William Bombardelli da Silva

August 26, 2018

Abstract

Contents

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). Short summary of the results. Remainder.

0.2 Related Work

TGG main references and reviews. Usability enhancements proposed for TGG. Graph grammar main references. NCE and B-eNCE graph grammar. Alternative proposal of graph grammars.

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 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 $\mathcal{G}_\Sigma$.

If $\phi_G(v) = a$ we say v is labeled by a. Two vertices v and w are neighbors (also adjacent) iff 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 iff $V_G \cap V_H = \emptyset$.

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 $\text{neigh}_G(U) = \{v \in V_G \setminus U \mid \text{ exists a } (v, l, u) \in E_G \text{ or a } (u, l, v) \in E_G \text{ with } u \in U\}$

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 if

$m(v) = w$ then $\phi_G(v) = \phi_H(w)$. 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 iff, $\nexists(v, \_, w) \in E_G.\ \phi_G(v) \in \Gamma \wedge \phi_G(w) \in \Gamma$.

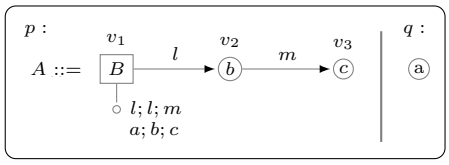
Definition. An graph grammar with neighborhood-controlled embedding (NCE graph grammar) $GG = (\Sigma, \Delta \subseteq \Sigma, S \in \Sigma, P)$ consists of a finite set of symbols Σ that is the alphabet, a subset of the alphabet $\Delta \subseteq \Sigma$ that holds the terminal symbols (we define the 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.

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. 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. Let $GG = (\Sigma, \Delta, S, P)$ be a graph grammar and G and H be two graphs over Σ disjoint from any right-hand side 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.

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 in one or more steps into H', or simply G derives into H'.

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

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

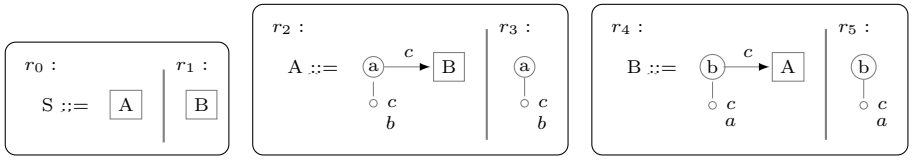
Definition. The language $L(GG)$ generated by the grammar GG is the set of all graphs containing only terminal vertices derived from the start graph Z_{GG}, that is

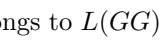
$$L(GG) = \{H \mid \phi_H(V_H) \subseteq \Delta \wedge Z_{GG} \Rightarrow^* H\}$$

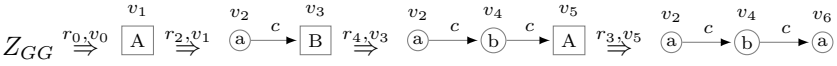
Notice that for every graph $G \in L(GG)$, there is at least one finite derivation $(Z_{GG} \overset{r_0,v_0}{\Rightarrow} \dots \overset{r_{n-1},v_{n-1}}{\Rightarrow} G)$, but it is not guaranteed that this derivation be unique. In the case that there are more than one derivation for a G, we say that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



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 (TGGs), supported by the TGG specification from ().

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) $m_s : V_{G_c} \rightarrow V_{G_s}$ and $m_t : V_{G_c} \rightarrow G_{G_t}$. Directed labeled triple graphs are often referred to simply as triple graphs 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. 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 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-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 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 0.3.1), 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 $L \subseteq R$.

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 $((A, A, A) \rightarrow (R_s \leftarrow R_c \rightarrow R_t), \omega_s, \omega_t)$ 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 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 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. Let $TGG = (\Sigma, \Delta, S, P)$ be a NCE TGG and G and H 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:

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

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

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.

Definition. 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. 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

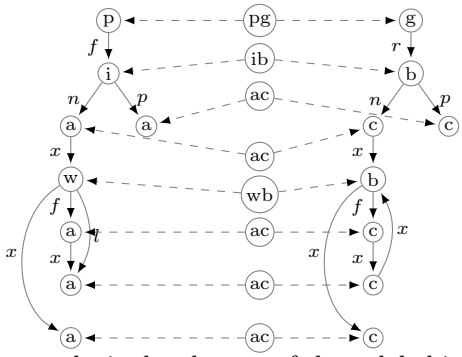
$$L(TGG) = \{H \mid \phi_H(V_H) \subseteq \Delta \wedge Z_{TGG} \Rightarrow^* H\}$$

Our concrete syntax for NCE TGG is similar to the one for NCE graph grammars and is presented below by means of the Example 0.3.2. The only difference is at the right-hand sides that depict the morphisms between the correspondence graph and source and target graphs.

Example. Pseudocode to Controlflow. This example illustrates the specification of the consistency relation between $Pseudocode$ graphs and $Controlflow$ 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 terms 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.

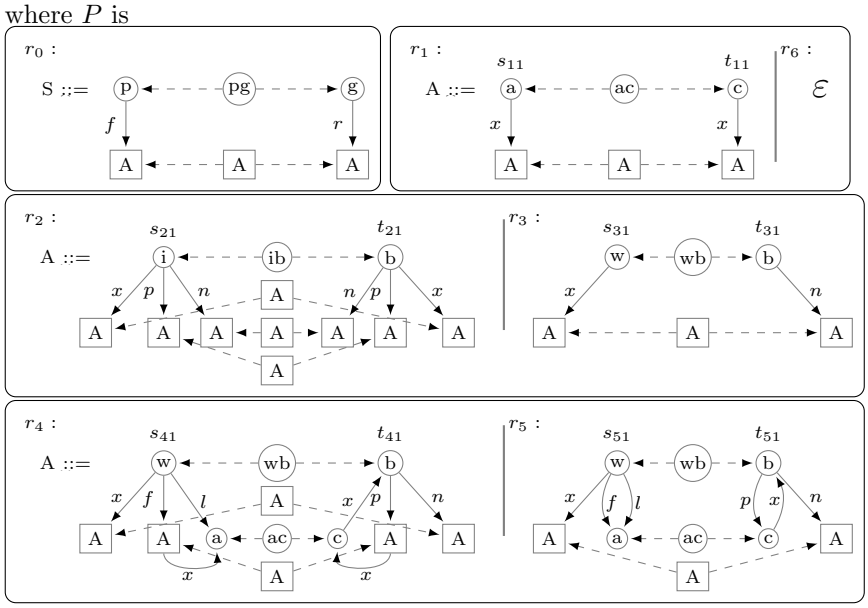
Consider, for instance, the program *main* below on the left, written in a pseudo-code. The triple graph TG on the right consists of the $Pseudocode$ graph of *main* connected to the $Controlflow$ graph of the same program through the correspondence graph in the middle of them. In such graph, the vertex 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 $1 \leq i \leq 10$.

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.

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 $z \in h$.

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 $(S, V_G) \in bup$ **?** $D(((S, V_G)^y \rightrightarrows X) \in pf)$ **:** $nothing$
  end function
Ensure: $return$ is either $nothing$ or of the form $Z_{GG} \Rightarrow^* G$

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

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

0.3.2) equivalent to finding a derivation $Z_{TGG} \Rightarrow^*_{TGG} G \leftarrow C \rightarrow T$.

Furthermore, consider the following definitions.

Definition. By extension of the Definition 0.4, a BNCE triple graph grammar TGG is neighborhood preserving if and only if, $S(TGG)$ and $T(TGG)$ are neighborhood preserving.

Definition. 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. 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. 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 $D = Z_{TGG} \overset{r_0, s_0, c_0, t_0}{\Rightarrow}_{TGG} G^1 \overset{r_1, s_1, c_1, t_1}{\Rightarrow}_{TGG} \dots \overset{r_{k-1}, s_{k-1}, c_{k-1}, t_{k-1}}{\Rightarrow}_{TGG} G^k$ be a derivation in TGG, D can equivalently be rewritten as the derivation in SG, $\overline{D} = Z_{SG} \overset{s(r_0), s_0}{\Rightarrow}_{SG} G^1_s \overset{s(r_1), s_1}{\Rightarrow}_{SG} \dots \overset{s(r_{k-1}), s_{k-1}}{\Rightarrow}_{SG} G^k_s$.*

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 0.3.2, $r_0 = ((S, S, S) \rightarrow (R_s \leftarrow R_c \rightarrow R_t), \omega_s, \omega_t) \in P$, and by Definition 0.5, $\mathrm{s}(r_0) = (S \rightarrow R_s, \omega_s) \in SP$, 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 0.3.1, 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 $V_{G^1_c}$, $V_{G^1_t}$, $E_{G^1_c}$, $E_{G^1_t}$, $\phi_{G^1_c}$, $\phi_{G^1_t}$ from Z_c and Z_t according to the Definition 0.3.2, we have $Z_{TGG} \overset{r_0, s_0, c_0, t_0}{\Rightarrow}_{TGG} G^1_s \leftarrow G^1_c \rightarrow G^1_t$

Now, for the induction step, we want to show that if $Z_{TGG} \Rightarrow^*_{TGG} G^i \overset{r_i, s_i, c_i, t_i}{\Rightarrow}_{TGG} G^{i+1}$ is a derivation in TGG, then $Z_{TGG} \Rightarrow^*_{SG} G^i_s \overset{s(r_i), s_i}{\Rightarrow}_{SG} G^{i+1}_s$ is a derivation in SG and vice-versa. 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 $G^i_s \overset{ms_i}{\leftarrow} G^i_c \overset{mt_i}{\rightarrow} G^i_t \overset{r_i,s_i,c_i,t_i}{\Rightarrow}_{TGG} G^{i+1}_s \leftarrow G^{i+1}_c \rightarrow G^{i+1}_t$, then, by Definition 0.3.2, $r_i = ((S,S,S) \rightarrow (R_s \leftarrow R_c \rightarrow R_t), \omega_s, \omega_t) \in P$, and by Definition 0.5, $\mathrm{s}(r_i) = (S \rightarrow R_s, \omega_s) \in SP$, hence, using it and the configuration of $\phi_{G^i_s}(s_i)$, $V_{G^{i+1}_s}$, $E_{G^{i+1}_s}$ and $\phi_{G^{i+1}_s}$, we have $G^i \overset{\mathrm{s}(r_i),s_i}{\Rightarrow}_{SG} G^{i+1}_s$.

In the other direction, we choose, using the bijectivity from the range-restricted 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 $\phi_{G^i_s}(s_0) = \phi_{G^i_c}(c_0) = \phi_{G^i_t}(t_i)$.

In this case, if $G^i_s \overset{s(r_i),s_i}{\Rightarrow}_{SG} G^{i+1}_s$, then by Definition 0.3.1, 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 $\phi_{G^i_s}(s_i)$, $\phi_{G^i_c}(c_i)$, $\phi_{G^i_t}(t_i)$ $V_{G^{i+1}_s}$, $E_{G^{i+1}_s}$ and $\phi_{G^{i+1}_s}$ and constructing $V_{G^{i+1}_c}$, $V_{G^{i+1}_t}$, $E_{G^{i+1}_c}$, $E_{G^{i+1}_t}$, $\phi_{G^{i+1}_c}$, $\phi_{G^{i+1}_t}$ from G^i_c and G^i_t according to the Definition 0.3.2, we have $G^i_s \leftarrow G^i_c \rightarrow G^i_t \overset{r_i,s_i,c_i,t_i}{\Rightarrow}_{TGG} G^{i+1}_s \leftarrow G^{i+1}_c \rightarrow G^{i+1}_t$.

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, 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 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 0.5
    $\overline{D} \leftarrow parse(GG, G)$ ▷ use algorithm 1
    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 $(G \leftarrow C \rightarrow T) \in L(TGG)$

0.6 An Extension of BNCE Triple Graph Grammars with Look-ahead

Problem with B-eNCE TGG. Solution with look-ahead. Critical view. Examples

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.

	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

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. 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 TGGs 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-like 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 monotonicity aspect of it (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 functions.

	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).