Model Transformation with Triple Graph Grammars and Non-terminal Symbols

William da Silva[1,2(✉)], Max Bureck[1], Ina Schieferdecker[1,2], and Christian Hein[1]

[1] Fraunhofer Fokus, Berlin, Germany
{william.bombardelli.da.silva,max.bureck,ina.schieferdecker,
christian.hein}@fokus.fraunhofer.de
[2] Technische Universität Berlin, Berlin, Germany

Abstract. This work proposes a new graph grammar formalism, that introduces non-terminal symbols to triple graph grammars (TGG) and shows how to apply it to solving the model transformation problem. Our proposed formalism seems to suit code generation from models well, outperforms the standard TGG in the grammar size in one evaluated case and is able to express one transformation that we could not express with TGG. We claim, that such advantages make a formal specification written in our formalism easier to validate and less error-prone, what befits safety-critical systems specially well.

Keywords: NCE Graph Grammars · Triple Graph Grammars · Model Transformation · Model-based Development.

1 Introduction

Quality of service is a very common requirement for software projects, especially for safety-critical systems. A technique that aims to assure and enhance quality of software is the model-based development approach, which consists of the use of abstract models to specify aspects of the system under construction. The use of such models often allows for cheaper tests and verification as well as facilitates discussions about the system, for more abstract models tend to reduce the complexity of the actual object of interest.

The construction of a system with model-based development commonly requires the creation of various models at different levels of abstraction, in which case we are interested in generating models automatically from other models. One example of such a situation is the transformation of a UML diagram into source-code or the compilation of source-code into machine-code. This problem is known as model transformation. For safety-critical systems, automatic model transformation is attractive, because transformers can be verified for correction carefully once and used many times, whereas manually or ad-hoc transformations have to be checked each time.

Several approaches to solving the model transformation problem have been proposed so far. Some of them consist of using the theory of graph grammars to formalize models and describe relations between them. One of which is the

triple graph grammar (TGG) approach [16], which consists of building context-sensitive grammars of, so-called, triple graphs.

Triple graphs are composed of three graphs, the source and the target graphs, representing two models, and the correspondence graph that connects the source and the target through morphisms. A triple graph can be used to express the relationship between two graphs through the morphisms between their vertices. In this sense, a TGG describes a language of pairs of graphs whose vertices have a certain relationship. For the context of model transformation, in which one is interested in defining a translator from a source model to a target model, a TGG can be used to describe the set of all correctly translated source models and its correspondent target models, in form of a language of triple graphs.

Despite the various positive aspects of TGG, like a well-founded theory and a reasonable tool support [1], they may sometimes get too big or too difficult to be constructed correctly. We judge, this downside stems from the absence of the concept of non-terminal symbols in the TGG formalism. This concept allows, in the theory of formal languages, for a very effective representation of abstract entities in string grammars.

So, motivated by this benefit, we present in this paper a novel formalism that redefines the standard triple graph grammars and introduces the notion of non-terminal symbols to create a context-free triple graph grammar formalism, that has in some cases a smaller size and with which we could describe one transformation that we could not with standard TGG.

Our approach consists of (1) mixing an already existent context-free graph grammar formalism, called NCE graph grammar from [11], with the standard TGG formalism from [16], to create the NCE TGG and (2) constructing a model transformer that interprets a NCE TGG to solve the model transformation problem.

The remainder of this paper is as follows, in Section 2, we present the research publications related to this topic, in Section 3, we give the main definitions necessary to build our approach, in Section 4, we propose our modified version of TGG, the NCE TGG, in Section 5 we argue that our approach can be used for model transformation, in Section 6 we evaluate our results and, finally, in Section 7 we summarize and close our discussion.

2 Related Works

In this section, we offer a short literary review on the graph grammar and triple graph grammar approaches that are more relevant to our work. We focus, therefore, on the context-free node label replacement approach for graph grammars, although, there is a myriad of different alternatives to it, for example, the algebraic approach [5]. We refer to context-free grammars, inspired by the use of such classification for string grammars, in a relaxed way without any compromise to any definition.

In the node label replacement context-free formalisms stand out the *node label controlled graph grammar* (NLC) and its successor *graph grammar with*

neighborhood-controlled embedding (NCE). NLC is based on the replacement of one vertex by a graph, governed by embedding rules written in terms of the vertex's label [15]. For various classes of these grammars, there exist polynomial-time top-down and bottom-up parsing algorithms [7, 8, 15, 18]. The recognition complexity and generation power of such grammars have also been analyzed [6, 13]. NCE occurs in several formulations, including a context-sensitive one, but here we focus on the context-free formulation, where one vertex is replaced by a graph, and the embedding rules are written in terms of the vertex's neighbors [11, 17]. For some classes of these grammars, polynomial-time bottom-up parsing algorithms and automaton formalisms were proposed and analyzed [12, 4]. In special, one of these classes is the *boundary graph grammar with neighborhood-controlled embedding* (BNCE), that is used in our approach for model transformation.

Regarding TGG [16], a 20 years review of the realm is put forward by Anjorin et al. [1]. In special, advances are made in the direction of expressiveness with the introduction of application conditions [14] and of modularization [2]. Furthermore, in the algebraic approach for graph grammars, we have found proposals that introduce inheritance [3, 9] and variables [10] to the formalisms. Nevertheless, we do not know any approach that introduces non-terminal symbols to TGG with the purpose of gaining expressiveness or usability. In this sense, our proposal brings something new to the current state-of-the-art.

3 Graph Grammars and Triple Graph Grammars

In this section, we introduce important definitions that are used throughout this paper. First, we present definitions regarding graphs, taken mainly from [15], second, we introduce the NCE graph grammar [11, 12] and then, we express our understanding of TGG, backed by [16].

Definition 1. *A directed labeled graph G over the finite set of symbols Σ, $G = (V, E, \phi)$ consists of a finite set of vertices V, a set of labeled directed edges $E \subseteq V \times \Sigma \times V$ and a total vertex labeling function $\phi : V \to \Sigma$.*

We refer to directed labeled graphs often simply as graphs. For a fixed graph G we refer to its components as V_G, E_G and ϕ_G. Moreover, we denote the set of all graphs over Σ by $\mathcal{G}_\Sigma$. Two graphs G and H are disjoint if, and only if, $V_G \cap V_H = \emptyset$. If $\phi_G(v) = a$ we say v is labeled by a. In special, we do not allow loops (vertices of the form (v, l, v)), but multi-edges with different labels are allowed.

Two vertices v and w are neighbors if, and only if, $(v, l, w) \in E_G$ or $(w, l, v) \in E_G$. In this case, we say (v, l, w) and (w, l, v) are adjacent edges to v and to w.

Definition 2. *A morphism of graphs G and H is a total mapping $m : V_G \to V_H$.*

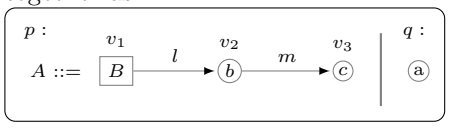
Definition 3. *An isomorphism of directed labeled graphs G and H is a bijective mapping $m : V_G \to V_H$ that maintains the connections between vertices and their labels, that is, $(v, l, w) \in E_G$ if, and only if, $(m(v), l, m(w)) \in E_H$ and $\phi_G(v) = \phi_H(m(v))$.*

If there exists an isomorphism of G and H, then G and H are said to be isomorphic and we denote the equivalence class of all graphs isomorphic to G by $[G]$. Notice that, contrary to isomorphisms, morphisms do not require bijectivity nor label or edge-preserving properties.

Definition 4. *A graph grammar with neighborhood-controlled embedding (NCE graph grammar) $GG = (\Sigma, \Delta \subseteq \Sigma, S \in \Sigma, P)$ consists of a finite set of symbols Σ that is called alphabet, a subset of the alphabet $\Delta \subseteq \Sigma$ that holds the terminal symbols (we define the complementary set of non-terminal symbols as $\Gamma := \Sigma \setminus \Delta$), a special symbol of the alphabet $S \in \Sigma$, that is called start symbol, and a finite set of production rules P of the form $(A \to R, \omega)$ where $A \in \Gamma$ is called left-hand side, $R \in \mathcal{G}_\Sigma$ is called right-hand side and $\omega : V_R \nrightarrow 2^{\Sigma \times \Sigma}$ is the partial embedding function from R's vertices to pairs of edge and vertex labels.*

Notice that, in the original definition of NCE graph grammars [11], the left-hand sides of the productions were allowed to contain any connected graph. Moreover, only undirected graphs without edge labels were allowed. So, strictly speaking, the definition above characterizes actually a 1-edNCE graph grammar, that contains only one element in the left-hand side and a directed edge-labeled graph in the right-hand side. Nevertheless, for simplicity, we use the denomination NCE graph grammar, or simply graph grammar, to refer to a 1-edNCE graph grammar along this paper. Moreover, vertices from the right-hand sides of rules labeled by non-terminal (terminal) symbols are said to be non-terminal (terminal) vertices. And finally, we define, for convenience, the start graph of GG as $Z_{GG} := (\{v_s\}, \emptyset, \{v_s \mapsto S\})$.

In the following, we present our concrete syntax inspired by the well-known Backus-Naur form to denote NCE graph grammar rules. Let $GG = (\{A, B, a, b, c, l, m\}, \{a, b, c, l, m\}, A, \{p, q\})$ be a graph grammar with production rules $p = (A \to G, \omega)$ and $q = (A \to H, \zeta)$ where $G = (\{v_1, v_2, v_3\}, \{(v_1, l, v_2), (v_2, m, v_3)\}, \{v_1 \mapsto B, v_2 \mapsto b, v_3 \mapsto c\})$, and $H = (\{u_1\}, \emptyset, \{u_1 \mapsto a\})$, we denote p and q together as



Observe that, we use squares for non-terminal vertices, circles for terminal vertices, position the respective label inside the shape and the (possibly omitted) identifier over it. Near each edge is positioned its respective label. The embedding function is not included in the notation, so it is expressed separately, if necessary.

In the sequel, we introduce the dynamic aspects of NCE graph grammars by means of the concepts of derivation step, derivation, and language.

Definition 5. *Let $GG = (\Sigma, \Delta, S, P)$ be a NCE graph grammar and G and H be two graphs over Σ that are disjoint to all right-hand sides from P, G concretely derives in one step into H with rule r and vertex v, we write $G \overset{r,v}{\Rightarrow}_{GG} H$ and*

call it a concrete derivation step, if, and only if, the following holds:

$$
\begin{aligned}
r = (A \to R, \omega) \in P \text{ and } A = \phi_G(v) \text{ and}\\
V_H = (V_G \setminus \{v\}) \cup V_R \text{ and}\\
E_H = (E_G \setminus (\{(v,l,w) \mid (v,l,w) \in E_G\} \cup \{(w,l,v) \mid (w,l,v) \in E_G\}))\\
\quad \cup E_R\\
\quad \cup \{(w,l,t) \mid (w,l,v) \in E_G \wedge (l, \phi_G(w)) \in \omega(t)\}\\
\quad \cup \{(t,l,w) \mid (v,l,w) \in E_G \wedge (l, \phi_G(w)) \in \omega(t)\} \text{ and}\\
\phi_H = (\phi_G \setminus \{(v,x) \mid x \in \Sigma\}) \cup \phi_R
\end{aligned}
$$

Without loss of generality, we set $\omega(t) = \emptyset$ for all vertices t without an image defined in ω. Furthermore, let H' be isomorphic to H, if G *concretely derives* in one step into H, we say G *derives* in one step into H' and write $G \overset{r,v}{\Rightarrow}_{GG} H'$.

When GG, r or v are clear in the context or irrelevant we might omit them and simply write $G \Rrightarrow H$ or $G \Rightarrow H$. Moreover, we denote the reflexive transitive closure of $\Rightarrow$ by $\Rightarrow^*$ and, for $G \Rightarrow^* H'$, we say G derives into H'.

A concrete derivation can be informally understood as the replacement of a non-terminal vertex v and all its adjacent edges in G by a graph R plus edges e between former neighbors w of v and some vertices t of R, provided e's label and w's label are in the embedding specification $\omega(t)$. That is, the embedding function ω of a rule specifies which neighbors of v are to be connected with which vertices of R, according to their labels and the adjacent edges' labels. The process that governs the creation of these edges is called embedding and can occur in various forms in different graph grammar formalisms. We opted for a rather simple approach, in which the edges' directions and labels are maintained.

Definition 6. *A derivation D in the grammar GG is a non-empty sequence of derivation steps and is written as*

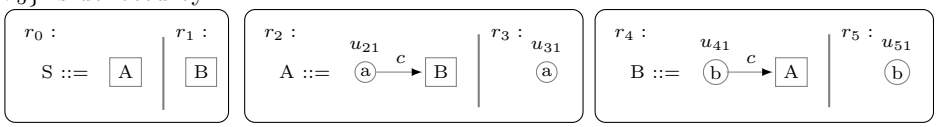
$$D = (G_0 \overset{r_0,v_0}{\Rightarrow} G_1 \overset{r_1,v_1}{\Rightarrow} G_2 \overset{r_2,v_2}{\Rightarrow} \dots \overset{r_{n-1},v_{n-1}}{\Rightarrow} G_n)$$

Definition 7. *The language $L(GG)$ generated by the grammar GG is the set of all graphs containing only terminal vertices derived from the start graph Z_{GG}, that is*

$$L(GG) = \{H \text{ is a graph over } \Delta \text{ and } Z_{GG} \Rightarrow^* H\}$$

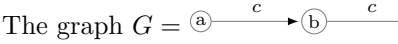
Below, we give one example of a grammar whose language consists of all chains of one or more vertices with interleaved vertices labeled with a and b.

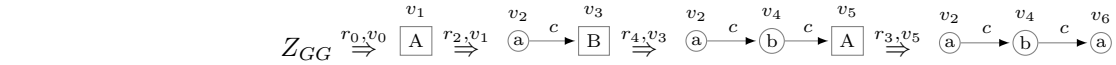
Example 1. $GG = (\{S, A, B, a, b, c\}, \{a, b, c\}, S, P)$, where $P = \{r_0, r_1, r_2, r_3, r_4, r_5\}$ is denoted by



with $\omega_0 = \omega_1 = \emptyset$, $\omega_2(u_{21}) = \omega_3(u_{31}) = \{(c, b)\}$ and $\omega_4(u_{41}) = \omega_5(u_{51}) =$

$\{(c, a)\}$ being the complete definition of the embedding functions of the rules, $r_0, r_1, r_2, r_3, r_4, r_5$ respectively.

The graph $G =$  belongs to $L(GG)$ because it contains only terminal vertices and Z_{GG} derives into it using the following derivation:



Building upon the concepts of graphs and graph grammars, we present, in the following, our understanding over triple graphs and triple graph grammars, supported by the TGG specification from [16].

Definition 8. *A directed labeled triple graph $TG = G_s \overset{m_s}{\leftarrow} G_c \overset{m_t}{\rightarrow} G_t$ over Σ consists of three disjoint directed labeled graphs over Σ (see Definition 1), respectively, the source graph G_s, the correspondence graph G_c and the target graph G_t, together with two injective morphisms (see Definition 2) $m_s : V_{G_c} \to V_{G_s}$ and $m_t : V_{G_c} \to G_{G_t}$.*

We refer to directed labeled triple graphs are often simply as triple graphs in this paper and we might omit the morphisms' names in the notation. Moreover, we define the special empty triple graph as $\varepsilon := E \overset{m_s}{\leftarrow} E \overset{m_t}{\rightarrow} E$ with $E = (\emptyset, \emptyset, \emptyset)$ and $m_s = m_t = \emptyset$ and we denote the set of all triple graphs over Σ by $\mathcal{TG}_\Sigma$. We also point out that in the literature, triple graphs are often modeled as typed graphs, but we judge that, for our circumstance, labeled graphs fit better.

Below, we introduce the standard definition of TGG.

Definition 9. *A triple graph grammar $TGG = (\Sigma, \Delta \subseteq \Sigma, S \in \Sigma, P)$ consists of, analogously to graph grammars (see Definition 4), an alphabet Σ, a set of terminal symbols Δ, a start symbol S and a set of production rules P of the form $L \to R$ with $L = L_s \leftarrow L_c \to L_t$ and $R = R_s \leftarrow R_c \to R_t$ and $L \subseteq R$.*

As the reader should notice, this definition of TGG does not fit our needs optimally, because it defines a context-sensitive graph grammar, whereas we wish a context-free graph grammar to use together with the NCE graph grammar formalism. Hence, we refine it, in the next section, to create a NCE TGG, that fits our context better.

4 NCE TGG: A TGG with Non-terminal Symbols

In this section, we put forward our first contribution, that is the result of mixing the NCE and the TGG grammars.

Definition 10. *A triple graph grammar with neighborhood-controlled embedding (NCE TGG) $TGG = (\Sigma, \Delta \subseteq \Sigma, S \in \Sigma, P)$ consists of an alphabet Σ, a set of terminal symbols Δ (also define $\Gamma := \Sigma \setminus \Delta$), a start symbol S and a set of production rules P of the form $(A \to (R_s \leftarrow R_c \to R_t), \omega_s, \omega_t)$ with $A \in \Gamma$ being the left-hand side, $(R_s \leftarrow R_c \to R_t) \in \mathcal{TG}_\Sigma$ the right-hand side and $\omega_s : V_{R_s} \nrightarrow 2^{\Sigma \times \Sigma}$ and $\omega_t : V_{R_t} \nrightarrow 2^{\Sigma \times \Sigma}$ the partial embedding functions from the right-hand side's vertices to pairs of edge and vertex labels.*

For convenience, define also the start triple graph of TGG as $Z_{TGG} := Z_s \overset{ms}{\leftarrow} Z_c \overset{mt}{\rightarrow} Z_t$ where $Z_s = (\{s_0\}, \emptyset, \{s_0 \mapsto S\})$, $Z_c = (\{c_0\}, \emptyset, \{c_0 \mapsto S\})$, $Z_t = (\{t_0\}, \emptyset, \{t_0 \mapsto S\})$, $ms = \{c_0 \mapsto s_0\}$ and $mt = \{c_0 \mapsto t_0\}$.

The most important difference between the traditional TGG and the NCE TGG is that the former allows any triple graph to occur in the left-hand sides, whereas the latter only one symbol. In addition to that, traditional TGG requires that the whole left-hand side occur also in the right-hand side, that is to say, the rules are monotonic. Therewith, embedding is not an issue, because an occurrence of the left-hand side is not effectively replaced by the right-hand side, instead, only new vertices are added. On the other hand, NCE TGG has to deal with embedding through the embedding function.

In the following, the semantics for NCE TGG is presented analogously to the semantics for NCE graph grammars.

Definition 11. *Let $TGG = (\Sigma, \Delta, S, P)$ be a NCE TGG and $G = G_s \overset{g_s}{\leftarrow} G_c \overset{g_t}{\rightarrow} G_t$ and $H = H_s \overset{h_s}{\leftarrow} H_c \overset{h_t}{\rightarrow} H_t$ be two triple graphs over Σ that are disjoint to all right-hand sides from P, G concretely derives in one step into H with rule r and distinct vertices v_s, v_c, v_t, we write $G \overset{r,v_s,v_c,v_t}{\Rightarrow}_{TGG} H$ if, and only if, the following holds:*

$$
\begin{aligned}
r &= (A \to (R_s \overset{r_s}{\leftarrow} R_c \overset{r_t}{\rightarrow} R_t), \omega_s, \omega_t) \in P \text{ and}\\
A &= \phi_{G_s}(v_s) = \phi_{G_c}(v_c) = \phi_{G_t}(v_t) \text{ and}\\
V_{H_s} &= (V_{G_s} \setminus \{v_s\}) \cup V_{R_s} \text{ and}\\
V_{H_c} &= (V_{G_c} \setminus \{v_c\}) \cup V_{R_c} \text{ and}\\
V_{H_t} &= (V_{G_t} \setminus \{v_t\}) \cup V_{R_t} \text{ and}\\
E_{H_s} &= (E_{G_s} \setminus (\{(v_s, l, w) \mid (v_s, l, w) \in E_{G_s}\} \cup \{(w, l, v_s) \mid (w, l, v_s) \in E_{G_s}\}))\\
&\quad \cup E_{R_s}\\
&\quad \cup \{(w, l, t) \mid (w, l, v_s) \in E_{G_s} \wedge (l, \phi_{G_s}(w)) \in \omega_s(t)\}\\
&\quad \cup \{(t, l, w) \mid (v_s, l, w) \in E_{G_s} \wedge (l, \phi_{G_s}(w)) \in \omega_s(t)\} \text{ and}\\
E_{H_c} &= (E_{G_c} \setminus (\{(v_c, l, w) \mid (v_c, l, w) \in E_{G_c}\} \cup \{(w, l, v_c) \mid (w, l, v_c) \in E_{G_c}\}))\\
&\quad \cup E_{R_c} \text{ and}\\
E_{H_t} &= (E_{G_t} \setminus (\{(v_t, l, w) \mid (v_t, l, w) \in E_{G_t}\} \cup \{(w, l, v_t) \mid (w, l, v_t) \in E_{G_t}\}))\\
&\quad \cup E_{R_t}\\
&\quad \cup \{(w, l, t) \mid (w, l, v_t) \in E_{G_t} \wedge (l, \phi_{G_t}(w)) \in \omega_t(t)\}\\
&\quad \cup \{(t, l, w) \mid (v_t, l, w) \in E_{G_t} \wedge (l, \phi_{G_t}(w)) \in \omega_t(t)\} \text{ and}\\
h_s &= (g_s \setminus \{(v_c, x) \mid x \in V_{G_s}\}) \cup r_s\\
h_t &= (g_t \setminus \{(v_c, x) \mid x \in V_{G_t}\}) \cup r_t\\
\phi_{H_s} &= (\phi_{G_s} \setminus \{(v_s, x) \mid x \in \Sigma\}) \cup \phi_{R_s} \text{ and}\\
\phi_{H_c} &= (\phi_{G_c} \setminus \{(v_c, x) \mid x \in \Sigma\}) \cup \phi_{R_c} \text{ and}\\
\phi_{H_t} &= (\phi_{G_t} \setminus \{(v_t, x) \mid x \in \Sigma\}) \cup \phi_{R_t}
\end{aligned}
$$

Without loss of generality, we set $\omega(t) = \emptyset$ for all vertices t without an image defined in ω. And, analogously to graph grammars, if $G \overset{r,v_s,v_c,v_t}{\Rightarrow}_{TGG} H$ and $H' \in [H]$, then $G \overset{r,v_s,v_c,v_t}{\Rightarrow}_{TGG} H'$, moreover the reflexive transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$ and we call these relations by the same names as before, namely, derivation in one step and derivation. We might also omit identifiers.

A concrete derivation of a triple graph $G = G_s \overset{g_s}{\leftarrow} G_c \overset{g_t}{\rightarrow} G_t$ can be informally understood as concrete derivations (see Definition 5) of G_s, G_c and G_t according to the right-hand sides R_s, R_c and R_t. The only remark is the absence of an embedding mechanism for the correspondence graph, whose edges are not important for our approach. Nevertheless, the addition of such a mechanism for the correspondence graph should not be a problem if it is desired.

Definition 12. *A derivation D in the triple graph grammar TGG is a non-empty sequence of derivation steps*

$$D = (G_0 \overset{r_0,s_0,c_0,t_0}{\Rightarrow} G_1 \overset{r_1,s_1,c_1,t_1}{\Rightarrow} G_2 \overset{r_2,s_2,c_2,t_2}{\Rightarrow} \dots \overset{r_{n-1},s_{n-1},c_{n-1},t_{n-1}}{\Rightarrow} G_n)$$

Definition 13. *The language $L(TGG)$ generated by the triple grammar TGG is the set of all triple graphs containing only terminal vertices derived from the start triple graph Z_{TGG}, that is*

$$L(TGG) = \{H \text{ is a triple graph over } \Delta \text{ and } Z_{TGG} \Rightarrow^* H\}$$

Our concrete syntax for NCE TGG is similar to the one for NCE graph grammars and is presented below by means of the Example 2. The only difference is at the right-hand sides, that include the morphisms between the correspondence graphs and source and target graphs depicted with dashed lines.

Example 2. This example illustrates the definition of a NCE TGG that characterizes the language of all $Pseudocode$ graphs together with their respective $Controlflow$ graphs. A $Pseudocode$ graph is an abstract representation of a program written in a pseudo-code where vertices refer to *actions*, *ifs* or *whiles* and edges connect these items together according to how they appear in the program. A $Controlflow$ graph is a more abstract representation of a program, where vertices can only be either a *command* or a *branch*.

Consider, for instance, the program *main*, written in a pseudo-code, and the triple graph TG in Figure 1. The triple graph TG consists of the $Pseudocode$ graph of *main* connected to the $Controlflow$ graph of the same program through the correspondence graph in the middle of them. In such graph, the vertex labels of the $Pseudocode$ graph p, i, a, w correspond to the concepts of *program*, *if*, *action* and *while*, respectively. The edge label f is given to the edge from the vertex p to the program's first statement, x stands for *next* and indicates that a statement is followed by another statement, p and n stand for *positive* and *negative* and indicate which assignments correspond to the positive of negative case of the *if*'s evaluation, finally l stands for *last* and indicates the last action of a loop. In the $Controlflow$ graph, the vertex labels g, b, c stand for the concepts

of *graph*, *branch* and *command*, respectively. The edge label r is given to the edge from the vertex g to the first program's statement, x, p and n mean, analogous to the former graph, *next*, *positive* and *negative*. In the correspondence graph, the labels pg, ib, ac, wb serve to indicate which labels in the source and target graphs are being connected through the triple graph's morphism.

The main difference between the two graphs is the absence of the w label in the $Controlflow$ graph, what makes it encode loops through the combination of b-labeled vertices and x-labeled edges.

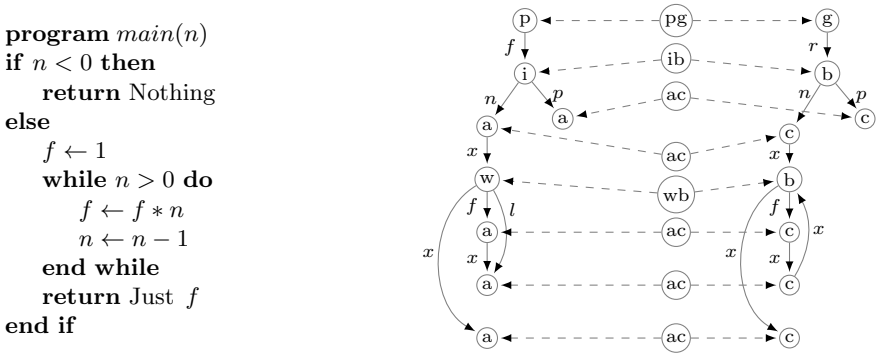
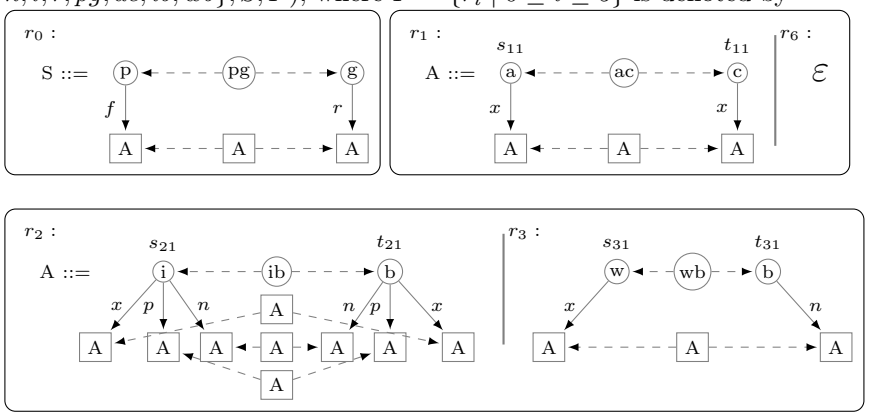
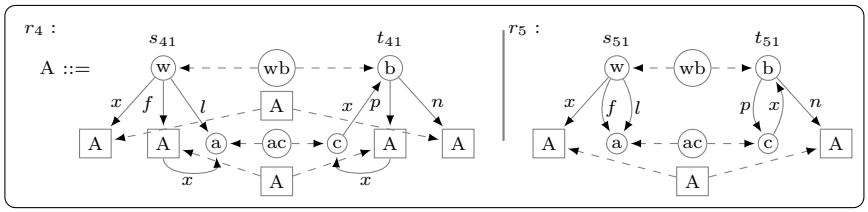


Fig. 1. A program written in pseudo-code on the left and its correspondent triple graph with the $PseudoCode$ and the $ControlFlow$ graphs on the right

The TGG that specifies the relation between these two types of graphs is $TGG = (\{S, A, p, a, i, w, g, b, c, f, x, n, l, r, pg, ac, ib, wb\}, \{p, a, i, w, g, b, c, f, x, n, l, r, pg, ac, ib, wb\}, S, P)$, where $P = \{r_i \mid 0 \leq i \leq 5\}$ is denoted by





with $\sigma_0 = \emptyset$, $\sigma_1(s_{11}) = \sigma_2(s_{21}) = \sigma_3(s_{31}) = \sigma_4(s_{41}) = \sigma_5(s_{51}) = \{(f,p),(x,a),(x,i),(x,w),(p,i),(n,i),(l,w),(f,w)\}$ and $\tau_1(t_{11}) = \tau_2(t_{21}) = \tau_3(t_{31}) = \tau_4(t_{41}) = \tau_5(t_{51}) = \{(r,g),(x,c),(x,b),(p,b),(n,b)\}$ being the complete definition of the source and target embedding functions of the rules r_0 to r_5, respectively.

The rule r_0 relates programs to graphs, r_1 actions to commands, r_2 ifs to branches, r_3 empty whiles to simple branches, r_4 filled whiles to filled loops with branches, r_5 whiles with one action to loops with branches with one command and, finally, r_6 produces an empty graph from a symbol A, what allows any derivation in the grammar to finish.

The aforementioned triple graph TG is in $L(TGG)$, because the derivation $Z_{TGG} \overset{r_0}{\Rightarrow} G_1 \overset{r_2}{\Rightarrow} G_2 \overset{r_6}{\Rightarrow} G_3 \overset{r_1}{\Rightarrow} G_4 \overset{r_6}{\Rightarrow} G_5 \overset{r_1}{\Rightarrow} G_6 \overset{r_4}{\Rightarrow} G_7 \overset{r_1}{\Rightarrow} G_8 \overset{r_6}{\Rightarrow} G_9 \overset{r_1}{\Rightarrow} G_{10} \overset{r_6}{\Rightarrow} TG$ is a derivation in TGG with appropriate G_i for $1 \leq i \leq 10$.

Ultimately, consider the definitions of Γ-boundary graphs and BNCE TGG, that are necessary for the next section.

Definition 14. *A Γ-boundary graph G is such that vertices labeled with any symbol from Γ are not neighbors. That is, the graph G is Γ-boundary if, and only if, there is no $(v,l,w) \in E_G$ with $\phi_G(v) \in \Gamma$ and $\phi_G(w) \in \Gamma$.*

Definition 15. *A boundary triple graph grammar with neighborhood-controlled embedding (BNCE TGG) is such that non-terminal vertices of the right-hand sides of rules are not neighbors. That is, the NCE triple graph grammar TGG is boundary if, and only if, for all its rules' right-hand sides $R_s \leftarrow R_c \rightarrow R_t$, R_s, R_c, and R_t are Γ-boundary graphs.*

5 Model Transformation with NCE TGG

As already introduced, TGG can be used to characterize languages of triple graphs holding correctly transformed models. That is, one can interpret a TGG as the description of the correctly-transformed relation between two sets of models $\mathcal{S}$ and $\mathcal{T}$, where two models $G \in \mathcal{S}$ and $T \in \mathcal{T}$ are in the relation if, and only if, G and T are respectively, source and target graphs of any triple graph of the language $L(TGG)$. That being said, we are interested in this section on defining a model transformation algorithm that interprets a NCE TGG TGG to transform a source model G into one of its correspondent target models T according to the correctly-transformed relation defined by TGG.

For that end, let $TGG = (\Sigma = \Sigma_s \cup \Sigma_t, \Delta, S, P)$ be a triple graph grammar defining the correctly-transformed relation between two arbitrary sets of graphs

$\mathcal{S}$ over Σ_s and $\mathcal{T}$ over Σ_t. And let $G \in \mathcal{S}$ be a source graph. We want to find a target graph $T \in \mathcal{T}$ such that $G \leftarrow C \rightarrow T \in L(TGG)$. To put in words, we wish to find a triple graph holding G and T that is in the language of all correctly transformed models. Hence, the model transformation problem is reduced— according to the definition of triple graph language (see Definition 13)— to the problem of finding a derivation $Z_{TGG} \Rightarrow^*_{TGG} G \leftarrow C \rightarrow T$.

Our strategy to solve this problem is, first, to get a derivation for G with the source part of TGG and, then, construct the derivation $Z_{TGG} \Rightarrow^*_{TGG} G \leftarrow C \rightarrow T$. For this purpose, consider the definition of the s function, that extracts the source part of a production rule.

Definition 16. *Let $r = (A \rightarrow (G_s \leftarrow G_c \rightarrow G_t), \omega_s, \omega_t)$ be a production rule of a triple graph grammar, $\mathrm{s}(r) = (A \rightarrow G_s, \omega_s)$ gives the source part of r. Moreover, $\mathrm{s}^{-1}((A \rightarrow G_s, \omega_s)) = r$ gives the original rule of a source rule.*

In order for s^{-1} to be well defined, we require that all source parts $(A \rightarrow G_s, \omega_s)$ be unique. This does not affect the generality of the formalism, for right-hand side graphs G_s are still allowed to be isomorphic.

Definition 17. *Let $TGG = (\Sigma, \Delta, S, P)$ be a triple graph grammar, $\mathrm{S}(TGG) = (\Sigma, \Delta, S, \mathrm{s}(P))$ gives the source grammar of TGG.*

Furthermore, consider the definition of the non-terminal consistent (NTC) property for TGG, which assures, that non-terminal vertices of the correspondent graphs are connected to vertices with the same label in the source and target graphs.

Definition 18. *A triple graph grammar $TGG = (\Sigma, \Delta, S, P)$ is non-terminal consistent (NTC) if and only if, for all rules $(A \rightarrow (G_s \overset{ms}{\leftarrow} G_c \overset{mt}{\rightarrow} G_t), \omega_s, \omega_t) \in P$, the following holds:*

1. *$\forall c \in V_{G_c}$. if $\phi_{G_c}(c) \in \Gamma$ then $\phi_{G_c}(c) = \phi_{G_s}(ms(c)) = \phi_{G_t}(mt(c))$ and*
2. *For the sets $N_s = \{v \mid \phi_{G_s}(v) \in \Gamma\}$ and $N_t = \{v \mid \phi_{G_t}(v) \in \Gamma\}$, the range-restricted functions $(ms \triangleright N_s)$ and $(mt \triangleright N_t)$ are bijective.*

Finally, the following result gives us an equivalence between a derivation in TGG and a derivation in its source grammar $S(TGG)$, which allows us to construct our goal derivation of $G \leftarrow C \rightarrow T$ in TGG using the derivation of G in $S(TGG)$.

Theorem 1. *Let $TGG = (\Sigma, \Delta, S, P)$ be a NTC TGG and $k \geq 1$, $D = Z_{TGG} \overset{r_0, s_0, c_0, t_0}{\Rightarrow} G^1 \overset{r_1, s_1, c_1, t_1}{\Rightarrow} \dots \overset{r_{k-1}, s_{k-1}, c_{k-1}, t_{k-1}}{\Rightarrow} G^k$ is a derivation in TGG if, and only if, $\overline{D} = Z_{S(TGG)} \overset{s(r_0), s_0}{\Rightarrow} G^1_s \overset{s(r_1), s_1}{\Rightarrow} \dots \overset{s(r_{k-1}), s_{k-1}}{\Rightarrow} G^k_s$ is a derivation in $S(TGG)$.*

Proof. We want to show that if D is a derivation in $TGG = (\Sigma, \Delta, S, P)$, then $\overline{D}$ is a derivation in $SG := S(TGG) = (\Sigma, \Delta, S, SP)$, and vice-versa. We prove it by induction in the following.

First, for the induction base, since, $Z_{TGG} \overset{r_0,s_0,c_0,t_0}{\Rightarrow}_{TGG} G^1$, then expanding Z_{TGG} and G^1, we have

$Z_s \leftarrow Z_c \rightarrow Z_t \overset{r_0,s_0,c_0,t_0}{\Rightarrow}_{TGG} G^1_s \leftarrow G^1_c \rightarrow G^1_t$, then, by Definition 11,
$r_0 = (S \rightarrow (R_s \leftarrow R_c \rightarrow R_t), \omega_s, \omega_t) \in P$ and, by Definition 16,
$\mathrm{s}(r_0) = (S \rightarrow R_s, \omega_s) \in SP$

Hence, using it plus the configuration of $\phi_{Z_s}(s_0)$, $V_{G^1_s}$, $E_{G^1_s}$ and $\phi_{G^1_s}$ and the equality $Z_s = Z_{SG}$, we have, by Definition 5, $Z_{SG} \overset{\mathrm{s}(r_0),s_0}{\Rightarrow}_{SG} G^1_s$.

In the other direction, we choose c_0, t_0 from the definition of Z_{TGG}, with $\phi_{Z_c}(c_0) = S$ and $\phi_{Z_t}(t_0) = S$. In this case, since,

$Z_{SG} \overset{s(r_0),s_0}{\Rightarrow}_{SG} G^1_s$, then by Definition 5,
$\mathrm{s}(r_0) = (S \rightarrow R_s, \omega_s) \in SP$ and, using the bijectivity of s, we get
$r_0 = \mathrm{s}^{-1}(s(r_0)) = (S \rightarrow (R_s \leftarrow R_c \rightarrow R_t), \omega_s, \omega_t) \in P$

Hence, using it plus the configuration of $\phi_{Z_{SG}}(s_0)$, $V_{G^1_s}$, $E_{G^1_s}$ and $\phi_{G^1_s}$, the equality $Z_s = Z_{SG}$ and constructing $V_{G^1_c}$, $V_{G^1_t}$, $E_{G^1_c}$, $E_{G^1_t}$, $\phi_{G^1_c}$, $\phi_{G^1_t}$ from Z_c and Z_t according to the Definition 11 $Z_{TGG} \overset{r_0,s_0,c_0,t_0}{\Rightarrow}_{TGG} G^1_s \leftarrow G^1_c \rightarrow G^1_t$.

Now, for the induction step, we want to show that if $Z_{TGG} \Rightarrow^*_{TGG} G^i \overset{r_i,s_i,c_i,t_i}{\Rightarrow}_{TGG} G^{i+1}$ is a derivation in TGG, then $Z_{SG} \Rightarrow^*_{SG} G^i_s \overset{s(r_i),s_i}{\Rightarrow}_{SG} G^{i+1}_s$ is a derivation in SG and vice-versa, provided that the equivalence holds for the first i steps, so we just have to show it for the step $i+1$.

So, since, $G^i \overset{r_i,s_i,c_i,t_i}{\Rightarrow}_{TGG} G^{i+1}$, that is

$G^i_s \overset{ms_i}{\leftarrow} G^i_c \overset{mt_i}{\rightarrow} G^i_t \overset{r_i,s_i,c_i,t_i}{\Rightarrow}_{TGG} G^{i+1}_s \leftarrow G^{i+1}_c \rightarrow G^{i+1}_t$, then, by Definition 11,
$r_i = (S \rightarrow (R_s \leftarrow R_c \rightarrow R_t), \omega_s, \omega_t) \in P$, and by Definition 16,
$\mathrm{s}(r_i) = (S \rightarrow R_s, \omega_s) \in SP$

Hence, using it plus the configuration of $\phi_{G^i_s}(s_i)$, $V_{G^{i+1}_s}$, $E_{G^{i+1}_s}$ and $\phi_{G^{i+1}_s}$, we have, by Definition 5, $G^i_s \overset{\mathrm{s}(r_i),s_i}{\Rightarrow}_{SG} G^{i+1}_s$.

In the other direction, we choose, using the bijectivity from the range restricted function s, stemming from the NTC property, $c_i = ms_i^{-1}(s_i), t_i = mt_i(c_i)$. Moreover, since TGG is NTC, and because, by induction hypothesis, $Z_{TGG} \Rightarrow^*_{TGG} G^i$ is a derivation in TGG and $\phi_{G^i_s}(s_i) \in \Gamma$, it is clear that $\phi_{G^i_s}(s_i) = \phi_{G^i_c}(c_i) = \phi_{G^i_t}(t_i)$.

In this case, since

$G^i_s \overset{s(r_i),s_i}{\Rightarrow}_{SG} G^{i+1}_s$, then, by Definition 5,
$\mathrm{s}(r_i) = (A \rightarrow R_s, \omega_s) \in SP$ and, using the bijectivity of s, we get
$r_i = \mathrm{s}^{-1}(s(r_i)) = (A \rightarrow (R_s \leftarrow R_c \rightarrow R_t), \omega_s, \omega_t) \in P$

Hence, using, additionally, the configuration of $\phi_{G_s^i}(s_i)$, $\phi_{G_c^i}(c_i)$, $\phi_{G_t^i}(t_i)$, $V_{G_s^{i+1}}$, $E_{G_s^{i+1}}$ and $\phi_{G_s^{i+1}}$ and constructing $V_{G_c^{i+1}}$, $V_{G_t^{i+1}}$, $E_{G_c^{i+1}}$, $E_{G_t^{i+1}}$, $\phi_{G_c^{i+1}}$, $\phi_{G_t^{i+1}}$ from G_c^i and G_t^i according to the Definition 11, we have

$$G_s^i \leftarrow G_c^i \rightarrow G_t^i \overset{r_i, s_i, c_i, t_i}{\Rightarrow}_{TGG} G_s^{i+1} \leftarrow G_c^{i+1} \rightarrow G_t^{i+1}$$

This finishes the proof. □

Therefore, the problem of finding a derivation $D = Z_{TGG} \Rightarrow^* G \leftarrow C \rightarrow T$ in TGG is reduced to finding a derivation $\overline{D} = Z_{S(TGG)} \Rightarrow^* G$ in $S(TGG)$, what can be done with the procedure in [15]. The final construction of the triple graph $G \leftarrow C \rightarrow T$ becomes then just a matter of creating D out of $\overline{D}$.

The complete transformation procedure is presented in the Algorithm 1. Thereby, it is required that the TGG be a BNCE TGG (see Definition 15) and be neighborhood preserving (NP) [15, 17], what poses no problem to our procedure, since any BNCE graph grammar can be transformed into the neighborhood preserving normal form.

Algorithm 1 Transformation Algorithm for NP NTC BNCE TGG

Require: TGG is a valid NP NTC BNCE triple graph grammar
Require: G is a valid graph over Σ
  function $transform(TGG = (\Sigma, \Delta, S, P), G = (V_G, E_G, \phi_G))$: $Graph$
    $SG \leftarrow S(TGG)$ ▷ see Definition 16
    $\overline{D} \leftarrow parse(SG, G)$ ▷ use procedure in [15]
    if $\overline{D} = Z_{SG} \Rightarrow^*_{SG} G$ **then** ▷ if parsed successfully
      From $\overline{D}$, construct $D = Z_{TGG} \Rightarrow^*_{TGG} G \leftarrow C \rightarrow T$ ▷ see Theorem 1
      return Just T
    else
      return Nothing ▷ no T satisfies $(G \leftarrow C \rightarrow T) \in L(TGG)$
    end if
  end function
Ensure: *return* is either Nothing or Just T, such that $(G \leftarrow C \rightarrow T) \in L(TGG)$

6 Evaluation

In order to evaluate the usability of the proposed BNCE TGG formalism, we compare the number of rules and elements (vertices, edges, and mappings) we needed to describe some model transformations in BNCE TGG and in standard TGG without application conditions. Table 1 presents these results.

In the case of $Pseudocode2Controlflow$, our proposed approach shows a clear advantage against the standard TGG formalism. We judge that similarly to what happens to programming languages, this advantage stems from the very nested structure of $Pseudocode$ and $Controlflow$ graphs. That is, for instance,

	Standard TGG		BNCE TGG	
Transformation	Rules	Elements	Rules	Elements
Pseudocode2Controlflow	45	1061	**7**	**185**
BTree2XBTree	**4**	**50**	5	80
Star2Wheel	-	-	**6**	**89**
Class2Database	**6**	**98**	-	-

Table 1. Results of the usability evaluation of the BNCE TGG formalism in comparison with the standard TGG for the model transformation problem

in rule the r_2 of this TGG (see Example 2), a node in a positive branch of an if-labeled vertex is never connected with a node in the negative branch. This disjunctive aspect allows every branch to be defined in the rule (as well as effectively parsed) independently of the other branch. This characteristic makes it possible for BNCE TGG rules to be defined in a very straightforward manner and reduces the total number of elements necessary.

In addition to that, the use of non-terminal symbols gives BNCE TGG the power to represent abstract concepts very easily. For example, whereas the rule r_1 encodes, using only few elements, that after each *action* comes any statement A, which can be another *action*, an if, a *while* or nothing (an empty graph), in the standard TGG without application condition or any special inheritance treatment, we need to write a different rule for each of these cases. For the whole grammar, we need to consider all combinations of *actions*, ifs and *whiles* in all rules, what causes the great number of rules and elements.

The $Star2Wheel$ transformation consists of transforming star graphs, which are complete bipartite graphs $K_{1,k}$— where the partitions are named center and border— to wheel graphs, that can be constructed from star graphs by adding edges between border vertices to form a minimal cycle. We could not describe this transformation in standard TGG, especially because of the rules' monotonicity (see Definition 9). That is, we missed the possibility to erase edges in a rule, feature that we do have in the semantics of BNCE TGG through the embedding mechanism.

The $Class2Database$ transformation consists of transforming class diagrams, similar to UML class diagrams, to database diagrams, similar to physical entity-relationship diagrams. We could not describe this transformation in BNCE TGG by the fact that the information about the production of a terminal vertex is owned exclusively by one derivation step. That is, this information cannot be used by other derivation steps (the BNCE grammar is context-free). Thus, in the case of $Class2Database$, in which an *association* is connected to two *classes*, each been produced by two different derivation steps, we could not connect one association with two classes.

7 Conclusion

We present in this paper a new triple graph grammar formalism, called NCE TGG, that is the result of mixing NCE graph grammars [11] with TGG [16] and that introduces for the first time, as far as we know, non-terminal symbols to TGG. Furthermore, we demonstrate how BNCE TGG can be used in the practice to solve the model transformation problem.

An experimental evaluation in Section 6 assesses the usability of BNCE TGG in comparison with standard TGG and reveals that our proposed approach has potential. In special, we could express one transformation with BNCE TGG that we could not with TGG. And, from the other three evaluated transformations, BNCE TGG outperformed TGG in one use case with a much smaller grammar. In our view, smaller and less complex rules tend to be easier to comprehend and validate, what in turn makes formal specifications in BNCE TGG more suitable for safety-critical systems.

We are aware that this disadvantage for TGG comes from the absence of (negative and positive) application conditions, but we also argue that such mechanisms are often unhandy for tools and researchers that want to reason about it. In this sense, the use of non-terminal symbols seems to be a neater alternative to it.

As a future work, we intend to carry out a broader usability and performance evaluation of our approach and extend NCE graph grammars with an application condition mechanism that should allow it to express more languages than it can now. Finally, although the extension of our approach for the bidirectional transformation problem is straightforward, the same does not seem to be true for the model synchronization problem. Whereas the former consists of simply performing transformation from source to target and from target to source, the latter consists of transforming already generated models in both directions without creating them from scratch and using only the information of the modifications. Therefore, we are also interested in studying how our approach can be used to solve it.

References

1. Anjorin, A., Leblebici, E., Schürr, A.: 20 years of triple graph grammars: A roadmap for future research. Electronic Communications of the EASST **73** (2016). https://doi.org/10.14279/tuj.eceasst.73.1031
2. Anjorin, A., Saller, K., Lochau, M., Schürr, A.: Modularizing triple graph grammars using rule refinement. In: International Conference on Fundamental Approaches to Software Engineering. pp. 340–354. Springer (2014). https://doi.org/10.1007/978-3-642-54804-8_24
3. Bardohl, R., Ehrig, H., De Lara, J., Taentzer, G.: Integrating meta-modelling aspects with graph transformation for efficient visual language definition and model manipulation. In: International Conference on Fundamental Approaches to Software Engineering. pp. 214–228. Springer (2004). https://doi.org/10.1007/978-3-540-24721-0_16

4. Brandenburg, F.J., Skodinis, K.: Finite graph automata for linear and boundary graph languages. Theoretical Computer Science **332**(1-3), 199–232 (2005). https://doi.org/10.1016/j.tcs.2004.09.040
5. Ehrig, H., Rozenberg, G., Kreowski, H.J., Montanari, U.: Handbook of graph grammars and computing by graph transformation, vol. 3. World Scientific (1999). https://doi.org/10.1142/3303
6. Flasiński, M.: Power properties of NLC graph grammars with a polynomial membership problem. Theoretical Computer Science **201**(1-2), 189–231 (1998). https://doi.org/10.1016/S0304-3975(97)00212-0
7. Flasiński, M.: On the parsing of deterministic graph languages for syntactic pattern recognition. Pattern Recognition **26**(1), 1–16 (1993). https://doi.org/10.1016/0031-3203(93)90083-9
8. Flasiński, M., Flasińska, Z.: Characteristics of bottom-up parsable edNLC graph languages for syntactic pattern recognition. In: International Conference on Computer Vision and Graphics. pp. 195–202. Springer (2014). https://doi.org/10.1007/978-3-319-11331-9_24
9. Hermann, F., Ehrig, H., Taentzer, G.: A typed attributed graph grammar with inheritance for the abstract syntax of UML class and sequence diagrams. Electronic Notes in Theoretical Computer Science **211**, 261–269 (2008). https://doi.org/10.1016/j.entcs.2008.04.048
10. Hoffmann, B.: Graph transformation with variables. In: Formal Methods in Software and Systems Modeling, pp. 101–115. Springer (2005). https://doi.org/10.1007/978-3-540-31847-7_6
11. Janssens, D., Rozenberg, G.: Graph grammars with neighbourhood-controlled embedding. Theoretical Computer Science **21**(1), 55–74 (1982). https://doi.org/10.1016/0304-3975(82)90088-3
12. Kim, C.: Efficient recognition algorithms for boundary and linear eNCE graph languages. Acta informatica **37**(9), 619–632 (2001). https://doi.org/10.1007/PL00013320
13. Kim, C.: On the structure of linear apex NLC graph grammars. Theoretical Computer Science **438**, 28–33 (2012). https://doi.org/10.1016/j.tcs.2012.02.038
14. Klar, F., Lauder, M., Königs, A., Schürr, A.: Extended triple graph grammars with efficient and compatible graph translators. In: Graph transformations and model-driven engineering, pp. 141–174. Springer (2010). https://doi.org/10.1007/978-3-642-17322-6_8
15. Rozenberg, G., Welzl, E.: Boundary NLC graph grammarsbasic definitions, normal forms, and complexity. Information and Control **69**(1-3), 136–167 (1986). https://doi.org/10.1016/S0019-9958(86)80045-6
16. Schürr, A.: Specification of graph translators with triple graph grammars. In: International Workshop on Graph-Theoretic Concepts in Computer Science. pp. 151–163. Springer (1994). https://doi.org/10.1007/3-540-59071-4_45
17. Skodinis, K., Wanke, E.: Neighborhood-preserving node replacements. In: International Workshop on Theory and Application of Graph Transformations. pp. 45–58. Springer (1998). https://doi.org/10.1007/978-3-540-46464-8_4
18. Wanke, E.: Algorithms for graph problems on BNLC structured graphs. Information and Computation **94**(1), 93–122 (1991). https://doi.org/10.1016/0890-5401(91)90035-Z