Model Transformation with Context-free Triple Graph Grammars and Application Conditions

Keywords: Triple Graph Grammars, NCE Graph Grammars, Model Transformation, Model-driven Development.

Abstract:

1 INTRODUCTION

Model transformation is a very important topic for the Model-driven Development (MDD) approach. In special, the problem of specifying model transformation between two metamodels is very significant for the application of MDD in practice. In this regard, the rule-based approach with triple graph grammars (TGG) proposes a technique to describe model transformation by means of a context-sensitive grammar of triple graphs.

A triple graph $G_s \overset{ms}{\leftarrow} G_c \overset{mt}{\rightarrow} G_t$ consists of three disjunct labeled graphs G_s, G_c, G_t, called source, correspondence, and target graphs, respectively, where the G_s and G_t contain elements of the two metamodels being specified, and G_c contains elements that connect G_s and G_t via two partial mappings $ms : G_c \nrightarrow G_s$ and $mt : G_c \nrightarrow G_t$ (Schürr, 1994). A triple graph can therefore express connections between elements of two metamodels (the source and the target), what in turn can be used to specify which elements of the source side is to be transformed into which elements of the target side. To accomplish that, triple graphs are combined in a triple graph grammar.

A TGG consists of a set of rules of the form $L \rightarrow R$, where L and R are triple graphs and L is contained in R ($L \subset R$). Therewith, each rule describes one unit of the transformation between the source and target metamodels referred by L and R. The application of a rule $L \rightarrow R$ on a triple graph G can be informally understood as the removal of the occurrence of L in G by R. By this means, a TGG, analogously to common string grammars, characterizes a language of triple graphs, that consists of all triple graphs generated by consecutive rules applications starting from a special initial triple graph Z_G.

Such TGG specifications are often required to be correct, easy to construct and comprehensible not only by the author. Thereby, the grammar size (i.e. the number of rules and elements in the rules) and the possibility to express abstract concepts with the rules become crucial requirements for a TGG formalism. We have identified that for some transformations these requirements are not optimally fulfilled by the current state-of-the-art of the TGG formalism. In particular, the standard version of TGG does not admit the expression of abstract notions like inheritance, what leads to grammars with bigger sizes and that are more difficult to comprehend.

In order to solve this problem, we propose in this paper a modified version of the TGG formalism that introduces the concept of non-terminal symbols and that requires the grammar rules to be context-free, that is of the from $A \rightarrow R$, where A is only one symbol and R a triple graph. Moreover, we also include in our formalism an application condition mechanism that should enhance its expressiveness and usability. Our proposal is inspired by the use non-terminal symbols in string grammars, that allows the creation of simpler grammars for which there are more efficient recognizer algorithms and with which it has become possible to built efficient programming language compilers.

Our new TGG formalism is strongly based on the results of (Rozenberg and Welzl, 1986) and (Janssens and Rozenberg, 1982) for graph grammars with non-terminal symbols, called *graph grammars with neighborhood-controlled embedding* (NCE graph grammars) . What we present in this paper is basically the transportation of the advances of NCE graph grammars to TGG, and extend it with a positive application condition mechanism. We name this new version of TGG as NCE TGG.

Our results include the presentation of the NCE TGG as well as of a parsing and transformation procedure suitable for batch bidirectional transformation and an experimental evaluation that evinces the potential of our proposal. In fact, with NCE TGG we could specify one transformation that we could not with the standard TGG, and from other four evaluated use cases, NCE TGG outperformed standard TGG in one scenario. Moreover, we claim that the cognitive

effort to comprehend a NCE TGG is lesser in comparison with TGG, by its context-free characteristic and, thus, proximity with other popular grammar formalisms for computer scientists, like the LL and LR grammars of high-level programming language compilers.

The remainder of this paper is as follows...

2 RELATED WORKS

In this section, we offer a short literary review on the graph grammar and triple graph grammar approaches that are more relevant to our work. We focus, therefore, on the context-free node label replacement approach for graph grammars, although, there is a myriad of different alternatives to it, for example, the algebraic approach (Ehrig et al., 1999). We refer to context-free grammars, inspired by the use of such classification for string grammars, in a relaxed way without any compromise to any definition.

In the node label replacement context-free formalisms stand out the *node label controlled graph grammar* (NLC) and its successor *graph grammar with neighborhood-controlled embedding* (NCE). NLC is based on the replacement of one vertex by a graph, governed by embedding rules written in terms of the vertex's label (Rozenberg and Welzl, 1986). For various classes of these grammars, there exist polynomial-time top-down and bottom-up parsing algorithms (Flasiński, 1993; Flasiński and Flasińska, 2014; Rozenberg and Welzl, 1986; Wanke, 1991). The recognition complexity and generation power of such grammars have also been analyzed (Flasiński, 1998; Kim, 2012). 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 (Janssens and Rozenberg, 1982; Skodinis and Wanke, 1998). For some classes of these grammars, polynomial-time bottom-up parsing algorithms and automaton formalisms were proposed and analyzed (Kim, 2001; Brandenburg and Skodinis, 2005). In special, one of these classes is the *boundary graph grammar with neighborhood-controlled embedding* (BNCE), that is used in our approach for model transformation.

Regarding TGG (Schürr, 1994), a 20 years review of the realm is put forward by Anjorin et al. (Anjorin et al., 2016). In special, advances are made in the direction of expressiveness with the introduction of application conditions (Klar et al., 2010) and of modularization (Anjorin et al., 2014). Furthermore, in the algebraic approach for graph grammars, we have found proposals that introduce inheritance (Bardohl et al., 2004; Hermann et al., 2008) and variables (Hoffmann, 2005) to the formalisms. Nevertheless, we do not know any approach that introduces nonterminal symbols to TGG with the purpose of gaining expressiveness or usability. In this sense, our proposal brings something new to the current state-of-the-art.

3 CONTEXT-FREE TGG

In this section, we present our main contribution, that is the application of the ideas of NCE graph grammars from (Rozenberg and Welzl, 1986) and (Janssens and Rozenberg, 1982) to the creation of a context-free version of TGG with a positive application condition (PAC NCE TGG). For that end, consider first some preliminary definitions.

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 $\phi : V \to \Sigma$.

We refer to directed labeled graphs often referred simply as graphs. For a fixed graph G we refer to its components as V_G, E_G and ϕ_G. Two graphs G and H are disjoint if, and only if, $V_G \cap V_H = \emptyset$. In special, we do not allow loops (vertices of the form (v, l, v)).

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 $\phi_G(v) = \phi_H(m(v))$.

We denote the equivalence class of all graphs isomorphic to G by $[G]$.

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, respectively, the source graph G_s, the correspondence graph G_c and the target graph G_t, together with two injective partial morphisms $m_s : V_{G_c} \rightharpoonup V_{G_s}$ and $m_t : V_{G_c} \rightharpoonup G_{G_t}$.

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

Definition. A triple graph grammar with neighborhood-controlled embedding with positive application conditions (PAC 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 $(A \to (R_s \leftarrow R_c \to R_t), \omega_s, \omega_t, U_s, U_t)$ 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 and $U_s \subseteq \{v \in V_{R_s} \mid \phi_{R_s}(v) \in \Delta\}$ the set of PAC vertices of R_s and $U_t \subseteq \{v \in V_{R_t} \mid \phi_{R_t}(v) \in \Delta\}$ the set of PAC vertices of R_t.

If all rules of a PAC NCE TGG have $U = \emptyset$, then we refer to it simply as NCE TGG. For convenience, we define the start triple graph of TGG as $Z_{TGG} := Z_s \overset{ms}{\leftarrow} Z_c \overset{mt}{\to} Z_t$ where $Z_s = (\{s_0\}, \emptyset, \{s_0 \mapsto S\})$, $Z_c = (\{c_0\}, \emptyset, \{c_0 \mapsto S\})$, $Z_t = (\{t_0\}, \emptyset, \{t_0 \mapsto S\})$, $ms = \{c_0 \mapsto s_0\}$ and $mt = \{c_0 \mapsto t_0\}$.

The application of a grammar rule $r = (A \to (R_s \leftarrow R_c \to R_t), \omega_s, \omega_t, U_s, U_t)$ using vertices v_s, v_c, and v_t to a triple graph G that produces another triple graph H, we write $G \overset{r, v_s, v_c, v_t, U_s, U_t}{\Rightarrow}_{TGG} H$, can be informally described as the replacement of a non-terminal vertex v_s with label A and all its adjacent edges in G_s by a graph R_s plus edges e between former neighbors w of v and some vertices t of R_s, provided e's label and w's label are in the embedding specification $\omega_s(t)$. That is, the embedding function ω_s of a rule specifies which neighbors of v_s are to be connected with which vertices of R_s, 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. The same replacement procedure occurs simultaneously for the correspondence and target graph using the vertex v_c, v_t and graphs G_c, G_t, R_c and R_t.

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

If $G \overset{r, v_s, v_c, v_t, U_s, U_t}{\Rightarrow}_{TGG} H$ and $H' \in [H]$, then we write $G \overset{r, v_s, v_c, v_t, W_s, W_t}{\Rightarrow}_{TGG} H'$ and call it a derivation step, where $W_s = m(U_s)$ and $W_t = m(U_t)$ where m is the isomorphism from H and H'. We might omit identifiers, when the context let them clear.

In the sequel, we give an example for a NCE TGG that specifies the transformation between two different metamodels, the *Pseudocode* and *Controlflow* metamodels, whose models are encoded via *Pseudocode* and *Controlflow* graphs, respectively.

Example. A *Pseudocode* graph is an abstract representation of a program written in a pseudo-code where vertices refer to *actions*, *ifs* or *whiles* and edges connect these items together according to how they appear in the program. A *Controlflow* graph is a more abstract representation of a program, where vertices can only be either a *command* or a *branch*.

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

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

The NCE 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 depicted in Figure 2 and $\sigma_0 = \emptyset$, $\sigma_1(s_{11}) = \sigma_2(s_{21}) = \sigma_3(s_{31}) = \sigma_4(s_{41}) = \sigma_5(s_{51}) = \{(f, p), (x, a), (x, i), (x, w), (p, i), (n, i), (l, w), (f, w)\}$

```
program main(n)
if n < 0 then
    return Nothing
else
    f ← 1
    while n > 0 do
        f ← f * n
        n ← n − 1
    end while
    return Just f
end if
```

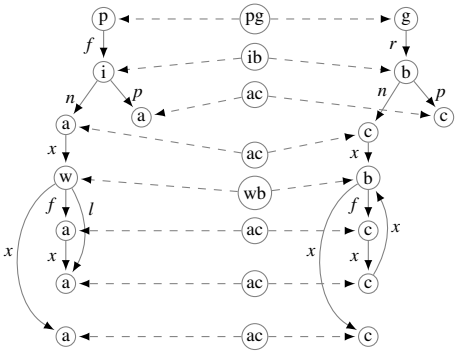


Figure 1: A program written in pseudo-code on the left and its correspondent triple graph with the *PseudoCode* and the *ControlFlow* graphs on the right

and $\tau_1(t_{11}) = \tau_2(t_{21}) = \tau_3(t_{31}) = \tau_4(t_{41}) = \tau_5(t_{51}) = \{(r,g),(x,c),(x,b),(p,b),(n,b)\}$ is the complete definition of the source and target embedding functions of the rules r_0 to r_5, respectively.

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 near it. Near each edge its respective label is positioned. The embedding function is not included in the notation, so it is expressed separately, if necessary. PAC vertices and their adjacent edges are depicted with dotted lines.

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.

Notice that, the aforementioned triple graph TG can be generated using the rules of TGG by the following sequence of derivation steps $Z_{TGG} \overset{r_0,;}{\Rightarrow} G_1 \overset{r_2,;}{\Rightarrow} G_2 \overset{r_6,;}{\Rightarrow} G_3 \overset{r_1,;}{\Rightarrow} G_4 \overset{r_6,;}{\Rightarrow} G_5 \overset{r_1,;}{\Rightarrow} G_6 \overset{r_4,;}{\Rightarrow} G_7 \overset{r_1,;}{\Rightarrow} G_8 \overset{r_6,;}{\Rightarrow} G_9 \overset{r_1,;}{\Rightarrow} G_{10} \overset{r_6,;}{\Rightarrow} TG$ with appropriate G_i for $1 \leq i \leq 10$.

So far, PAC vertices do not have any special influence in the behavior of a derivation step and the set W in a derivation step serves just to tag which vertices are PAC in a sentential form. In particular, the grammar of the Example 3 was not a PAC NCE TGG, so the definition of derivation steps that we posed before would already be enough to define the complete behavior for this grammar.

Nevertheless, PAC vertices play an important role in a PAC NCE TGG, by making it possible for a derivation step to refer to a vertex that is not essentially produced by it, but by another step. This is done via the mechanism of resolution step. The resolution step for the triple graph G to the triple graph H using the resolution partial function $\rho : V_G \nrightarrow V_G$, we write $G \overset{\rho,\tau}{\rightarrowtail} H$, can be described as the removal of the PAC vertices of G_s that are in the domain of the resolution function ρ followed by the redirection of the edges adjacent to these PAC vertices to other vertices of H_s, given by the respective image of ρ, that are required not to be PAC vertices. The same is performed with the PAC vertices in G_t by τ.

Combining derivation and resolution steps we can define the language of a PAC NCE TGG, what we do in the following.

Definition. The language $L(G)$ generated by the PAC NCE TGG G is

$$L(G) = \{H \text{ is a triple graph over } \Delta \text{ and } Z_G \overset{r_0,s_0,c_0,t_0,W_0,T_0}{\Rightarrow} G_1 \overset{r_1,s_1,c_1,t_1,W_1,T_1}{\Rightarrow} \dots \overset{r_{n-1},s_{n-1},c_{n-1},t_{n-1},W_{n-1},T_{n-1}}{\Rightarrow} G_n^0 \overset{\rho_0,\tau_0}{\rightarrowtail} G_n^1 \overset{\rho_1,\tau_0}{\rightarrowtail} \dots \overset{\rho_{n-1},\tau n-1}{\rightarrowtail} G_n^n$$

with $\rho_i : m_i(W_i) \to V_{G_n^i}$ and $\tau_i : n_i(T_i) \to V_{G_n^i}$ being the resolution total functions and $m_i : W_i \to V_{G_n^i}$ and $n_i : T_i \to V_{G_n^i}$ the mappings from the PAC vertices generated on the derivation step i to their correspondent vertices in G_n^i, for all $0 \leq i < n$.

For the sake of demonstrating how PAC vertices, consider a simplified view of the class diagram metamodel, whose models are encoded via class-diagram graphs. In a such graphs 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 3, a class-diagram graph with two classes connected by two associations is depicted.

In Figure 4 we put forward the the rules of the PAC NCE TGG whose language is the set of all class-diagram graphs. This grammar is $GG = (\{K,A,a,c,s,t\},\{a,c,s,t\},K,\{r_0,r_1,r_2,r_3\})$ with $\omega_0(c_0) = \{(t,a)\}$, $\omega_2(a_2) = \{(s,c)\}$, $\omega_2(c_2) =$

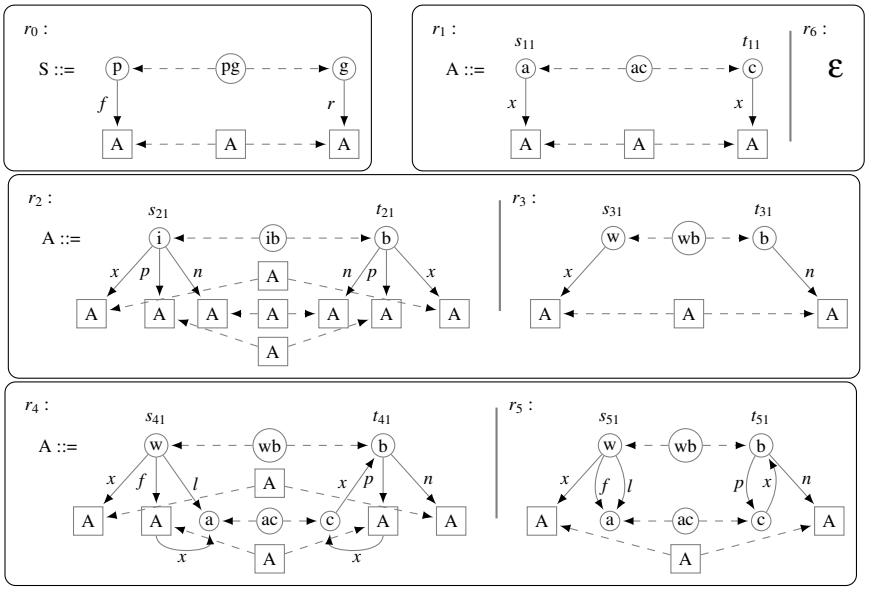


Figure 2: The depiction of the rules r_0 to r_6 of the TGG for the transformation between *Pseudocode* and *Control flow* graphs.

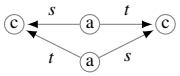


Figure 3: A class-diagram graph

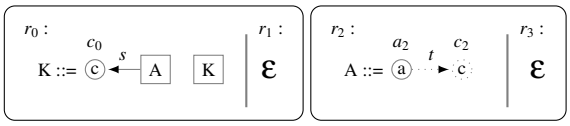
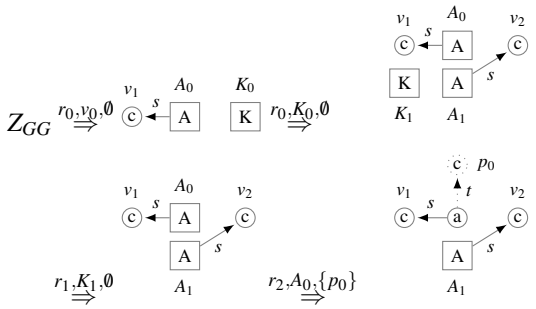
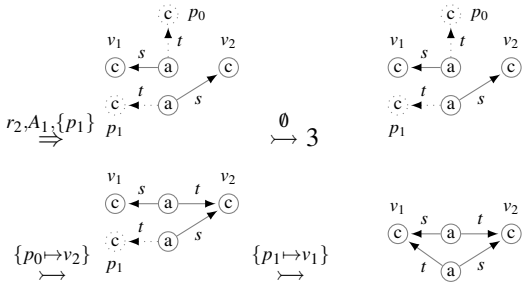


Figure 4: The depiction of the rules of *GG*

$\{(s,a),(t,a)\}$, $\omega_1 = \omega_3 = \emptyset$ being the complete characterization of the embedding functions of the respective rules.

Below, we demonstrate that the graph from Figure 3 is in $L(GG)$, by means of a sequence of four derivation steps followed by four resolution steps in *GG*.





In this sequence of derivation and resolution steps, the rule r_2 creates, through two applications, two PAC vertices p_0 and p_1, which are then removed and have their adjacent vertices moved to the vertices v_2 and v_1, respectively, through the last two resolution steps. The resolution steps have thus the power of connecting vertices that could not be connected otherwise, because in a NCE TGG without PAC, 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.

4 MODEL TRANSFORMATION WITH NCE PAC TGG

5 EVALUATION

In order to evaluate the usability of the proposed BNCE TGG formalism, we compare the number of

rules and elements (vertices, edges, and mappings) we needed to describe some model transformations in BNCE TGG and in standard TGG without application conditions. Table 1 presents these results.

In the case of *Pseudocode2Control flow*, 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 *Control flow* graphs. That is, for instance, in rule the r_2 of this TGG (see Example 3), a node in a positive branch of an if-labeled vertex is never connected with a node in the negative branch. This disjunctive aspect allows every branch to be defined in the rule (as well as effectively parsed) independently of the other branch. This characteristic makes it possible for BNCE TGG rules to be defined in a very straightforward manner and reduces the total number of elements necessary.

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

The *Star2Wheel* transformation consists of transforming star graphs, which are complete bipartite graphs $K_{1,k}$— where the partitions are named center and border— to wheel graphs, that can be constructed from star graphs by adding edges between border vertices to form a minimal cycle. We could not describe this transformation in standard TGG, especially because of the rules' monotonicity (see Definition **??**). 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.

6 CONCLUSION

REFERENCES

Anjorin, A., Leblebici, E., and Schürr, A. (2016). 20 years of triple graph grammars: A roadmap for future research. *Electronic Communications of the EASST*, 73.

Anjorin, A., Saller, K., Lochau, M., and Schürr, A. (2014). Modularizing triple graph grammars using rule refinement. In *International Conference on Fundamental Approaches to Software Engineering*, pages 340–354. Springer.

Bardohl, R., Ehrig, H., De Lara, J., and Taentzer, G. (2004). 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.

Brandenburg, F.-J. and Skodinis, K. (2005). Finite graph automata for linear and boundary graph languages. *Theoretical Computer Science*, 332(1-3):199–232.

Ehrig, H., Rozenberg, G., Kreowski, H.-J., and Montanari, U. (1999). *Handbook of graph grammars and computing by graph transformation*, volume 3. World Scientific.

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

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

Flasiński, M. and Flasińska, Z. (2014). Characteristics of bottom-up parsable ednlc graph languages for syntactic pattern recognition. In *International Conference on Computer Vision and Graphics*, pages 195–202. Springer.

Hermann, F., Ehrig, H., and Taentzer, G. (2008). 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.

Hoffmann, B. (2005). Graph transformation with variables. In *Formal Methods in Software and Systems Modeling*, pages 101–115. Springer.

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

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

Kim, C. (2012). On the structure of linear apex nlc graph grammars. *Theoretical Computer Science*, 438:28–33.

Klar, F., Lauder, M., Königs, A., and Schürr, A. (2010). Extended triple graph grammars with efficient and compatible graph translators. In *Graph transformations and model-driven engineering*, pages 141–174. Springer.

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

Schürr, A. (1994). Specification of graph translators with triple graph grammars. In *International Workshop*

	Standard TGG		BNCE TGG	
Transformation	Rules	Elements	Rules	Elements
Pseudocode2Controlflow	45	1061	**7**	**185**
BTree2XBTree	**4**	**50**	5	80
Star2Wheel	-	-	**6**	**89**
Class2Database	**6**	**98**		
Statemachine2Petrinet				

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

on Graph-Theoretic Concepts in Computer Science, pages 151–163. Springer.

Skodinis, K. and Wanke, E. (1998). Neighborhood-preserving node replacements. In *International Workshop on Theory and Application of Graph Transformations*, pages 45–58. Springer.

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