Model Transformation with Triple Graph Grammars and Non-terminal Symbols

William da Silva[1,2] and Max Bureck[1]

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

Abstract. **Keywords:** First keyword · Second keyword · Another keyword.

1 Introduction

Quality of service is a very common requirement for software and engineering projects, specially for safety-critical systems. A technique that aims to assure and enhance quality of software is the model-based engineering (MDE) 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 easier discussions about the system.

The construction of a system with MDE commonly requires the creation of various models in different levels of abstraction, in which case we are interested in generating models automatically from other models. This problem is known as model transformation and several approaches to solve it have been proposed so far. Several approaches 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 [4], which consists of building context-sensitive-like grammars of triple graphs, that are composed of two graphs, the source and the target graphs (representing two models), connected by morphisms from a third graph, the correspondence graph.

Despite the various positive aspects of TGGs, 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-like 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-like graph grammar formalism, called BNCE graph grammar from [2], with the stan-

dard TGG formalism from [4], to create the BNCE TGG and (2) showing an argumentation of how it can be used to solve the model transformation problem.

The remainder of this paper is as follows, in Section 2, we present the research publications related to the topic, in Section 3, we give the main definitions necessary to build our approach, in Section 4, we finally propose the our modified version of TGG, the BNCE TGG, in Section 5 we argument 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

3 Graph Grammars and Triple Graph Grammars

In this section, we introduce important definitions that are used throughout this paper. First, we present the definitions, taken mainly from [3], regarding graphs, second, we introduce the families of graph grammars NCE and BNCE taken from [2] and then, we express our understanding of TGG, backed by [4].

Definition 1. *A directed labeled graph G over the 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 $\phi : V \to \Sigma$.*

Directed labeled graphs are often referred to simply as graphs. For a fixed graph G we refer to its components as V_G, E_G and ϕ_G. Moreover, we define the special empty graph as $\varepsilon := (\emptyset, \emptyset, \emptyset)$ and we denote the set of all graphs over Σ by $\mathcal{G}_\Sigma$. Two graphs G and H are disjoint iff $V_G \cap V_H = \emptyset$. 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.

Definition 2. *Two vertices v and w are neighbors (also adjacent) iff there is one or more edges between them, that is, $(v, l, w) \in E_G \vee (w, l, v) \in E_G$. And function $\text{neigh}_G : 2^{V_G} \to 2^{V_G}$, applied to U gives the set of neighbors of the vertices in U minus U. That is $\text{neigh}_G(U) = \{v \in V_G \setminus U \mid$ exists a $(v, l, u) \in E_G$ or a $(u, l, v) \in E_G$ with $u \in U\}$*

Definition 3. *A morphism of graphs G and H is a total mapping $m : V_G \to V_H$.*

Definition 4. *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 $\phi_G(v) = \phi_H(m(v))$. In this case, G and H are said to be isomorphic, we write $G \cong H$, 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.

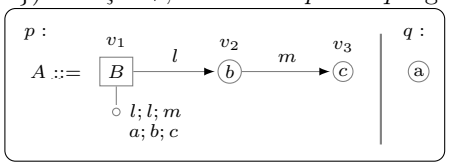
Definition 5. *A Γ-boundary graph G is such that vertices labeled with any symbol from Γ are not neighbors. That is, the graph G is Γ-boundary iff, $\nexists(v, l, w) \in E_G.\ \phi_G(v) \in \Gamma \wedge \phi_G(w) \in \Gamma$.*

Definition 6. *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 the 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 the start symbol, and a finite set of production rules P of the form $(A \to R, \omega)$ where $A \in \Gamma$ is the so-called left-hand side, $R \in \mathcal{G}_\Sigma$ is the right-hand side and $\omega : V_R \nrightarrow 2^{\Sigma \times \Sigma}$ is the partial embedding function from the R's vertices to pairs of edge and label symbols.*

NCE graph grammars are often referred to as graph grammars or simply as grammars in this paper. For convenience, define the start graph of GG as $Z_{GG} := (\{v_s\}, \emptyset, \{v_s \mapsto S\})$. Vertices from the right-hand sides of rules labeled by non-terminal (terminal) symbols are said to be non-terminal (terminal) vertices.

Definition 7. *A boundary graph grammar with neighborhood-controlled embedding (BNCE graph grammar) GG is such that non-terminal vertices of the right-hand sides of rules are not neighbors. That is, the graph grammar GG is boundary iff all its rules' right-hand sides are Γ-boundary graphs.*

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



Notice 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. Over each edge is positioned its respective label. To depict the embedding function, we place near the respective vertex a small circle labeled with the image pairs of the embedding function for this node aligned vertically and separated by semi-colons, which in certain circumstances may also be omitted.

With these syntactic notions of the formalism presented, we introduce below its semantics by means of the concepts of derivation step, derivation and language.

Definition 8. *Let $GG = (\Sigma, \Delta, S, P)$ be a 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 $G \overset{r,v}{\Rrightarrow}_{GG} H$ and call it a concrete derivation step, if, and only if, the following holds:

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

Notice that, without loss of generalization, 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 it *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 in one or more steps into H', or simply G derives into H'.

Definition 9. *A derivation D in GG is a sequence of derivation steps and is written as*

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

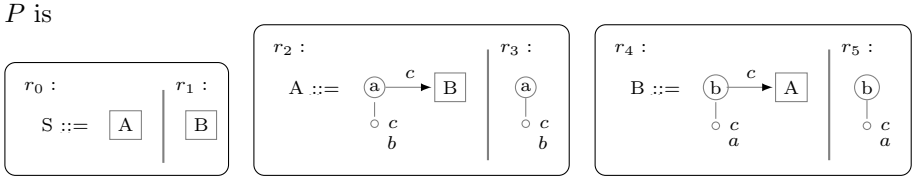
Definition 10. *The language $L(GG)$ generated by the grammar GG is the set of all graphs containing only terminal vertices derived from the start graph Z_{GG}, that is*

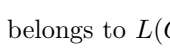
$$L(GG) = \{H \mid \phi_H(V_H) \subseteq \Delta \wedge Z_{GG} \Rightarrow^* H\}$$

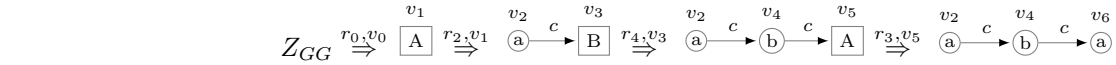
Notice that for every graph $G \in L(GG)$, there is at least one finite derivation $(Z_{GG} \overset{r_0,v_0}{\Rightarrow} \dots \overset{r_{n-1},v_{n-1}}{\Rightarrow} G)$, but it is not guaranteed that this derivation be unique. In the case that there are more than one derivation for a G, we say that the grammar GG is ambiguous.

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. Chains of a's and b's. $GG = (\{S, A, B, a, b, c\}, \{a, b, c\}, S, P)$, where P is



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 (TGGs), supported by the TGG specification from [4].

Definition 11. *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 1), respectively, the source graph G_s, the correspondence graph G_c and the target graph G_t, together with two injective morphisms (see 3) $m_s : V_{G_c} \rightarrow V_{G_s}$ and $m_t : V_{G_c} \rightarrow G_{G_t}$.*

Directed labeled triple graphs are often referred to simply as triple graphs in this paper and we might omit the morphisms' names in the notation. Moreover, we denote the set of all triple graphs over Σ as $\mathcal{TG}_\Sigma$. We might refer to all vertices of TG by $V_{TG} := V_s \cup V_c \cup V_t$, all edges by $E_{TG} := E_s \cup E_c \cup E_t$ and the complete labeling function by $\phi_{TG} := \phi_{G_s} \cup \phi_{G_c} \cup \phi_{G_t}$.

Definition 12. *A Γ-boundary triple graph $TG = G_s \leftarrow G_c \rightarrow G_t$ is such that G_s, G_c and G_t are Γ-boundary graphs.*

Below we introduce the standard definition of TGG. As the reader should notice, this definition of TGG does not fit our needs optimally, because it defines a context-sensitive-like graph grammar whilst we wish a context-free-like graph grammar to use together with the NCE graph grammar formalism. Hence, after presenting the conventional TGG definition, we refine it, in the next Section, to create a NCE TGG, that fits our context best.

Definition 13. *A triple graph grammar $TGG = (\Sigma, \Delta \subseteq \Sigma, S \in \Sigma, P)$ consists of, analogously to graph grammars (see 6), 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 $L \rightarrow R$ with $L = L_s \leftarrow L_c \rightarrow L_t$ and $R = R_s \leftarrow R_c \rightarrow R_t$ and $L \subseteq R$.*

4 BNCE 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 14. *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 $(A \rightarrow (R_s \leftarrow R_c \rightarrow R_t), \omega_s, \omega_t)$ with $A \in \Gamma$ being the left-hand side, $(R_s \leftarrow R_c \rightarrow R_t) \in \mathcal{TG}_\Sigma$ 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 label symbols.*

We might refer to the complete embedding function of a rule by $\omega := \omega_s \cup \omega_t$. 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 $mt = \{c_0 \mapsto t_0\}$.

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 triple graph grammar TGG is boundary iff all its rules' right-hand sides are Γ-boundary triple graphs.*

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

Definition 16. *Let $TGG = (\Sigma, \Delta, S, P)$ be a NCE TGG and G and H 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:*

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

Notice that, without loss of generalization, 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 along this paper.

Definition 17. *A derivation D in TGG is a 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 18. *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*

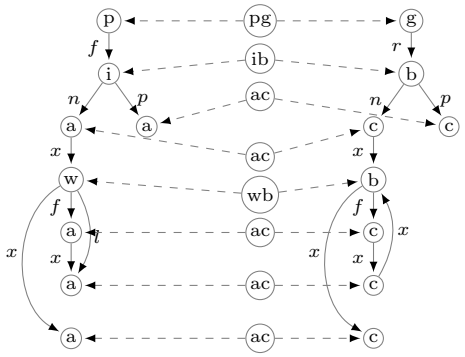
$$L(TGG) = \{H \mid \phi_H(V_H) \subseteq \Delta \wedge Z_{TGG} \Rightarrow^* H\}$$

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 graph and source and target graphs depicted with dashed lines.

Example 2. Pseudocode to Controlflow. This example illustrates the specification of the consistency relation between $Pseudocode$ graphs and $Controlflow$ graphs with BNCE TGG. 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*.

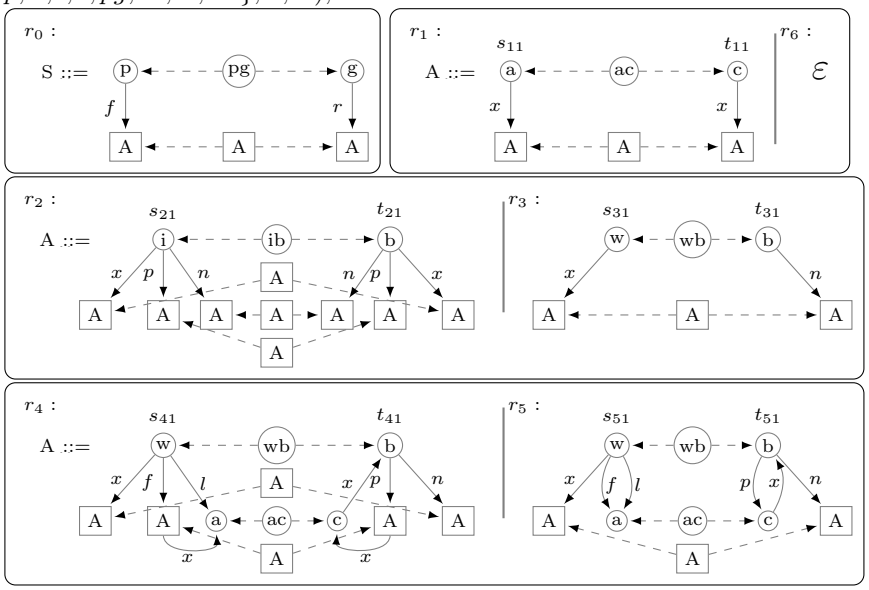
Consider, for instance, the program *main* below on the left, written in a pseudo-code. The triple graph TG on the right consists of the $Pseudocode$ graph of *main* connected to the $Controlflow$ graph of the same program through the correspondence graph in the middle of them. In such graph, the vertex 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, 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.

program $main(n)$
if $n < 0$ **then**
  return *nothing*
else
  $f \leftarrow 1$
  while $n > 0$ **do**
    $f \leftarrow f * n$
    $n \leftarrow n - 1$
  end while
  return $Justf$
end if



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.

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



with $\sigma_1(s_{11}) = \sigma_2(s_{21}) = \sigma_3(s_{31}) = \sigma_4(s_{41}) = \sigma_5(s_{51}) = \{(f, p), (x, a), (x, i), (x, w), (p, i), (n, i), (l, w), (f, w)\}$ 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, 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 $1 \leq i \leq 10$.

5 Model Transformation with BNCE TGG

Given a graph G over Δ, one may be interested in finding another graph T over Δ that is somehow consistent to G, write $G \sim T$. One example of such a situation is the compilation of a source-code, represented abstractly by G, into a machine-code, represented by T. Since, very often, T can be generated from G, following some specification, this problem is referred to as model transformation problem.

Now, let $TGG = (\Sigma, \Delta, S, P)$ be a triple graph grammar such that $G \sim T$ if and only if, $G \leftarrow C \rightarrow T \in L(TGG)$, that is, if we interpret TGG as the descriptor of the consistency relation between G and T, then the model transformation problem is reduced to finding a triple graph $G \leftarrow C \rightarrow T$ that belongs to $L(TGG)$. This is, by the definition of triple graph language (see 18) equivalent to finding a derivation $Z_{TGG} \Rightarrow^*_{TGG} G \leftarrow C \rightarrow T$.

Furthermore, consider the following definitions.

Definition 19. *By extension of the Definition* **??**, *a BNCE triple graph grammar TGG is neighborhood preserving if and only if, $S(TGG)$ and $T(TGG)$ are neighborhood preserving.*

Definition 20. *Let $r = ((A, A, A) \rightarrow (G_s \leftarrow G_c \rightarrow G_t), \omega_s, \omega_t)$ be a production rule of a triple graph grammar, $\mathrm{s}(r) = (A \rightarrow G_s, \omega_s)$ gives the source part of r (symmetrically, $\mathrm{s}^- 1((A \rightarrow G_s, \omega_s)) = r$). Moreover, for any triple graph grammar $TGG = (\Sigma, \Delta, S, P)$, define the source grammar $\mathrm{S}(TGG) = (\Sigma, \Delta, S, SP)$, where $SP = \mathrm{s}(P)$. Analogously, $\mathrm{t}(r) = (C \rightarrow G_t, \omega_t)$, $\mathrm{t}^- 1((C \rightarrow G_t, \omega_t)) = r$ and $\mathrm{T}(TGG) = (\Sigma, \Delta, S, TP)$, where $TP = \mathrm{t}(P)$.*

Definition 21. *A triple graph $G_s \overset{ms}{\leftarrow} G_c \overset{mt}{\rightarrow} G_t$ is Γ-consistent if and only if, $\forall c \in V_{G_c}$. if $\phi_{G_c}(c) \in \Gamma$ then $\phi_{G_c}(c) = \phi_{G_s}(ms(c)) = \phi_{G_t}(mt(c))$ and 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.*

Definition 22. *A triple graph grammar $TGG = (\Sigma, \Delta, S, P)$ is non-terminal consistent (NTC) if and only if, all rules $r \in P$ are Γ-consistent.*

Theorem 1. *Let $TGG = (\Sigma, \Delta, S, P)$ be a NTC TGG, $SG = S(TGG) = (\Sigma, \Delta, S, SP)$ its source grammar and $D = Z_{TGG} \overset{r_0, s_0, c_0, t_0}{\Rightarrow}_{TGG} G^1 \overset{r_1, s_1, c_1, t_1}{\Rightarrow}_{TGG} \ldots \overset{r_{k-1}, s_{k-1}, c_{k-1}, t_{k-1}}{\Rightarrow}_{TGG} G^k$ be a derivation in TGG, D can equivalently be rewritten as the derivation in SG, $\overline{D} = Z_{SG} \overset{s(r_0), s_0}{\Rightarrow}_{SG} G^1_s \overset{s(r_1), s_1}{\Rightarrow}_{SG} \ldots \overset{s(r_{k-1}), s_{k-1}}{\Rightarrow}_{SG} G^k_s$.*

Proof. We want to show that if D is a derivation in TGG, then $\overline{D}$ is a derivation in SG, and vice-versa. We prove it by induction in the following.

First, if $Z_{TGG} \overset{r_0, s_0, c_0, t_0}{\Rightarrow}_{TGG} G^1$, that is $Z_s \leftarrow Z_c \rightarrow Z_t \overset{r_0, s_0, c_0, t_0}{\Rightarrow}_{TGG} G^1_s \leftarrow G^1_c \rightarrow G^1_t$, then, by Definition 16, $r_0 = ((S, S, S) \rightarrow (R_s \leftarrow R_c \rightarrow R_t), \omega_s, \omega_t) \in P$, and by Definition 20, $\mathrm{s}(r_0) = (S \rightarrow R_s, \omega_s) \in SP$, hence, using it, the configuration of $\phi_{Z_s}(s_0)$, $V_{G^1_s}$, $E_{G^1_s}$ and $\phi_{G^1_s}$ and the equality $Z_s = Z_{S(TGG)}$, we have $Z_{S(TGG)} \overset{\mathrm{s}(r_0), s_0}{\Rightarrow}_{SG} G^1_s$.

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, if $Z_{SG} \overset{s(r_0), s_0}{\Rightarrow}_{SG} G^1_s$, then by Definition 8, we have $\mathrm{s}(r_0) = (S \rightarrow R_s, \omega_s) \in SP$ and using the bijectivity of s, we get $r_0 = \mathrm{s}^{-1}(s(r_0)) = ((S, S, S) \rightarrow (R_s \leftarrow R_c \rightarrow R_t), \omega_s, \omega_t) \in P$. Hence, using it, 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 $V_{G_c^1}$, $V_{G_t^1}$, $E_{G_c^1}$, $E_{G_t^1}$, $\phi_{G_c^1}$, $\phi_{G_t^1}$ from Z_c and Z_t according to the Definition 16, we have $Z_{TGG} \overset{r_0,s_0,c_0,t_0}{\Rightarrow}_{TGG} G_s^1 \leftarrow G_c^1 \rightarrow G_t^1$

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 $Z_{TGG} \Rightarrow^*_{SG} G_s^i \overset{s(r_i),s_i}{\Rightarrow}_{SG} G_s^{i+1}$ is a derivation in SG and vice-versa. By induction hypothesis it holds for the first i steps, so we just have to show it for the step $i+1$.

So, if $G^i \overset{r_i,s_i,c_i,t_i}{\Rightarrow}_{TGG} G^{i+1}$, that is $G_s^i \overset{ms_i}{\leftarrow} G_c^i \overset{mt_i}{\rightarrow} G_t^i \overset{r_i,s_i,c_i,t_i}{\Rightarrow}_{TGG} G_s^{i+1} \leftarrow G_c^{i+1} \rightarrow G_t^{i+1}$, then, by Definition 16, $r_i = ((S,S,S) \rightarrow (R_s \leftarrow R_c \rightarrow R_t), \omega_s, \omega_t) \in P$, and by Definition 20, $\mathrm{s}(r_i) = (S \rightarrow R_s, \omega_s) \in SP$, hence, using it and the configuration of $\phi_{G_s^i}(s_i)$, $V_{G_s^{i+1}}$, $E_{G_s^{i+1}}$ and $\phi_{G_s^{i+1}}$, we have $G^i \overset{\mathrm{s}(r_i),s_i}{\Rightarrow}_{SG} G_s^{i+1}$.

In the other direction, we choose, using the bijectivity from the range-restricted 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 $Z_{TGG} \Rightarrow^*_{TGG} G^i$ is a derivation in TGG it is clear that G^i is NTC, thus $\phi_{G_s^i}(s_0) = \phi_{G_c^i}(c_0) = \phi_{G_t^i}(t_i)$.

In this case, if $G_s^i \overset{s(r_i),s_i}{\Rightarrow}_{SG} G_s^{i+1}$, then by Definition 8, we have $\mathrm{s}(r_i) = (A \rightarrow R_s, \omega_s) \in SP$ and using the bijectivity of s, we get $r_i = \mathrm{s}^{-1}(s(r_i)) = ((A,A,A) \rightarrow (R_s \leftarrow R_c \rightarrow R_t), \omega_s, \omega_t) \in P$.

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 16, 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, by Theorem 1, the problem of finding a derivation $D = Z_{TGG} \Rightarrow^*_{TGG} G \leftarrow C \rightarrow T$ is reduced to finding a derivation $\overline{D} = Z_{S(TGG)} \Rightarrow_{S(TGG)} G$, what can be done with the already presented parsing algorithm **??**. 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 neighborhood preserving, what poses no problem to our procedure, once any TGG can be transformed into the neighborhood preserving normal form.

6 Evaluation and Discussion

7 Conclusion

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)
2. Janssens, D., Rozenberg, G.: Graph grammars with neighbourhood-controlled embedding. Theoretical Computer Science **21**(1), 55–74 (1982)

Algorithm 1 Transformation Algorithm for NP NTC BNCE TGGs

Require: TGG is a valid NP NTC BNCE triple graph grammar
Require: G is a valid graph over Σ
  function $transform(TGG = (\Sigma, \Delta, S, P), G = (V_G, E_G, \phi_G))$: $Graph$
    $GG \leftarrow S(TGG)$ ▷ see 20
    $\overline{D} \leftarrow parse(GG, G)$ ▷ use algorithm **??**
    if $\overline{D} = Z_{S(TGG)} \Rightarrow^*{}_{S(TGG)} G$ **then** ▷ if parsed successfully
      from $\overline{D}$ construct $D = Z_{TGG} \Rightarrow^*{}_{TGG} G \leftarrow C \rightarrow T$
      return 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 T, such that $(G \leftarrow C \rightarrow T) \in L(TGG)$

3. Rozenberg, G., Welzl, E.: Boundary nlc graph grammarsbasic definitions, normal forms, and complexity. Information and Control **69**(1-3), 136–167 (1986)
4. 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)