William Bombardelli da Silva

September 26, 2018

Abstract

Contents

0.1 Introduction 2
0.2 Related Work 2
0.3 Theoretical Review 4
0.3.1 Graph Grammars 4
0.3.2 Triple Graph Grammars 8
0.4 Parsing of Graphs with BNCE Graph Grammars 15
0.5 Model Transformation with BNCE Triple Graph Grammars . . . 18
0.6 An Extension of NCE Triple Graph Grammars with Application Conditions 23
0.7 Implementation 26
0.8 Evaluation 26
0.9 Conclusion 28

0.1 Introduction

Overview of the MDE. Potential and problems of it. The challenge of model transformation. A solution: Triple Graph Grammars (TGG) (justification). A problem of TGG (usability/ amount of grammar rules). Our solution for this problem: TGG with non-terminal nodes. Overview of our approach (graph grammars with non-terminal nodes, NCE grammars, graph language, parsing, transformation). Brief intro into graph grammars and embedding. Explain why BNCE (HRG parsers were for classes too restricted), why grammars. Short summary of the results. Remainder.

0.2 Related Work

In this section, we offer a literary review on the topics of graph grammars and triple graph grammars as well as we indicate published works that are related with our approach. Here, we focus on the node label and the hyperedge replacement approach for graph grammars. Nevertheless, the field does not restrict to this topic, instead, there is a myriad of different approaches to it, for example, the algebraic approach [13]. We refer to context-free and context-sensitive grammars, inspired by the use of such classification for string grammars, in a relaxed way without any compromise to the correct definition of context-freeness for graph grammars.

Hyperedge replacement graph grammars (HRG) are context-free grammars with semantics based on the replacement of hyperedges by hypergraphs [12] governed by morphisms. Prominent polynomial-time top-down and shift-reduce parsing techniques for classes of such grammars can be found in [10, 11, 5, 8] and applications for syntax definition of a visual language can be found in [28, 15].

We divide the node label replacement approaches into context-sensitive and context-free approaches, we refer to context-sensitive and context-free grammars, inspired by the use of such classification for string grammars, in a relaxed way without any compromise to any definition of context-freeness for

graph grammars. The context-sensitive field includes the *layered graph grammar*, whose semantics consists of the replacement of graphs by other graphs governed by morphisms [30] and for which exponential-time bottom-up parsing algorithms have been proposed [29, 6, 19]. Another context-sensitive formalism is the *reserved graph grammar*, that is based on the replacement of directed graphs by necessarily greater directed graphs governed by simple embedding rules [37] and for which exponential and polynomial-time bottom-up algorithms have been proposed in [36, 38].

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 [31]. For various classes of these grammars, there exists polynomial-time top-down and bottom-up parsing algorithms [17, 18, 31, 35]. The recognition complexity and generation power of such grammars have also been analyzed [16, 25]. 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 [23, 34]. For some classes of these grammars, polynomial-time bottom-up parsing algorithms and automaton formalisms were proposed and analyzed [24, 7]. In special, one of these classes is the *boundary graph grammar with neighborhood-controlled embedding* (BNCE), that is used to construct our own formalism. Moreover, it is worth mentioning that, according to [14], BNCE and HRG have the same generative power.

Beyond the approaches presented above, there is a myriad of alternative proposals for graph grammars, including a context-sensitive NCE [1], an edge-based grammar [33], a grammar that replaces star graphs by other graphs [9], a coordinate system-based grammar [27] and a regular graph grammar [20].

Regarding TGG [32], a 20 years review of the realm is put forward by Anjorin et al. [2]. In special, advances are made in the direction of expressiveness with the introduction of application conditions [26] and of modularization [3].

Furthermore, in the algebraic approach for graph grammars, we have found proposals that introduce inheritance [4, 21] and variables [22] 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.

0.3 Theoretical Review

In this section, we introduce the theoretical concepts used along this thesis. The definitions below are taken from the works of ...We first go on to define graphs and graph grammars and then, building upon it, we construct the so-called triple graph grammars.

0.3.1 Graph Grammars

We start presenting our notation for graphs and grammars, accompanied by examples, then we introduce the dynamic aspects of the graph grammar formalism that is, how graph grammars are to be interpreted.

Definition. 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 $\phi : V \to \Sigma$. 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 denote the set of all graphs over Σ by $\mathcal{G}_\Sigma$. In special, we do not allow loops (vertices of the form (v, l, v)), but multi-edges with different labels are allowed.

If $\phi_G(v) = a$ we say v is labeled by a. Two vertices v and w are neighbors (also adjacent) if, and only if,there is one or more edges between them, that is, $(v, \_, w) \in E_G \vee (w, \_, v) \in E_G$. Two graphs G and H are disjoint if, and only if,$V_G \cap V_H = \emptyset$.

We define also de function $\text{neigh}_G : 2^{V_G} \to 2^{V_G}$, that applied to U gives the set of neighbors of 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. A morphism of graphs G and H is a total mapping $m : V_G \to V_H$.

Definition. 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, morphism do not require bijectivity nor label or edge-preserving properties.

Definition. 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, $\nexists (v, \_, w) \in E_G.\ \phi_G(v) \in \Gamma \wedge \phi_G(w) \in \Gamma$.

We use graphs to represent models, first because of the extensive theory behind them and, second, because their very abstract structure suits the description of a large spectrum of practical models. In the following we introduce graph grammars, which also suit our needs very well, because they serve as a very effective tool to characterize (possibly infinite) sets of graphs using very few notation.

Definition. 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 vertex labels. NCE graph grammars are often referred to as graph grammars or simply as grammars.

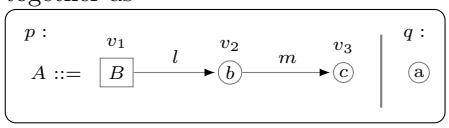
For convenience, define the start graph of GG as $Z_{GG} := (\{v_s\}, \emptyset, \{v_s \mapsto S\})$

Vertices from the right-hand sides of rules labeled by non-terminal (terminal) symbols are said to be non-terminal (terminal) vertices.

Notice that, in the original definition of NCE grammars [23], the left-hand side of the productions were allowed to contain any connected graph. 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 to mean a 1-edNCE grammar.

Definition. 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 if, and only if,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 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. 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:

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

Notice that, without loss of generality, we set $\omega(t) = \emptyset$ for all vertices t without an image defined in ω.

If G *concretely derives* in one step into any graph H' isomorphic to 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 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 from former neighbors w of v to 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 and cannot be used to define embedding. As an additional note, it is worth mentioning, that string grammars have no embedding because a replaced symbol in a string has "connections" only with its left and right neighbors, so the replacement is always "connected" with both sides.

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

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

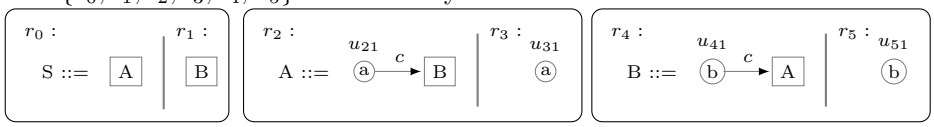
Definition. The language $L(GG)$ generated by the grammar GG is the set of all graphs containing only terminal vertices derived from the start graph Z_{GG}, that is

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

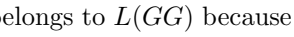
It is clear that, for every graph $G \in L(GG)$, there is at least one finite derivation $(Z_{GG} \overset{r_0,v_0}{\Rightarrow} \dots \overset{r_{n-1},v_{n-1}}{\Rightarrow} G)$ with $n \geq 1$, but it is not guaranteed that this derivation be unique. In the case that there is more than one derivation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

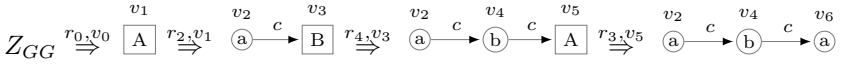
Below we give one example of a grammar whose language consists of all chains of one or more vertices with interleaved vertices labeled with a and b.

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



0.3.2 Triple Graph Grammars

Building upon the concepts of graphs and graph grammars, we present, in the following, our understanding over triple graphs and triple graph grammars (TGG), supported by the TGG specification from [32].

Definition. 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 0.3.1), respectively, the source graph G_s, the correspondence graph G_c and the target graph G_t, together with two injective morphisms (see 0.3.1) $m_s : V_{G_c} \to V_{G_s}$ and $m_t : V_{G_c} \to G_{G_t}$. Directed labeled triple graphs are often referred to simply as triple graphs 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 $\phi_{TG} := \phi_{G_s} \cup \phi_{G_c} \cup \phi_{G_t}$. Moreover, we define the special empty triple graph as $\varepsilon := E \overset{ms}{\leftarrow} E \overset{mt}{\rightarrow} E$ with $E = (\emptyset, \emptyset, \emptyset)$ and $ms = mt = \emptyset$.

Definition. 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.

As stated before, triple graphs are for us a good tool to express relations between the vertices of two graphs. In the context of model transformation, where graphs represent models, a triple graph holds, for example, a source model and a target model generated from the source, together with the relationship between their vertices. We also advise that in literature, TGG are often modeled as typed graphs, but we judge that for our circumstance labeled graphs fit better and we are convinced that such divergence does not threat the validity of our approach.

Below we start introducing the standard definition of TGG of the current research's literature. As the reader should notice, this definition of TGG does not fit our needs optimally, because it defines a context-sensitive graph grammar whilst we wish a context-free graph grammar to use together with the NCE graph grammar formalism. Hence, after presenting the conventional TGG definition, we refine it to create a NCE TGG, that fits our context best.

Definition. A triple graph grammar $TGG = (\Sigma, \Delta \subseteq \Sigma, S \in \Sigma, P)$ consists of, analogously to graph grammars (see Definition 0.3.1), 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 $L = L_s \leftarrow L_c \rightarrow L_t$ and $R = R_s \leftarrow R_c \rightarrow R_t$ and $L \subseteq R$.

Definition. 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. We might refer to the complete embedding function by $\omega := \omega_s \cup \omega_t$.

For convenience, define 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. 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 if, and only if,all its rules' right-hand sides are Γ-boundary triple graphs.

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 crescent. 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. 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 disjoint from any right-hand side 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:

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

Notice that, without loss of generality, we set $\omega(t) = \emptyset$ for all vertices t without an image defined in ω.

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 de informally understood as concrete derivations (see 0.3.1) 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, which edges are not important for our application. Nevertheless, the addition of such a mechanism for the correspondence graph should not be a problem if it is desired.

Definition. 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. 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 0.3.2. 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. Pseudocode to Controlflow. This example illustrates the definition of a BNCE 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*.

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 Just f
end if

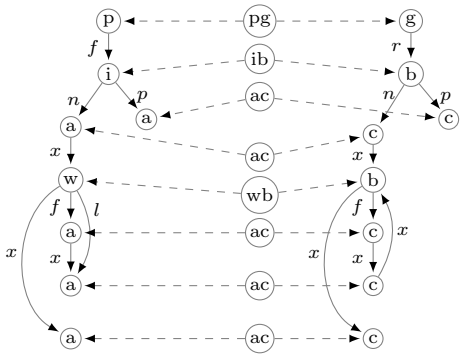
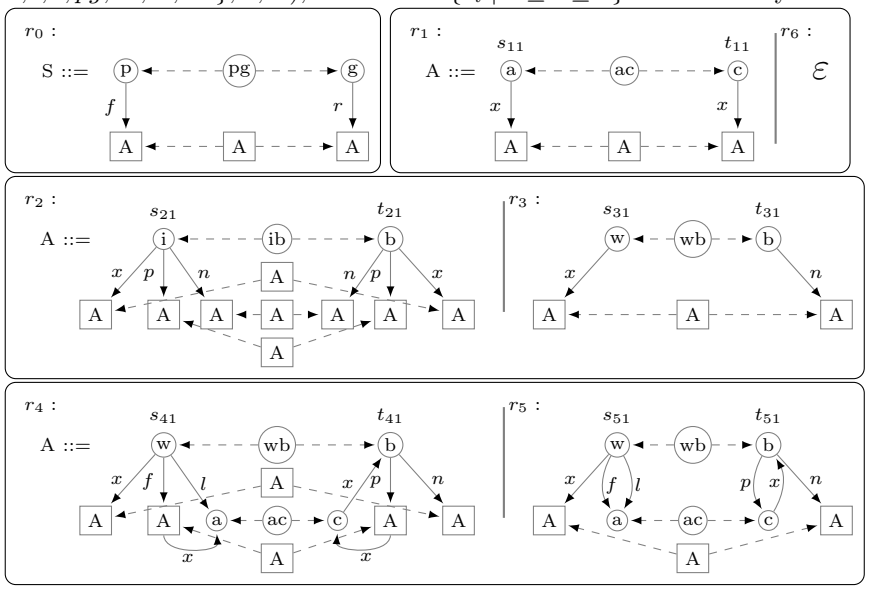


Figure 1: A program written in pseudo-code on the left and its correspondent triple graph with the $PseudoCode$ and the $ControlFlow$ graphs on the right

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.

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 $1 \leq i \leq 10$.

0.4 Parsing of Graphs with BNCE Graph Grammars

In the last section we cleared how the concepts of graphs and languages fit together. In this section we are interested in the problem of deciding, given a BNCE graph grammar GG and a graph G, whether $G \in L(GG)$. This is sometimes called the *membership* problem and can be solved through a recognizer algorithm that always finishes answering yes if and only if $G \in L(GG)$ and no otherwise. A slight extension of this problem is the *parsing* problem, which consists of deciding if $G \in L(GG)$ and finding a derivation $Z_{GG} \Rightarrow^* G$.

The parsing algorithm posed in this section is an imperative view of the method proposed by (), which is basically a version fro graphs of the well-known CYK (Cocke-Young-Kassami) algorithm for parsing of strings with a context-free (string) grammar. Preliminarily to the actual algorithm's presentation, we introduce some necessary concepts that are used by it. The first of them is the neighborhood preserving normal form.

Definition. A BNCE graph grammar $GG = (\Sigma, \Delta, S, P)$ is neighborhood preserving (NP), if and only if, the embedding of each rule with left-hand side A is greater or equal than the context of each A-labeled vertex in the grammar. That is, let

$$\text{cont}_{(A \to R, \omega)}(v) = \{(l, \phi_R(w)) \mid (v, l, w) \in E_R \text{ or } (w, l, v) \in E_R\} \cup \omega(v)$$

be the context of v in the rule $(A \to R, \omega)$ and

$$\eta_{GG}(A) = \bigcup_{(B \to Q, \zeta) \in P, v \in V_Q, \phi_Q(v) = A} \text{cont}_{B \to Q, \zeta}(v)$$

be the context of the symbol A in the grammar GG, then GG is a NP BNCE graph grammar, if and only if,

$$\forall r = (A \to R, \omega) \in P.\ \eta_{GG}(A) \subseteq \bigcup_{v\ in V_R} \omega(v)$$

The NP property is important to the correctness of the parsing algorithm. Furthermore, it is guaranteed that any BNCE graph grammar can be transformed in an equivalent NP BNCE graph grammar in polynomial time. More details in ()

The next paragraphs present zone vertices and zone graphs, that are our understanding of the concepts also from

Definition. A zone vertex h of a graph G over Σ is a pair $(\sigma \in \Sigma, U \subseteq V_G)$, that is, a symbol from Σ and a subset of the vertices of G.

A zone vertex can be understood as a contraction of a subgraph of G defined by the vertices U into one vertex with symbol σ.

Definition. Let $H = \{(\sigma_0, U_0), (\sigma_1, U_1), \ldots, (\sigma_m, U_m)\}$ be a set of zone vertices of a graph G over Σ with disjoint vertices (i.e. $U_i \cap U_j = \emptyset$ for all $0 \leq i, j \leq m$ and $i \neq j$) and $V(H) = \bigcup_{0 \leq i \leq m} U_i$. A zone graph $Z(H)$ for H is $Z(H) = (V, E, \phi)$ with V being the zone vertices, $E \subseteq V \times \Sigma \times V$ the edges between zone vertices and $\phi : V \to \Sigma$ the labeling function, determined by

$$
\begin{aligned}
V &= H \cup \{(\phi_G(x), \{x\}) \mid x \in neigh_G(V(H))\} \\
E &= \{((\sigma, U), l, (\eta, T)) \mid (\sigma, U), (\eta, T) \in V \text{ and } U \neq T \text{ and} \\
&\quad (u, l, t) \in E_G \text{ and } u \in U \text{ and } t \in T\} \\
\phi &= \{(\sigma, U) \mapsto \sigma\}
\end{aligned}
$$

The zone graph $Z(H)$ can be intuitively understood as a subgraph of G, where each zone vertex in $V_{Z(H)}$ is either a (σ_i, U_i) of H, which is a contraction of the vertices U_i of G, or a $(\phi_G(x), \{x\})$, which stems from x being a neighbor of some vertex in V_i.

For convenience, define $Y(H)$ as the subgraph of $Z(H)$ induced by H.

Definition. Let h be a zone vertex, r a production rule and X a (potentially empty) set of parsing trees, $(h^r \rightrightarrows X)$ is a parsing tree, whereby h is called the root node and X the children and r is optional. $D(pt)$ gives a derivation for the parsing tree pt, which can be calculated by performing a depth-first walk on pt,

starting from its root node, producing as result a sequence of derivation steps that correspond to each visited node and its respective rule. Additionally, a set of parsing trees is called a parsing forest.

Finally, the Algorithm 1 displays the parsing algorithm of graphs with a NP BNCE graph grammar. Informally, the procedure follows a bottom-up strategy that tries to find production rules in GG that generate zone graphs of G until it finds a rule that generates a zone graph containing all vertices of G and finishes answering yes and returning a valid derivation for G or it exhausts all the possibilities and finishes answering no.

Algorithm 1 Parsing Algorithm for NP BNCE Graph Grammars

Require: GG is a valid NP BNCE graph grammar

Require: G is a valid graph over Δ ▷ G has terminal vertices only

function $parse(GG = (\Sigma, \Delta, S, P), G = (V_G, E_G, \phi_G))$: $Derivation$

$bup \leftarrow \{(\phi_G(x), \{x\}) \mid x \in V_G\}$ ▷ start bup with trivial zone vertices

$pf \leftarrow \{(b \rightrightarrows \emptyset) \mid b \in bup\}$ ▷ initialize parsing forest

repeat

$h \leftarrow$ **select**$\{X \subseteq bup \mid$ for all $U_i, U_j \in X$ with $i \neq j$. $U_i \cap U_j = \emptyset\}$

for all $d \in \Gamma$ **do** ▷ for each non-terminal symbol

$r \leftarrow$ any $\{(d \to R, \omega) \in P \mid R \cong Y(h)\}$

$l \leftarrow (d, V(h))$

if $Z(\{l\}) \overset{r,l}{\Rightarrow} Z(h)$ **then**

$bup \leftarrow bup \cup \{l\}$ ▷ new zone vertex found

$pf \leftarrow pf \cup \{(l^r \rightrightarrows \{(z^y \rightrightarrows X) \mid (z^y \rightrightarrows X) \in pf, z \in h\})\}$

end if

end for

until $(S, V_G) \in bup$ ▷ if found the root, stop

return $(S, V_G) \in bup$ **?** Just $D(((S, V_G)^y \rightrightarrows X) \in pf)$ **:** Nothing

end function

Ensure: *return* is either Nothing or of the form Just $Z_{GG} \Rightarrow^* G$

The variable *bup* (*bup* stands for bottom-up parsing set, see ())is started with the trivial zone vertices of G, each containing only one vertex of V_G, and grows iteratively with bigger zone vertices that can be inferred using the grammar's rules and the elements of *bup*.

The variable h stands for handle and is any subset from *bup* chosen to be evaluated for the search of new zone vertices to insert in *bup*. The procedure **select** gives one yet not chosen handle or an empty set and cares for the termination of the execution. Then, for the chosen h, rules r with left-hand side d and right-hand side isomorphic to $Y(h)$ that produce $Z(h)$ from $Z(\{l\})$ are searched. If any is found, then $l = (d, V(h))$ is inserted into *bup*. This basically means that it found a zone vertex that encompasses the vertices $V(h)$ (a possibly bigger subset than other elements in *bup*), from which, through the application of a sequence of rules, we can produce the subgraph of G induced by $V(h)$. This information is saved in the parsing forest pf in form of a parsing tree with node l and children $(z^y \rightrightarrows X)$, already in the parsing forest pf, for all $z \in h$.

If, in some iteration the zone vertex (S, V_G) is inferred, then it means that the whole graph G can be produced through the application of a derivation starting from the start graph Z_{GG} and thus $G \in L(GG)$. This derivation is, namely, the result of a depth-first walk in the parsing tree whose root is (S, V_G). If, otherwise, all possibilities for h were exhausted without inferring such zone vertex, then Nothing is returned, what means that G cannot de parsed with GG and therefore $G \notin L(GG)$.

0.5 Model Transformation with BNCE Triple Graph Grammars

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 BNCE 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 0.3.2)— 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 definitions of the s and t functions, that extract the source and the target part of production rules.

Definition. 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 and $\mathrm{t}(r) = (A \rightarrow G_t, \omega_t)$ gives the target part. Moreover, $\mathrm{s}^{-1}((A \rightarrow G_s, \omega_s)) = r$ and $\mathrm{t}^{-1}((A \rightarrow G_t, \omega_t)) = r$ are the inverse of these functions.

Definition. 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 and $\mathrm{T}(TGG) = (\Sigma, \Delta, S, \mathrm{t}(P))$ gives the target grammar of TGG.

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

Definition. A triple graph grammar $TGG = (\Sigma, \Delta, S, P)$ is non-terminal consistent (NTC) if and only if, for all rules $(A \to (G_s \overset{ms}{\leftarrow} G_c \overset{mt}{\to} G_t), \omega_s, \omega_t) \in P$, 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 $G \leftarrow C \to T$ 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, $\overline{D} = Z_{S(TGG)} \overset{s(r_0), s_0}{\Rightarrow} G_s^1 \overset{s(r_1), s_1}{\Rightarrow} \dots \overset{s(r_{k-1}), s_{k-1}}{\Rightarrow} G_s^k$ 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

$Z_s \leftarrow Z_c \to Z_t \overset{r_0, s_0, c_0, t_0}{\Rightarrow}_{TGG} G_s^1 \leftarrow G_c^1 \to G_t^1$, then, by Definition 0.3.2,

$r_0 = (S \to (R_s \leftarrow R_c \to R_t), \omega_s, \omega_t) \in P$ and, by Definition 0.5,

$\mathrm{s}(r_0) = (S \to R_s, \omega_s) \in SP$

Hence, using it plus the configuration of $\phi_{Z_s}(s_0)$, $V_{G_s^1}$, $E_{G_s^1}$ and $\phi_{G_s^1}$ and the equality $Z_s = Z_{SG}$, we have, by Definition 0.3.1, $Z_{SG} \overset{\mathrm{s}(r_0), s_0}{\Rightarrow}_{SG} G_s^1$.

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,

$Z_{SG} \overset{s(r_0),s_0}{\Rightarrow}_{SG} G_s^1$, then by Definition 0.3.1,

$\mathrm{s}(r_0) = (S \to R_s, \omega_s) \in SP$ and, using the bijectivity of s, we get

$r_0 = \mathrm{s}^{-1}(s(r_0)) = (S \to (R_s \leftarrow R_c \to R_t), \omega_s, \omega_t) \in P$

Hence, using it plus the configuration of $\phi_{Z_{SG}}(s_0)$, $V_{G_s^1}$, $E_{G_s^1}$ and $\phi_{G_s^1}$, 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 0.3.2 $Z_{TGG} \overset{r_0,s_0,c_0,t_0}{\Rightarrow}_{TGG} G_s^1 \leftarrow G_c^1 \to G_t^1$.

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 $Z_{SG} \Rightarrow^*_{SG} G_s^i \overset{s(r_i),s_i}{\Rightarrow}_{SG} G_s^{i+1}$ is a derivation in SG and vice-versa, 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

$G_s^i \overset{ms_i}{\leftarrow} G_c^i \overset{mt_i}{\to} G_t^i \overset{r_i,s_i,c_i,t_i}{\Rightarrow}_{TGG} G_s^{i+1} \leftarrow G_c^{i+1} \to G_t^{i+1}$, then, by Definition 0.3.2,

$r_i = (S \to (R_s \leftarrow R_c \to R_t), \omega_s, \omega_t) \in P$, and by Definition 0.5,

$\mathrm{s}(r_i) = (S \to R_s, \omega_s) \in SP$

Hence, using it plus 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, by Definition 0.3.1, $G_s^i \overset{\mathrm{s}(r_i),s_i}{\Rightarrow}_{SG} G_s^{i+1}$.

In the other direction, we choose, using the bijectivity from the range restricted 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 $\phi_{G_s^i}(s_i) \in \Gamma$, it is clear that $\phi_{G_s^i}(s_i) = \phi_{G_c^i}(c_i) = \phi_{G_t^i}(t_i)$.

In this case since

$G_s^i \overset{s(r_i),s_i}{\Rightarrow}_{SG} G_s^{i+1}$, then, by Definition 0.3.1,

$\mathrm{s}(r_i) = (A \to R_s, \omega_s) \in SP$ and, using the bijectivity of s, we get

$r_i = \mathrm{s}^{-1}(s(r_i)) = (A \to (R_s \leftarrow R_c \to R_t), \omega_s, \omega_t) \in P$

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 0.3.2, 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 1. 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 2. Thereby it is required that the TGG be neighborhood preserving (NP), what poses no problem to our procedure, once any TGG can be transformed into the neighborhood preserving normal form.

Algorithm 2 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 0.5

$\overline{D} \leftarrow parse(SG, G)$ ▷ use algorithm 1

if $\overline{D} = Z_{SG} \Rightarrow^*_{SG} G$ **then** ▷ if parsed successfully

from $\overline{D}$ construct $D = Z_{TGG} \Rightarrow^*_{TGG} G \leftarrow C \rightarrow T$

return Just T

else

return Nothing ▷ no T satisfies $(G \leftarrow C \rightarrow T) \in L(TGG)$

end if

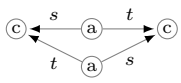
end function

Ensure: *return* is either Nothing or Just T, such that $(G \leftarrow C \rightarrow T) \in L(TGG)$

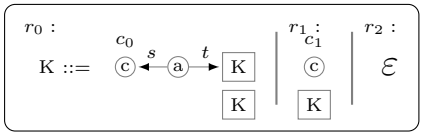
0.6 An Extension of NCE Triple Graph Grammars with Application Conditions

The NCE graph grammar formalism from [23], presented in the previous sections, can define with very few rules the languages of several classes of labeled graphs, including trees, path graphs, star graphs, control-flow graphs, edgeless graphs, complete graphs, and others. However, it is at least difficult to define the languages of other classes, like the class-diagram graphs, with NCE graph grammars. In this Section, we approach the problem of defining a NCE graph grammar for these classes of graphs and propose a solution for that by means of an extension of NCE that includes application conditions.

Class diagrams are commonly used to model object-oriented software artifact that are composed of several classes related by associations. For the sake of demonstrating the problem of NCE with class diagrams, consider a simplified view of the class-diagrams graphs, in which a vertex has either label c or a, respectively representing a class or an association, and an edge between an association and a class with label s (t) signalizes that the class is the source (target) of the association. In Figure 2a, a class-diagram graph with two classes connected by two associations is depicted. An attempt for a NCE graph grammar that would describe the language of all class-diagram graphs is $GG = (\{K, a, b, s, t\}, \{a, b, s, t\}, K, \{r_0, r_1, r_2\})$, with r_0, r_1, and r_2 depicted in Figure 2b and $\omega_0(c_0) = \omega_1(c_1) = \{(t, a)\}$ and $\omega_0 = \emptyset$ being the complete embedding definition of the rules r_0, r_1, and r_2, respectively.



(a) ....

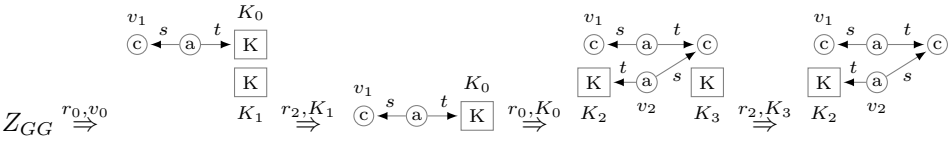


(b) ...

Figure 2: ...

The problem of the graph grammar GG is that it does not define the complete

language of the class-diagram graph. In fact, the graph in Figure 2a is not in $L(GG)$. To see this, consider the following derivation in GG:



This is the closest we get to deriving the graph in Figure 2a using GG. Thereby, we would like to connect the association v_2 to the class v_1 but it is not possible, because v_1 was not a neighbor of the vertex K_0 that preceded v_2. Notice that a vertex in any sentential form can only be either connected to vertices that stem from the same rule application or to neighbors of its precedent vertex. In fact, this seems to be a general characteristic for context-free grammars, where the information about elements in the context of the precedents are not available for descendant elements. In order to overcome it, one could potentially elaborate an alternative grammar that defines the desired language completely and concisely, but we believe that such ad-hoc solution would include a bigger number of rules and add complexity to the grammar. With that in mind, we propose in the sequel an extension of the NCE grammar formalism with positive application conditions (PAC) that solves this issue.

In NCE graph grammars with PAC, rules' right-hand sides are equipped with application conditions in form of special vertices that are produced by derivation steps and removed by so-called resolution steps. A resolution step is responsible for removing such special vertices and moving their adjacent edges to other vertices. This resolution mechanism allows that the vertex v_2 from the previous example be connected to v_1.

In order to define the PAC mechanism in detail, the definitions of rule and derivation step are augmented as follows.

Definition. A production rule with PAC is of the form $(A \to R, \omega, U)$ with A, R and ω as described in 0.3.1 and $U \subseteq \{v \in V_R \mid \phi_R(v) \in \Delta\}$, the set of special vertices, called PAC vertices.

If a graph grammar has at least one rule with PAC, then we say it is a graph grammar with PAC.

Definition. A *concrete derivation step* with PAC in the graph grammar GG is of the form $G \overset{r,v,U}{\Rightarrow}_{GG} H$ with G, H, v being as described in 0.3.1, and $r = (A \to R, \omega, U)$ being a production rule with PAC. Given that, a *derivation step* is, analogously, of the form $G \overset{r,v,W}{\Rightarrow}_{GG} H'$ with $W = m(U)$ where m is the isomorphism from H and H'.

So far, PAC vertices do not change anything in the behavior of a derivation step and the set U in a derivation step serves just to tag which vertices are PAC in a sentential form. Nevertheless, PAC vertices play an important role on a resolution step, defined below.

Definition. Let $GG = (\Sigma, \Delta, S, P)$ be a graph grammar and G a graph over Δ, G resolves into H with the resolution partial function $s : V_G \nrightarrow V_G$, we write $G \overset{s}{\rightarrowtail} H$ and call it a resolution step, if, and only if, the following holds:

$$
\begin{aligned}
&\forall v \in \operatorname{dom} s.\ s(v) \neq v \text{ and } \phi_G(s(v)) = \phi_G(v) \text{ and} \\
&V_H = V_G \setminus \operatorname{dom} s \text{ and} \\
&E_H = (E_G \setminus (\{(u,l,t) \mid u \in \operatorname{dom} s, (u,l,t) \in E_G\} \\
&\qquad\qquad \cup \{(t,l,u) \mid u \in \operatorname{dom} s, (t,l,u) \in E_G\})) \\
&\quad \cup \{(s(u),l,t) \mid u \in \operatorname{dom} s, (u,l,t) \in E_G\} \\
&\quad \cup \{(t,l,s(u)) \mid u \in \operatorname{dom} s, (t,l,u) \in E_G\}
\end{aligned}
$$

A resolution step can be informally understood as the removal of the PAC vertices of G— that are in the domain of the resolution function s —followed by the redirection of the edges adjacent to the PAC vertices to other vertices of H.

For the PAC mechanism to work, it is still necessary to combine derivation and resolution steps to define the language of a grammar with PAC, what we do in the following.

Definition. A derivation D in a graph grammar with PAC is a sequence of n derivation steps followed by n resolution steps with $n > 0$, as follows:

$$D = (G_0 \overset{r_0,v_0,W_0}{\Rightarrow} G_1 \overset{r_1,v_1,W_1}{\Rightarrow} \dots \overset{r_{n-1},v_{n-1},W_{n-1}}{\Rightarrow} G_n^0 \overset{s_0}{\rightarrowtail} G_n^1 \overset{s_1}{\rightarrowtail} \dots \overset{s_{n-1}}{\rightarrowtail} G_n^n)$$

with s_i being a resolution total function $s_i : m_i(W_i) \to V_{G_n^i}$ and $m_i : W_i \to V_{G_n^i}$ the mapping from the PAC vertices generated on the derivation step i to their correspondent vertices in G_n^i, for all $0 \leq i < n$.

Notice that, the mapping m of the previous definition exists and is bijective because all PAC vertices are, by definition, terminal and, therefore, are not deleted by derivation steps and, moreover, the images of all m_i are pair-wise disjunct.

Definition. The language $L(GG)$ generated by the grammar GG with PAC is

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

0.7 Implementation

Concrete implementation. Critical view.

0.8 Evaluation

In order to evaluate the proposed BNCE TGG formalism, we compare the amount of rules and elements (vertices, edges and mappings) we needed to describe some typical 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, in rule the r_2 of this TGG (see Example 0.3.2), a node in a positive branch of an if-labeled vertex is never connected with a node in the negative branch.

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

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

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 amount 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 amount of rules and elements.

The $Star2Wheel$ transformation consists of transforming star graphs, which are complete bipartite graphs $K_{1,k}$, with the partitions 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 write this transformation in standard TGG, specially because of the rules' monotonicity (see Definition 0.3.2). 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.

	Standard TGG		BNCE TGG	
Transformation	Forward	Backward	Forward	Backward
Pseudocode2Controlflow				
BTree2XBTree				
Star2Wheel	-	-		
Total				
Average				

Table 2: Results of the empirical evaluation of the B-NLC TGG in comparison with standard TGG

0.9 Conclusion

Summary and closing words. Future work (e.g. lexicalization for model synchronization).

Bibliography

[1] Yoshihiro Adachi, Suguru Kobayashi, Kensei Tsuchida, and Takeo Yaku. An nce context-sensitive graph grammar for visual design languages. In *Visual Languages, 1999. Proceedings. 1999 IEEE Symposium on*, pages 228–235. IEEE, 1999.

[2] Anthony Anjorin, Erhan Leblebici, and Andy Schürr. 20 years of triple graph grammars: A roadmap for future research. *Electronic Communications of the EASST*, 73, 2016.

[3] Anthony Anjorin, Karsten Saller, Malte Lochau, and Andy Schürr. Modularizing triple graph grammars using rule refinement. In *International Conference on Fundamental Approaches to Software Engineering*, pages 340–354. Springer, 2014.

[4] Roswitha Bardohl, Hartmut Ehrig, Juan De Lara, and Gabriele Taentzer. Integrating meta-modelling aspects with graph transformation for efficient visual language definition and model manipulation. In *International Conference on Fundamental Approaches to Software Engineering*, pages 214–228. Springer, 2004.

[5] Henrik Björklund, Frank Drewes, and Petter Ericson. Between a rock and a hard place–uniform parsing for hyperedge replacement dag grammars. In *International Conference on Language and Automata Theory and Applications*, pages 521–532. Springer, 2016.

[6] Paolo Bottoni, Gabriele Taentzer, and A Schurr. Efficient parsing of visual languages based on critical pair analysis and contextual layered graph transformation. In *Proceeding 2000 IEEE International Symposium on Visual Languages*, pages 59–60. IEEE, 2000.

[7] Franz-Josef Brandenburg and Konstantin Skodinis. Finite graph automata for linear and boundary graph languages. *Theoretical Computer Science*, 332(1-3):199–232, 2005.

[8] David Chiang, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, Bevan Jones, and Kevin Knight. Parsing graphs with hyperedge replacement grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 924–932, 2013.

[9] Frank Drewes, Berthold Hoffmann, Dirk Janssens, and Mark Minas. Adaptive star grammars and their languages. *Theoretical Computer Science*, 411(34-36):3090–3109, 2010.

[10] Frank Drewes, Berthold Hoffmann, and Mark Minas. Predictive top-down parsing for hyperedge replacement grammars. In *International Conference on Graph Transformation*, pages 19–34. Springer, 2015.

[11] Frank Drewes, Berthold Hoffmann, and Mark Minas. Predictive shift-reduce parsing for hyperedge replacement grammars. In *International Conference on Graph Transformation*, pages 106–122. Springer, 2017.

[12] Frank Drewes, H-J Kreowski, and Annegret Habel. Hyperedge replacement graph grammars. In *Handbook Of Graph Grammars And Computing By Graph Transformation: Volume 1: Foundations*, pages 95–162. World Scientific, 1997.

[13] Hartmut Ehrig, Grzegorz Rozenberg, Hans-J Kreowski, and Ugo Montanari. *Handbook of graph grammars and computing by graph transformation*, volume 3. World Scientific, 1999.

[14] Joost Engelfiet and Grzegorz Rozenberg. A comparison of boundary graph grammars and context-free hypergraph grammars. *Information and Computation*, 84(2):163–206, 1990.

[15] Joost Engelfriet and Sebastian Maneth. Tree languages generated by context-free graph grammars. In *International Workshop on Theory and Application of Graph Transformations*, pages 15–29. Springer, 1998.

[16] M Flasiński. Power properties of nlc graph grammars with a polynomial membership problem. *Theoretical Computer Science*, 201(1-2):189–231, 1998.

[17] Mariusz Flasiński. On the parsing of deterministic graph languages for syntactic pattern recognition. *Pattern Recognition*, 26(1):1–16, 1993.

[18] Mariusz Flasiński and Zofia Flasińska. Characteristics of bottom-up parsable ednlc graph languages for syntactic pattern recognition. In *International Conference on Computer Vision and Graphics*, pages 195–202. Springer, 2014.

[19] Luka Fürst, Marjan Mernik, and Viljan Mahnič. Improving the graph grammar parser of rekers and schürr. *IET software*, 5(2):246–261, 2011.

[20] Sorcha Gilroy, Adam Lopez, and Sebastian Maneth. Parsing graphs with regular graph grammars. In *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (* SEM 2017)*, pages 199–208, 2017.

[21] Frank Hermann, Hartmut Ehrig, and Gabriele Taentzer. 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.

[22] Berthold Hoffmann. Graph transformation with variables. In *Formal Methods in Software and Systems Modeling*, pages 101–115. Springer, 2005.

[23] Dirk Janssens and Grzegorz Rozenberg. Graph grammars with neighbourhood-controlled embedding. *Theoretical Computer Science*, 21(1):55–74, 1982.

[24] Changwook Kim. Efficient recognition algorithms for boundary and linear ence graph languages. *Acta informatica*, 37(9):619–632, 2001.

[25] Changwook Kim. On the structure of linear apex nlc graph grammars. *Theoretical Computer Science*, 438:28–33, 2012.

[26] Felix Klar, Marius Lauder, Alexander Königs, and Andy Schürr. Extended triple graph grammars with efficient and compatible graph translators. In *Graph transformations and model-driven engineering*, pages 141–174. Springer, 2010.

[27] Jun Kong, Kang Zhang, and Xiaoqin Zeng. Spatial graph grammars for graphical user interfaces. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 13(2):268–307, 2006.

[28] Mark Minas. Syntax definition with graphs. *Electronic Notes in Theoretical Computer Science*, 148(1):19–40, 2006.

[29] Jan Rekers and A Schurr. A graph grammar approach to graphical parsing. In *Proceedings of Symposium on Visual Languages*, pages 195–202. IEEE, 1995.

[30] Jan Rekers and Andy Schürr. Defining and parsing visual languages with layered graph grammars. *Journal of Visual Languages & Computing*, 8(1):27–55, 1997.

[31] Grzegorz Rozenberg and Emo Welzl. Boundary nlc graph grammars—basic definitions, normal forms, and complexity. *Information and Control*, 69(1-3):136–167, 1986.

[32] Andy Schürr. Specification of graph translators with triple graph grammars. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 151–163. Springer, 1994.

[33] Zhan Shi, Xiaoqin Zeng, Song Huang, Zekun Qi, Hui Li, Bin Hu, Sainan Zhang, Yanyun Liu, and Cailing Wang. A method to simplify description and implementation of graph grammars. In *Computing, Communication and Networking Technologies (ICCCNT), 2015 6th International Conference on*, pages 1–6. IEEE, 2015.

[34] Konstantin Skodinis and Egon Wanke. Neighborhood-preserving node replacements. In *International Workshop on Theory and Application of Graph Transformations*, pages 45–58. Springer, 1998.

[35] Egon Wanke. Algorithms for graph problems on bnlc structured graphs. *Information and Computation*, 94(1):93–122, 1991.

[36] Xiaoqin Zeng, Kang Zhang, Jun Kong, and Guang-Lei Song. Rgg+: An enhancement to the reserved graph grammar formalism. In *005 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 272–274. IEEE, 2005.

[37] Da-Qian Zhang, Kang Zhang, and Jiannong Cao. A context-sensitive graph grammar formalism for the specification of visual languages. *The Computer Journal*, 44(3):186–200, 2001.

[38] Yang Zou, Xiaoqin Zeng, Yufeng Liu, and Huiyi Liu. Partial precedence of context-sensitive graph grammars. In *Proceedings of the 10th International Symposium on Visual Information Communication and Interaction*, pages 16–23. ACM, 2017.