform any tasks on them. No restrictions are introduced on vertex and edge labels, and their evaluation in process of rules applications. Proposed GRS was implemented in language C/C++ and its implementation respects its theoretical design.

Future work concerning graph rewriting system implementation will concentrate on further optimisation of graph rewriting algorithm and also optimisation of graph matching algorithm. The plan is to introduce categories of rules, determined by their analysis; insert, remove, and alter for example, and optimise rewrite technique (algorithm) separately for each of them. Parallel application of rules, based on analysis of redex overlays. In context of subgraph isomorphism search: isomorphism search based on incremental actualization of set of detected isomorphisms.

Further work will also concern examination of graph rewriting system properties. Graph rewriting system termination, Church-Rosser property, confluence, and from them resulting convergence should be discussed and evaluated.

A survey of possible applications of designed and implemented graph rewriting system, other than description or representation of image knowledge is also planned. Applications that can be possibly implemented and tested in context of graph rewriting system are for example (as was mentioned in Introduction): simulation of traffic networks, chemical reactions, automatic construction of artificial neural networks, etc.

ACKNOWLEDGEMENTS

This work is supported by the European Commission under contract FP7-215453 - WeKnowIt.

This work was (also) supported by the project "Security Oriented Research in Information Technology" by Ministry of Eduction, Youth and Sports of Czech Republic no. MSM0021630528.

REFERENCES

[1] A study of two graph rewriting formalisms: Interaction nets and MONSTR, February 20 1998.

[2] Zena M. Ariola and Jan Willem Klop. Equational term graph rewriting. *Fundam. Inf.*, 26(3-4):207–240, 1996.

[3] R. Banach. Transitive term graph rewriting, April 30 1996.

[4] R. Banach. The contractum in algebraic graph rewriting, April 30 1998.

[5] Dorothea Blostein, Hoda Fahmy, and Ann Grbavec. Issues in the practical use of graph rewriting, December 18 1996.

[6] A. Corradini. A 2-categorical presentation of term graph rewriting, July 20 1997.

[7] Hartmut Ehrig. Introduction to the algebraic theory of graph grammars (a survey). In *Graph-Grammars and Their Application to Computer Science and Biology*, pages 1–69, 1978.

[8] Hartmut Ehrig. Tutorial introduction to the algebraic approach of graph grammars. In *Proceedings of the 3rd International Workshop on Graph-Grammars and Their Application to Computer Science*, pages 3–14, London, UK, 1987. Springer-Verlag.

[9] Hartmut Ehrig, Annegret Habel, Hans-Jörg Kreowski, and Francesco Parisi-Presicce. From graph grammars to high level replacement systems. In *Proceedings of the 4th International Workshop on Graph-Grammars and Their Application to Computer Science*, pages 269–291, London, UK, 1991. Springer-Verlag.

[10] P. Foggia, C. Sansone, and M. Vento. A database of graphs for isomorphism and sub-graph isomorphism benchmarking. In *CoRR*, pages 176–187, 2001.

[11] Pieter J. Mosterman Juan de Lara, Hans Vangheluwe. Modelling and analysis of traffic networks based on graph transformation.

[12] Christoph Klauck. Graph grammar based object recognition for image retrieval. In *ACCV '95: Invited Session Papers from the Second Asian Conference on Computer Vision*, pages 561–569, London, UK, 1996. Springer-Verlag.

[13] Peter M and Peter M C Brien. Implementing logic programming languages by graph rewriting, April 22 1999.

[14] Detlef Plump. The graph programming language gp. In *CAI '09: Proceedings of the 3rd International Conference on Algebraic Informatics*, pages 99–122, Berlin, Heidelberg, 2009. Springer-Verlag.

[15] Francesc Rosselló and Gabriel Valiente. Chemical graphs, chemical reaction graphs, and chemical graph transformation.

[16] Medha Shukla Sarkar, Dorothea Blostein, and James R. Cordy. GXL - A graph transformation language with scoping and graph parameters, September 12 1998.

[17] Vladimiro Sassone and Pawel Sobocinski. Coinductive reasoning for contextual graph-rewriting, February 02 2004.

[18] A. Schfürr. Programmed graph replacement systems. pages 479–546, 1997.

[19] Sjaak Smetsers. Term graph rewriting and strong sequentiality, November 06 1992.