Toward Bridging the Gap Between Formal Semantics and Implementation of Triple Graph Grammars

Holger Giese, Stephan Hildebrandt, Leen Lambers

Technische Berichte Nr. 37

des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam | 37

Holger Giese | Stephan Hildebrandt | Leen Lambers

Toward Bridging the Gap Between Formal Semantics and Implementation of Triple Graph Grammars

Universitätsverlag Potsdam

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der
Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind
im Internet über http://dnb.d-nb.de abrufbar.

Universitätsverlag Potsdam 2010
http://info.ub.uni-potsdam.de/verlag.htm

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 4623 / Fax: 3474
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam.

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URL http://pub.ub.uni-potsdam.de/volltexte/2010/4521/
URN urn:nbn:de:kobv:517-opus-45219
http://nbn-resolving.org/urn:nbn:de:kobv:517-opus-45219

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-078-6

Toward Bridging the Gap Between Formal Semantics and Implementation of Triple Graph Grammars[*]

Holger Giese, Stephan Hildebrandt, and Leen Lambers

Hasso Plattner Institute at the University of Potsdam,
Prof.-Dr.-Helmert-Straße 2-3, 14482 Potsdam, Germany
{holger.giese,stephan.hildebrandt,leen.lambers}@hpi.uni-potsdam.de

Abstract. The correctness of model transformations is a crucial element for the model-driven engineering of high quality software. A prerequisite to verify model transformations at the level of the model transformation specification is that an unambiguous formal semantics exists and that the employed implementation of the model transformation language adheres to this semantics. However, for existing relational model transformation approaches it is usually not really clear under which constraints particular implementations are really conform to the formal semantics. In this paper, we will bridge this gap for the formal semantics of triple graph grammars (TGG) and an existing efficient implementation. Whereas the formal semantics assumes backtracking and ignores non-determinism, practical implementations do not support backtracking, require rule sets that ensure determinism, and include further optimizations. Therefore, we capture how the considered TGG implementation realizes the transformation by means of operational rules, define required criteria and show conformance to the formal semantics if these criteria are fulfilled. We further outline how static analysis can be employed to guarantee these criteria.

1 Introduction

Model transformations are a crucial element of Model-Driven Engineering (MDE) [24] and allow to automate several aspects of software development. Therefore, it is crucial that model transformations are correct and repeatable to support incremental development and maintenance of high quality software. Consequently, model transformation languages, like programming languages, require an unambiguous semantics as a reference to enable to verify the outcome considering the model transformation specification (cf. [9]) and to ensure that different implementations result in the same outcome. In addition, an unambiguous formal

[*] This work was developed in the course of the project - Correct Model Transformations - Hasso Plattner Institut, Universität Potsdam and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft. See http://www.hpi.uni-potsdam.de/giese/projekte/kormoran.html?L=1.

semantics and clear understanding how the relational specification is operationalized can help to identify which optimizations are really the most appropriate ones. However, for existing relational model transformation approaches, it is usually not really clear under which constraints particular implementations are really conform to the formal semantics, or even more, how to statically check these constraints [25].

We will consider this challenge for the specific case of triple graph grammars (TGG) [23], which have a well understood formal semantics and are quite similar to other relational approaches such as QVT Relational (c.f. [13]). For TGGs, there are different tools, which realize slightly different dialects, such as Fujaba TGG Engine [3], MOFLON [1], or ATOM3 [19]. Furthermore, even for a single tool holds that different tool versions with different optimizations exist. For the Fujaba TGG Engine, there further exists a batch version with support for incremental synchronization [12], a version optimized for synchronizing multiple updates [10], and a version that further improves the runtime for synchronization [11] and can also be employed, for example, for runtime monitoring [27].

In this paper, we bridge the gap between the formal semantics of triple graph grammars [22] and our related efficient implementation [11]. The formal semantics assumes backtracking and ignores non-determinism and, thus, cannot be used to build an efficient implementation. Practical implementations for TGGs in contrast do not support backtracking, require rule sets that ensure determinism, and include further optimizations in order to ensure an efficient solution. Closing this gap for this example is of general interest, as also for other TGG implementations (e.g., [16], [1]) as well as other relational model transformation techniques (e.g., ATL[15], QVT[20]) a similar gap exists and no result that closes any of these gaps exists yet. Therefore, the outlined approach could serve as a scheme to also close this gap for these other cases.

To close the gap, we characterize exactly the subset of TGGs that the implementation assumes, demonstrate that the operationalization for the practical implementation and the semantics are conform for the identified subset, that it ensures determinism, and outline how static analysis can be employed to check these criteria defining the subset.

Therefore, we provide a first operationalization and then step by step eliminate assumptions such as backtracking while adding constraints that the TGG rules have to fulfill to permit their proper and efficient operationalization:

– As a starting point we explain and formally define the formal semantics of TGGs and the related forward and backward transformations named *relational scheme*.
– At first, we derive a naive operationalization for TGGs in form of the *conform scheme* that employs backtracking and bookkeeping for which we can show conformance to the formal semantics by demonstrating consistency – each transformation result of the implementation must fit to the semantics – and completeness – all possible transformations for the semantics are also covered by the operationalization.

– Then, we define a *deterministic scheme* via suitable criteria for determinism and show that for these criteria the operational rules can guarantee a deterministic result of the transformation. We thus can exclude non-determinism, which is not ruled out by the original TGG semantics but necessary as in practice a model transformation must be a function. Furthermore, we can still show conformance to the formal semantics via the beforehand introduced operationalization.
– Finally, there is the limitation that the considered TGG implementation only employs a bookkeeping approach for nodes but not edges. Again we can define a related *implementation scheme* via adjusting the rules and criteria that if fulfilled guarantee that also this scheme conforms to the formal semantics referring to the beforehand introduced operationalization.

It is to be noted that we stop the mapping at the level of the standard graph transformation semantics and omit several additional optimization tricks employed in the implementation that go beyond this abstraction. The implementation in particular avoids searching for matches in the whole source graph and makes use of control flow constructs of the employed underlying graph transformation language Story Diagrams [8].

The remaining paper is structured as follows: We first introduce TGGs as a relational specification scheme for model transformations in Section 2. Then, we outline an operational computation scheme and operational rules, and prove conformance to the formal semantics (Section 3). This scheme is further refined toward a deterministic computation scheme in Section 4, where proper restrictions for the rules are introduced that ensure determinism. Then, we introduce minor derivations in the computation scheme in Section 5 that hold for the implementation, show how the corresponding restrictions for the rules can be checked statically, and prove that conformance and determinism also hold for the implementation. Finally, we discuss related work (Section 6) and close the paper with our conclusions and an outlook on planned future work.

2 Relational Scheme

Triple graph grammars relate three different models: A source model, a target model, and a correspondence model that stores correspondence relationships between model fragments built from source and target elements. We use the meta models shown in Fig. 1 to illustrate the following explanations. It shows the meta model of simple block[1] and class diagrams. The elements of both models can be connected to nodes of the correspondence model.

Formally, the models can be interpreted as typed attributed graphs according to attributed type graphs. In case of TGGs, the type graph adheres to a particular structure reflecting its three components. A *triple graph* SCT[2] is a graph typed over

[1] This is a very simplified version of SDL [14] block diagrams.
[2] We use a triple of variables SCT to denote one triple graph.

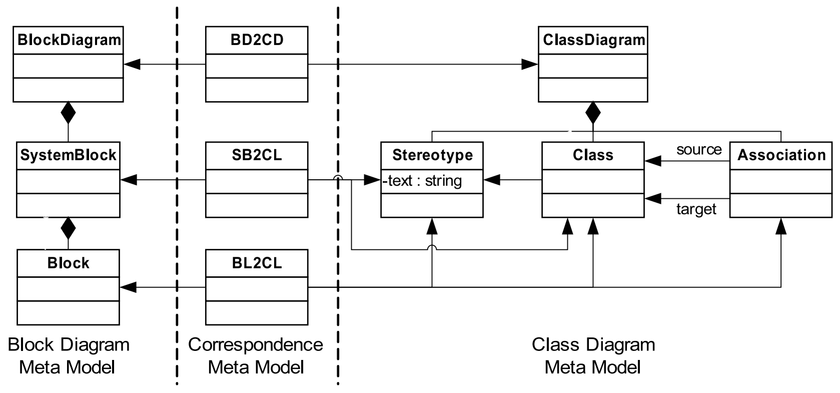


Fig. 1. Meta models of block diagrams, class diagrams, and correspondence models

TRIPLE:
$$\begin{tikzcd}
s \arrow[loop left, "l_s"] & c \arrow[l, "e_{cs}"'] \arrow[r, "e_{ct}"] & t \arrow[loop right, "l_t"]
\end{tikzcd}$$

Therefore, a triple graph SCT consists of a *source component* S, containing all elements of type s and l_s; a *correspondence component* C, containing all elements of type s, t, e_{cs},e_{ct}, and c; and a *target component* T, containing all elements of type t and l_t. A *triple type graph* $S_T C_T T_T$ is a special triple graph defining node and edge types for the source component, correspondence component, and target component of triple graphs. The meta models of Fig. 1 can be interpreted as a *triple type graph.* A *typed triple graph* is a triple graph typed over $S_T C_T T_T$, e.g. concrete block and class diagrams connected by a correspondence model like shown in Fig. 3. We say that a finite graph S (T or C) typed over S_T (T_T or C_T) is a source graph (target graph or correspondence graph, respectively) and belongs to the language $\mathcal{L}(S_T)$ ($\mathcal{L}(T_T)$ or $\mathcal{L}(C_T)$, respectively).

A triple graph grammar consists of an axiom (the grammar's start graph) and several TGG rules. An example TGG for the transformation of block and class diagrams is shown in Fig. 2[3]. Here, we use a short notation that combines the left-hand (LHS) and right-hand sides (RHS) of the graph transformation rule. Elements that belong to the LHS and RHS are drawn black, elements that belong only to the RHS (i.e. which are created by the rule) are drawn green and marked with "++". TGG rules are divided into three domains: The source model domain (left), target model domain (right), and the correspondence model domain (middle). The axiom in Fig. 2 creates the root elements of the three models and relates them to each other. Rule 1 creates a *SystemBlock* and a corresponding *Class*. The *BlockDiagram* and *ClassDiagram* must already exist. Rule 2 creates a *Block* in the block diagram domain and connects it to the *SystemBlock*. In the class diagram domain, a class is created and connected to the *SystemBlock*'s *Class* with an *Association.*

[3] Note, that the types defined in Fig. 1 are abbreviated in Fig. 2.

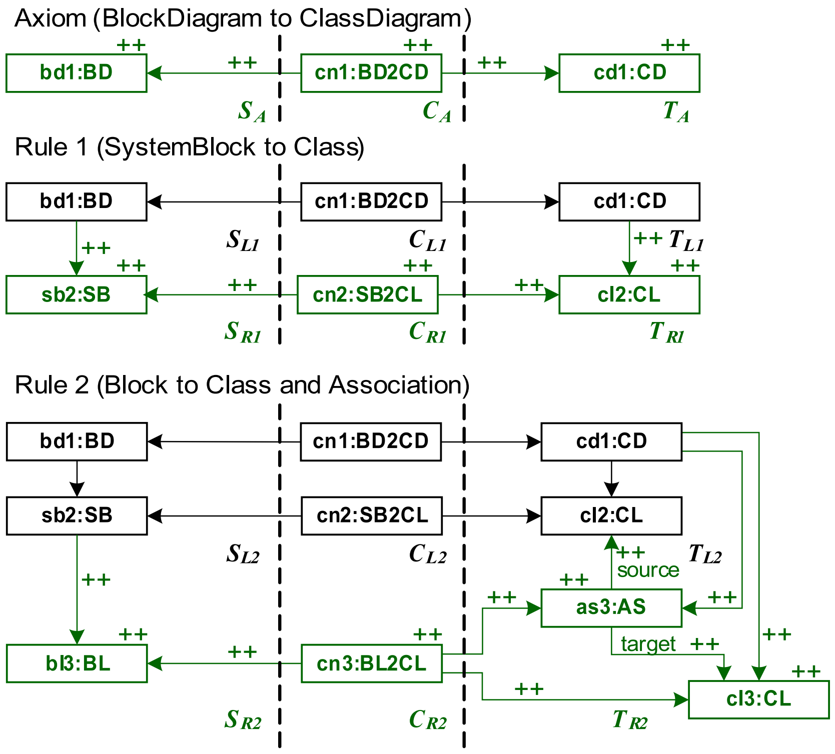


Fig. 2. Example TGG for the transformation of block and class diagrams

Formally, a *triple graph grammar* $TGG = (S_AC_AT_A, \mathcal{R})$ consists of an axiom $S_AC_AT_A$ and a set of non-deleting rules $\mathcal{R}$ for triple graphs. Each rule consists of an inclusion from the LHS $S_LC_LT_L$ to the RHS $S_RC_RT_R$ of the rule. Each element in $S_LC_LT_L$ corresponds to an element in $S_RC_RT_R$ and is preserved by the rule. Attribute values of preserved elements are not changed. All elements in $S_RC_RT_R \setminus S_LC_LT_L$ are created when the rule is applied. A rule r can be applied to a triple graph $S_GC_GT_G$ if a match m of its LHS $S_LC_LT_L$ into $S_GC_GT_G$ can be found. The result $S_HC_HT_H$ of the rule application $S_GC_GT_G \overset{m,r}{\rightarrow} S_HC_HT_H$ consists of the gluing of $S_GC_GT_G$ with elements in $S_RC_RT_R \setminus S_LC_LT_L$ via m. This means that all elements created by r are added to $S_GC_GT_G$. We say that $\mathcal{L}(TGG)$ is the set of all triple graphs that can be derived from $S_AC_AT_A$ using rule applications via rules in $\mathcal{R}$. Thereby, $\rightarrow^*_{TGG}$ denotes the reflexive and transitive closure of a rule application via some rule in TGG. Note, that the standard results on graph transformation such as the Concurrency Theorem, Local-Church

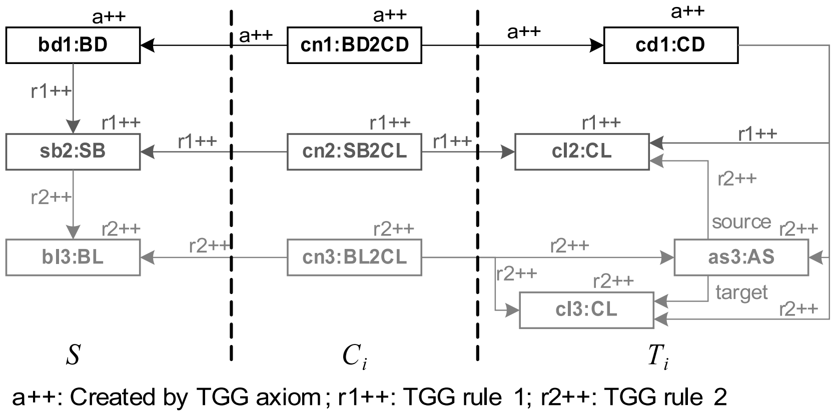


Fig. 3. Example block and class diagrams connected by a correspondence model

Rosser Theorem, and Critical Pair Lemma[4, 18] hold for typed triple graph transformations[4].

In general, three kinds of transformations can be performed with TGGs: Forward, backward, and correspondence transformations. A forward (backward) transformation takes a source (target) model as input and creates the correspondence and target (source) model. A correspondence transformation requires a source and target model and creates only the correspondence model. Subsequently, we focus on forward transformations. Analogous results can be derived for the backward and correspondence case straightforwardly. The formal definition of a relational forward transformation is as follows:

Definition 1 (relational scheme: FT_{TGG}). $FT_{TGG} : \mathcal{L}(S_T) \rightarrow \mathcal{P}(\mathcal{L}(TGG))$ *is defined as follows:* $FT_{TGG}(S) := \{SC_iT_i | S_AC_AT_A \rightarrow^*_{TGG} SC_iT_i\}$.

Fig. 3 shows a block diagram S with a corresponding class diagram T_i. Both models are connected by a correspondence model C_i. Assuming that we want to transform S, this is a valid forward transformation result in $FT_{TGG}(S)$ because there is a corresponding transformation producing a triple graph SC_iT_i according to the TGG in Fig. 2. The annotations in Fig. 3 indicate which rules create the corresponding elements. Because of the characteristics of the TGG $FT_{TGG}(S)$ contains only one element. However, this does not need to be the case in general, as we will demonstrate later on.

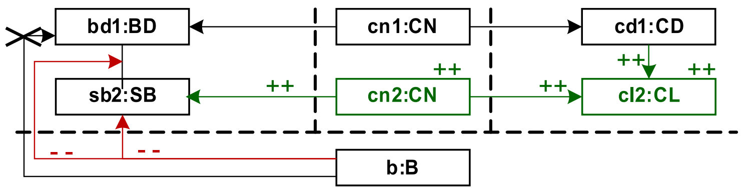


Fig. 4. Operational forward rule r_1^{BF} derived from rule r_1

3 Conform Scheme

We can derive operational transformation rules from the relational TGG rules. For the forward transformation, all elements belonging to the source domain are added to the LHS of the rule. More formally, given a triple graph rule $r : S_LC_LT_L \to S_RC_RT_R$ a forward rule $r^F : S_RC_LT_L \to S_RC_RT_R$ is derived.

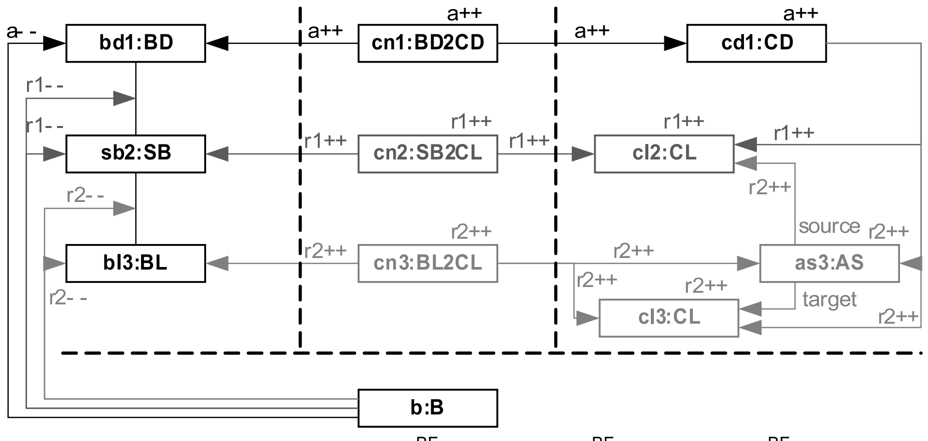
Furthermore, we use an operational transformation that keeps track of the elements that are not yet transformed. For this purpose, we introduce a bookkeeping node b, which holds bookkeeping edges to all source model nodes and edges that are not yet transformed[5]. This implies that the bookkeeping edges have to be created once before the transformation. Fig. 4 shows the operational bookkeeping forward rule derived from rule 1 in Fig. 2. When the operational rule is applied, the bookkeeping edges are deleted. This is indicated by the "**- -**" annotation and the red color of these edges. The model elements of S_L must have been transformed already. This is ensured by negative application conditions, which prohibit bookkeeping edges to these elements.

Formally, r^F is extended[6] to $r^{BF} : B_{S_R \setminus S_L} S_RC_LT_L \leftarrow bS_RC_LT_L \to bS_RC_RT_R$, a span of inclusions with NAC^{BF} defined as follows: We add a special bookkeeping node b, which is preserved. For each node and edge $x \in S_R \setminus S_L$ we add a bookkeeping edge from b to x, which is deleted by the rule. The set $B_{S_R \setminus S_L}$ consists of these bookkeeping edges and the bookkeeping node b. NAC^{BF} is a set of negative application conditions, which forbid for each element $x \in S_L$ an incoming bookkeeping edge from b. In the example in Fig. 4, $B_{S_R \setminus S_L}$ contains b, the bookkeeping edges to *sb2*, and the link between *bd1* and *sb2*. NAC^{BF} forbids the edge from b to *bd1*. We can interpret the axiom $S_AC_AT_A$ of a TGG as a triple graph rule, called *axiom rule*, $a : \emptyset \to S_AC_AT_A$. The bookkeeping forward rule a^{BF} of a is built analogously to regular TGG rules. However, the bookkeeping is not sufficient to exclude, that it is applied multiple times. Thus,

[4] As explained in more detail in Appendix A, we can define the category $\mathbf{ATripleGraphs}_{S_TC_TT_T}$, having typed attributed triple graphs as objects and typed attributed triple graph morphisms as arrows, such that $\mathbf{ATripleGraphs}_{S_TC_TT_T}$ with the set $\mathcal{M}$ of injective morphisms is an adhesive HLR category [4, 18].

[5] Note, that we use a graph model, where edges from nodes to edges are allowed, as explained more in detail in Appendix B.

[6] We therefore extend also the graph $TRIPLE$ and $S_TC_TT_T$ with corresponding bookkeeping node and edge types as presented in Appendix B.



a++/a- -: Created/deleted by a^{BF}; r1++/r1- -: r_1^{BF}; r2++/r2- -: r_2^{BF}

Fig. 5. Forward transformation of a block diagram.

this axiom rule must be controlled separately at the beginning of each forward transformation.

We can apply a bookkeeping forward rule r^{BF} to a triple graph with bookkeeping $B_G S_G C_G T_G$ if there exists a match $m : B_{S_R \setminus S_L} S_R C_L T_L \to B_G S_G C_G T_G$, fulfilling NAC^{BF}, meaning that each node and edge in $m(S_L)$ has *no* incoming edge from b (translated already), and each node and edge in $m(S_R \setminus S_L)$ has an incoming edge from b (to be translated). The application of rule r^{BF} to $B_G S_G C_G T_G$ via m deletes all bookkeeping edges in $m(B_{S_R \setminus S_L})$ and adds the translation $S_R C_R T_R \setminus S_R C_L T_L$. Fig. 5 shows the forward transformation of the example block diagram. Initially, there is a link from the bookkeeping node b to each source model node and edge. The bookkeeping forward axiom rule transforms *bd1* and produces *cn1* and *cd1*. The bookkeeping edge to *bd1* is deleted. Next, bookkeeping forward rule r_1^{BF} is applied to create *cn2* and *cl2*. It also deletes the bookkeeping edges to *sb2* and to the link between *bd1* and *sb2*. Finally, bookkeeping forward rule r_2^{BF} is applied, analogously.

In order to formally define an operational scheme FT_{CON}, we define $CON = OP_{FT}(TGG)$ as the rule set consisting of a bookkeeping forward rule r^{BF} for each rule r of the TGG. Moreover, we have a mapping $Trans^F$ computing for each triple graph with bookkeeping $BSCT$: $Trans^F(BSCT)$ the part of S that has already been translated by some bookkeeping forward rule. In particular, it consists of all nodes and edges with no incoming bookkeeping edges. Finally, we define $B_{init}S$ as the initial source graph consisting of S and b equipped with one bookkeeping edge for each graph element in S.

Definition 2 (conform scheme: FT_{CON}). *Given a TGG and its operationalization CON, then* $FT_{CON} : \mathcal{L}(S_T) \to \mathcal{P}(\mathcal{L}(TGG))$ *of a source graph S is*

defined as follows, $FT_{CON}(S) :=$
$\{SC_iT_i | B_{init}S \rightarrow_{a^{BF}} B_ASC_AT_A \rightarrow^*_{CON} B_iSC_iT_i \wedge S = Trans^F(B_iSC_iT_i)\}$.

This implies that all elements in S have been transformed exactly once. We call this a valid transformation. In the following, we show that $FT_{TGG}(S)$ and $FT_{CON}(S)$ deliver the same set of valid transformation results (consistency and completeness). In particular, we show that for each forward transformation via bookkeeping rules a corresponding forward transformation according to the TGG exists (consistency) and the other way round (completeness).

In [22, 5], it is argued already in detail that a TGG rule application can be decomposed into a sequence of transformations via the corresponding source rule[7] followed by a transformation via the corresponding forward rule and the other way round (composition), where the correspondence and target component of the rule are empty. Since our forward and source rules are constructed analogously to [22, 5], in the following proof ideas we assume these results and concentrate on arguing that the bookkeeping mechanism as added in this paper to the forward rules leads to consistency and completeness as described above. Complete proofs can be found in Appendix C.

As an auxiliary result, we show that each application of bookkeeping forward rules is backed up by a corresponding TGG rule application.

Lemma 1 (partial consistency). *For a TGG and its operationalization* $CON = OP_{FT}(TGG)$ *holds that* $B_{init}S \rightarrow_{a^{BF}} B_ASC_AT_A \rightarrow^*_{CON} B_iSC_iT_i \wedge Trans^F(B_iSC_iT_i) = S_i$ *implies* $S_AC_AT_A \rightarrow^*_{TGG} S_iC_iT_i$ *via the related TGG rules.*

Proof. (Proof idea) The bookkeeping forward axiom rule a^{BF} *is applied to* $B_{init}S$ *via some match* m_A *conform to* $S_AC_AT_A$ *only once. The set of translated elements after this first step* $Trans^F(B_ASC_AT_A) = m_A(S_A)$. *Furthermore, bookkeeping of the rules in* CON *implies that during the transformation of* $B_ASC_AT_A$ *each node and edge of* $S \backslash m_A(S_A)$ *is translated at most once conform to the corresponding TGG rules. Each rule application of* CON *via some rule* r^{BF} *enlarges the set of translated elements in* S *with the matched elements of* $S_R \setminus S_L$. *Accordingly, when a series of rule applications via* CON *starting with* $B_ASC_AT_A$ *delivers* $B_iSC_iT_i$ *such that* $Trans^F(B_iSC_iT_i) = S_i$, *then applying the related TGG rules generates* $S_iC_iT_i$ *from* $S_AC_AT_A$. □

Theorem 1 (conformance). *For a TGG and its operationalization* $CON = OP_{FT}(TGG)$, *it holds that* $FT_{TGG}(S) = FT_{CON}(S)$. *In particular,* $B_{init}S \rightarrow_{a^{BF}} B_ASC_AT_A \rightarrow^*_{CON} B_iSC_iT_i$ *and* $Trans^F(B_iSC_iT_i) = S$ *if and only if* $S_AC_AT_A \rightarrow^*_{TGG} SC_iT_i$ *via the related TGG rules.*

Proof. (Proof idea) $FT_{CON}(S) \subseteq FT_{TGG}(S)$ *(consistency) follows from Lemma 1 for the special case that* $Trans^F(B_iSC_iT_i) = S$.

[7] Given a TGG rule $r : S_LC_LT_L \rightarrow S_RC_RT_R$, then we have the following corresponding source rule $r^S : S_L \rightarrow S_R$.

$FT_{CON}(S) \supseteq FT_{TGG}(S)$ (completeness) holds because of the following argumentation: The forward axiom rule a^{BF} can be applied to $B_{init}S$ such that $B_{init}S \rightarrow_{a^{BF}} B_A S C_A T_A$ since the source axiom S_A is contained in S. Moreover, for each TGG rule application via r generating the graph elements $S_R \setminus S_L$ in S, the related bookkeeping forward rule r^{BF} of CON can be applied, translating exactly those elements in S conform to r. Since each element in S, except the axiom elements, is generated by such a TGG rule application, we have that $B_{init}S \rightarrow_{a^{BF}} B_A S C_A T_A \rightarrow^*_{CON} B_i S C_i T_i$ such that $Trans^F(B_i S C_i T_i) = S$. $\square$

4 Deterministic Scheme

It is not guaranteed that whenever a valid transformation result exists, it can be found without backtracking. Valid means that the complete source graph has been covered by the transformation. The determinism criteria studied in this section restrict the TGGs to those ones where backtracking can be safely avoided. These criteria ensure, on one hand, that whenever a valid transformation result exists, it can be found without backtracking. On the other hand, if no valid transformation result exists, then we can find this out without backtracking, as well.

In order to avoid backtracking, we need to show that applying bookkeeping forward rules as long as possible always terminates with a unique result. To this extent, we use the theory of critical pairs guaranteeing that under specific conditions a set of bookkeeping forward rules is locally confluent [4, 17, 18]. A *critical pair* describes a conflict in a minimal context. Conflicts arise for bookkeeping forward rules if one rule deletes a bookkeeping edge marked for deletion also by the other rule[8]. This is because after applying the first rule and deleting the bookkeeping edge which is marked for deletion also by the other rule, this rule cannot be applied anymore. Note that we ignore critical pairs with same rules and same matches, since they represent a confluent situation in a trivial way. Moreover, we introduce a termination criterion ensuring that each application of a bookkeeping forward rule diminishes the number of translated elements indeed.

Definition 3 (determinism criteria). *The forward determinism criteria for a TGG and its operationalization $CON = OP_{FT}(TGG)$ are defined as follows:*

- *each TGG rule creates at least one graph element on the source part (termination criterion)*
- *for the rules in $CON = OP_{FT}(TGG)$ there exist no critical pairs, ignoring pairs with same rules and same matches (conflict-freeness criterion)*

The following Theorem guarantees that we can define a scheme without backtracking for TGGs fulfilling the above determinism criteria. Thereby, we only

[8] Note that neither produce-forbid conflicts can occur, since no bookkeeping edges are produced, nor attribute conflicts can occur, since attributes are only written if the corresponding node is created.

translate source graphs belonging to $\mathcal{L}(S_T^A) \subseteq \mathcal{L}(S_T)$, containing the source component of the TGG axiom only once (*domain restriction criterion*). This is because in order to obtain a unique transformation result, the elements from which the translation should be started should be fixed upfront uniquely.

Theorem 2 (*FT_{CON}* forward deterministic). *For a TGG and its operationalization $CON = OP_{FT}(TGG)$ fulfilling the forward determinism criteria of Definition 3, it holds that for each $S \in \mathcal{L}(S_T^A)$ either some SCT exists such that $FT_{CON}(S) = \{SCT\}$ or $FT_{CON}(S) = \emptyset$. We say that FT_{CON} is forward deterministic.*

Proof. In case that no transformation of S exists such that all elements can be translated, $FT_{CON}(S)$ is empty.

Suppose that $FT_{CON}(S)$ is not empty and that SCT belongs to $FT_{CON}(S)$. Then, we have that $B_{init}S \rightarrow_{a^{BF}} B_ASC_AT_A \rightarrow^*_{CON} BSCT$ such that $Trans^F(BSCT) = S$. It follows that SCT is the only element belonging to $FT_{CON}(S)$ because of the following argumentation: Since $S \in \mathcal{L}(S_T^A)$, the match of the forward axiom rule a^{BF} is uniquely fixed. Because of the termination criterion in Definition 3, it holds that each bookkeeping forward rule deletes at least one bookkeeping edge to a source element in S. Because bookkeeping forward rules are not producing any source elements and S is finite, this means that applying bookkeeping forward rules as long as possible always terminates. Moreover, it follows from the Critical Pair Lemma in [4, 17, 18] and the conflict-freeness criterion in Definition 3 that CON is locally confluent. In particular, if there are no critical pairs, ignoring pairs with same rules and same matches, then we can conclude that for each pair of transformations $H_1 \overset{r_1,m_1}{\Leftarrow} G \overset{r_2,m_2}{\Rightarrow} H_2$ either $H_1 \cong H_2$ or there exist transformations $H_1 \overset{r_2,m_1'}{\Rightarrow} X \overset{r_1,m_2'}{\Leftarrow} H_2$.

Together with termination this means that $\rightarrow^*_{CON}$ is confluent and thus, the application of rules in $\rightarrow^*_{CON}$ as long as possible terminates with a unique result. If all elements in S have been translated, no rule in CON is applicable anymore, since the application of any other bookkeeping rule would need at least one non-translated element (see termination criterion in Def. 3). Therefore, the result BSCT is a terminal state, which is unique such that $FT_{CON}(S) = \{SCT\}$. □

Note that the conflict-freeness criterion could be relaxed by allowing for critical pairs that are strictly NAC-confluent [17, 18]. Since we want to provide feasible practical tool support for our approach, we have opted however for the more severe conflict-freeness criterion. Currently, there is no tool support for computing if critical pairs are strictly NAC-confluent. Implementing such an algorithm would involve exponential complexity with respect to the depth of the search tree. Therefore, we rather plan to allow for adding priorities to rules in case of conflicts, leading to determinism in an alternative way, which seems at the same time also practically feasible.

Because of Theorem 2 it is now possible to define a deterministic scheme FT_{DET}. Since uniqueness of the transformation result is guaranteed, this scheme works *without backtracking*.

Definition 4 (deterministic scheme: FT_{DET}). *Given a TGG and its operationalization $CON = OP_{FT}(TGG)$ fulfilling the forward determinism criteria of Definition 3, then the deterministic bookkeeping forward transformation $FT_{DET} : \mathcal{L}(S_T^A) \to \mathcal{L}(TGG)$ is a partial mapping such that $FT_{DET}(S) := SCT$ if $FT_{CON}(S) = \{SCT\}$, else $FT_{DET}(S)$ is undefined.*

Note that for the subset of TGGs fulfilling the forward determinism criteria of Definition 3 and the deterministic scheme FT_{DET}, conformance with the TGG still holds, since it is based on the operational rules in CON.

5 Implementation Scheme

Our implementation [12] is based on the Eclipse Modeling Framework[9] (EMF). Currently, this implementation only provides bookkeeping on nodes and does not provide bookkeeping for edges, because edges do not have an identity in EMF-based models[10]. In the following, we define specific criteria for our implementation with node bookkeeping only, such that conformance with the TGG and determinism is still ensured.

The operational rules for the implementation, IMP, are analogous to the bookkeeping forward rules in CON, apart from the fact that the bookkeeping for edges is omitted. Given a TGG rule r, we therefore define a node bookkeeping rule r^{IF}. First, let us assume that S_{R_N} and S_{L_N} denotes the set of nodes in S_R and S_L, respectively. Given a TGG rule r, then a node bookkeeping forward rule $r^{IF} : B^N_{S_{R_N} \setminus S_{L_N}} S_R C_L T_L \leftarrow b S_R C_L T_L \rightarrow b S_R C_R T_R$ is a span of inclusions, deleting for each node in $S_{R_N} \setminus S_{L_N}$ the corresponding bookkeeping edges, together with NAC^{IF} a set of NACs forbidding for each node n in S_L an incoming bookkeeping edge from b, expressing that the node has been translated already. For example, r_1^{IF} is equal to r_1^{BF} (see Fig. 4) apart from the bookkeeping edge to the edge between $bd1$ and $sb2$. Given a TGG, then $IMP = OP_{FT}^{IMP}(TGG)$ is the rule set consisting of a node bookkeeping forward rule r^{IF} for each rule r of the TGG. For the axiom rule a, we have the node bookkeeping forward axiom rule a^{IF}. Because we now do only bookkeeping on nodes but not on edges, we adapt the forward determinism criteria as given in Section 4.

Definition 5 (forward implementation criteria). *The forward implementation criteria for a TGG and its operationalization $IMP = OP_{FT}^{IMP}(TGG)$ are defined as follows:*

- *each TGG rule creates at least one graph node on the source part (refined termination criterion)*
- *for the rules in IMP there exist no critical pairs, ignoring pairs with same rules and same matches (conflict-freeness criterion)*

[9] http://www.eclipse.org/modeling/emf/

[10] To provide edge bookkeeping, some kind of helper structure would be required.

The mapping $Trans^{FN}$ computes for each triple graph with node bookkeeping $B^N SCT$ the nodes that have been translated by some bookkeeping forward rule already: $Trans^{FN}(B^N SCT)$ is a subgraph of S, consisting of all nodes with no incoming bookkeeping edge. We define $B^N_{init}S$ as the initial source graph, consisting of S and b equipped with one bookkeeping edge for each graph node in S. Given a TGG and its operationalization $IMP = OP^{IMP}_{FT}(TGG)$ fulfilling the implementation criteria of Definition 5, then we can define $FT_{IMP} : \mathcal{L}(S_T) \to \mathcal{P}(\mathcal{L}(TGG))$ as follows: $FT_{IMP}(S) := \{SC_iT_i | B^N_{init}S \to_{a^{IF}} B^N_A SC_A T_A \to^*_{IMP} B^N_i SC_iT_i \wedge S_N = Trans^{FN}(B^N_i SC_iT_i)\}$, where S_N is the set of all nodes in S. We prove that FT_{IMP} is forward deterministic if it fulfills the forward implementation criteria and domain restriction criterion. Then, we can define the implementation scheme FT_{IMPD} that works without backtracking.

Theorem 3 (FT_{IMP} forward deterministic). *For a TGG and its operationalization $IMP = OP^{IMP}_{FT}(TGG)$ fulfilling the forward implementation criteria of Definition 5, it holds that for each $S \in \mathcal{L}(S^A_T)$ either some SCT exists such that $FT_{IMP}(S) = \{SCT\}$ or $FT_{IMP}(S) = \emptyset$. We say that FT_{IMP} is forward deterministic.*

Proof. In case that no transformation via node bookkeeping rules exists such that all nodes of S can be translated, $FT_{IMP}(S)$ is empty.

Suppose that $FT_{IMP}(S)$ is not empty and that SCT belongs to $FT_{IMP}(S)$. Then, we have that $B^N_{init}S \to_{a^{IF}} B^N_A SC_A T_A \to^*_{IMP} B^N SCT$ such that $S_N = Trans^{FN}(B^N SCT)$. It follows that SCT is the only element belonging to $FT_{IMP}(S)$ because of the following argumentation. Recall that since $S \in \mathcal{L}(S^A_T)$, the way to match the forward axiom rule a^{IF} is uniquely fixed. Moreover, analogous to the proof of Theorem 2, it follows that the application of node bookkeeping forward rules, fulfilling the forward implementation criteria, as long as possible terminates with a unique result. Thereby note that rules in IMP only delete bookkeeping edges to nodes (not to edges) and therefore, we need the refined termination criterion of Def. 5. If all nodes in S have been translated, no rule in IMP is applicable anymore, since the application of any other node bookkeeping rule would need at least one non-translated node (again because of the refined termination criterion in Def. 5). Therefore, the result $B^N SCT$ is a terminal state, which is unique such that $FT_{IMP}(S) = \{SCT\}$. □

Definition 6 (implementation scheme: FT_{IMPD}). *Given a TGG and its operationalization $IMP = OP^{IMP}_{FT}(TGG)$ fulfilling the forward implementation criteria of Definition 5, then $FT_{IMPD} : \mathcal{L}(S^A_T) \to \mathcal{L}(TGG)$ is a partial mapping such that $FT_{IMPD}(S) := SCT$ if $FT_{IMP}(S) = \{SCT\}$, else $FT_{IMPD}(S)$ is undefined.*

For valid source models, we can prove conformance of FT_{IMPD} with the TGG. A source model $S \in \mathcal{L}(S^A_T)$ is *valid* if a triple graph $SCT \in \mathcal{L}(TGG)$ with source component S exists.

Theorem 4 (FT_{IMPD} conform with FT_{TGG}). *Given a TGG with operationalization $IMP = OP^{IMP}_{FT}(TGG)$ fulfilling the forward implementation cri-*

teria and some valid $S \in \mathcal{L}(S_T^A)$*, it holds that* $\{FT_{IMPD}(S)\} = FT_{IMP}(S) = FT_{TGG}(S)$*.*

Proof. Because each rule application via r^{BF} *doing bookkeeping on nodes and edges implies a rule application via* r^{IF}*, where bookkeeping on edges is disregarded, we can conclude that* $FT_{CON}(S) \subseteq FT_{IMP}(S)$*.*

Moreover, we can prove that $FT_{CON}(S) \supseteq FT_{IMP}(S)$*. Suppose that SCT belongs to* $FT_{IMP}(S)$ *and therefore,* $B^N SCT$ *exists such that* $S_N = Trans^{FN}(B^N SCT)$*. We know by assumption that S is valid and therefore, it follows that there exists* $SC_*T_* \in \mathcal{L}(TGG)$ *and consequently,* $SC_*T_* \in FT_{TGG}(S)$*. Then, it follows from completeness of* FT_{CON} *that* $SC_*T_* \in FT_{CON}(S)$*. Therefore, there exists* $B_*SC_*T_*$ *such that* $Trans^F(B_*SC_*T_*) = S$*. Since* $FT_{CON}(S) \subseteq FT_{IMP}(S)$*, it follows that* $SC_*T_* \in FT_{IMP}(S)$ *with* $Trans^{FN}(B_*^N SC_*T_*) = S_N$*. Because of determinism of* IMP*, it follows that* $SCT = SC_*T_*$*. Therefore, it follows that there exists* $B_*SC_*T_* = B_*SCT$ *such that* $S = Trans^F(B_*SC_*T_*) = Trans^F(B_*SCT)$*. Consequently, SCT belongs to* $FT_{CON}(S)$*.*

Concluding, $FT_{IMP}(S) = FT_{CON}(S)$ *and because of conformance of* FT_{CON} *with* FT_{TGG} *also* $FT_{IMP}(S) = FT_{TGG}(S)$*. Since S is valid,* $FT_{TGG}(S)$ *is not empty and therefore* $FT_{IMP}(S) = \{FT_{IMPD}(S)\}$*.* □

Note that only with the restriction that S is valid, we can conclude conformance of FT_{IMPD} with the TGG. It would be better to have an implementation, which checks this while transforming S. The scheme FT_{DET}, with bookkeeping also for edges, provides such a solution. Therefore, it is currently used to realize an implementation providing this feature.

When we analyze the forward transformation of our example TGG, we see that the forward axiom rule in Fig. 2 can only be matched in a unique way to S, so there is no need to fix it uniquely (initialization criterion). Moreover, it is obvious that each TGG rule in Fig. 2 creates at least one source node (refined termination criterion). For the conflict-freeness criterion, we need to compute critical pairs for the node bookkeeping forward rules r_1^{IF} and r_2^{IF}. We use AGG[11] [26] to check that the node bookkeeping forward rules IMP of our example are conflict-free. In particular, AGG computes in 0.8s[12] that indeed there exist no critical pairs.

Analyzing the backward transformation, it is easy to see that the refined termination criterion is fulfilled for the target components of the TGG rules. The conflict-freeness criterion is not fulfilled, since AGG computes a critical pair (8.4s) for the node bookkeeping backward rules r_1^{IB} and r_2^{IB}. Both rules compete to translate the same class, being target of an association. In particular, r_1^{IB} schedules it for translation into a system block and r_2^{IB} for translation into a block. Therefore, we have two cases: (a) if the encoding of the target cannot be changed, the backward transformation cannot be used. We can conclude that

[11] AGG does not provide the possibility to specify edges from nodes to edges. However, since we do critical pair analysis on rules in IMP, doing only bookkeeping on nodes, we do not need this possibility.

[12] Pentium Dual Core E5300 @ 2.60 GHz 2.60GHz, 4.00 GB RAM

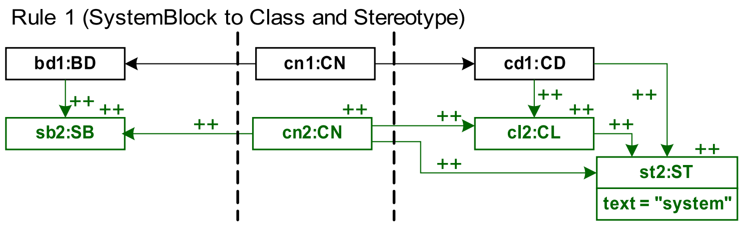


Fig. 6. Corrected TGG rule 1 with a stereotype

the criteria implicitly allow us to check thus whether the rule set is bidirectional or not. (b) We correct the encoding of the target if necessary and also the TGG rules in order to obtain backward determinism. In our example, this can be achieved by adding a stereotype in $r1$ to classes corresponding to system blocks and another stereotype to classes being target of an association as in rule $r2$. The *text* attribute of the stereotype is set to "system" and "block", respectively. Fig. 6 shows the corrected TGG rule $r1$. For these corrected backward rules, AGG computes in 10.5s that indeed there exist no critical pairs[13].

Concluding, suppose that in our implementation we would not be able to rely on determinism of the model transformation result. In this case, we would need to apply backtracking in order to find all possible transformation results. The complexity of such a backtracking algorithm would be exponential with respect to the depth of the search tree. On the contrary, if determinism can be assumed because it has been computed statically beforehand, then performing a model transformation becomes linear with respect to the depth of the search tree (abstracting from the complexity of matching rules, which would be comparable in both algorithms).

6 Related Work

In [22, 5] consistency and completeness of TGG forward rules is shown. However, checking consistency of a forward transformation may become very inefficient, because parsing is involved. Note that in [22, 5] triple graphs are defined as spans of injective morphisms[14], which does not allow to connect one correspondence node with more than one source or target element, respectively.

[13] Note that for computing critical pairs in AGG we interpret stereotypes with different text attributes as distinct types and use maximal multiplicities in the type graph ruling out critical pairs, describing conflicts that would never occur anyway. In [18], it is proven that ruling out critical pairs by maximal multiplicities does not affect completeness of critical pairs.

[14] On the contrary, we have chosen a formalization based on plain graphs typed over a suitable type triple graph, which suits better to our implementation. Moreover, we do not need a flattening construction (as in [5]) in order to be able to transfer theoretical concepts to the implementation level.

In [23] a bookkeeping mechanism for the operationalization of TGGs was proposed for which consistency could be shown, but completeness is not warranted. Based on the same idea, an approach is presented in [6], which checks consistency of a forward transformation on-the-fly, maintains completeness, but still involves parsing. In [7], from consistent forward transformations so-called terminating NAC-consistent forward ones are derived and checked for determinism. In this paper, conversely we argue that having a deterministic set of forward rules with integrated bookkeeping, then consistency follows. From a practical point of view, this implication direction is more interesting, because backtracking can be avoided. Summarizing, as far as we know, there is no other approach guaranteeing consistency, completeness, and determinism all at once for a specific subset of TGGs on a formal as well as implementation level.

Another approach to relational, bidirectional transformations with a formal basis is [21], where a terminating, correct, and complete operationalization for so-called patterns, some kind of graph constraints for triple graphs, is derived. Large sets of valid transformation results may occur, leading to efficiency problems concerning implementation.

QVT[20] is an OMG standard for bidirectional model transformations. The standard itself does not explicitly forbid non-deterministic rule sets. Stevens[25] reports about semantic issues of the QVT standard concerning bijectivity of bidirectional model transformations. The author also argues that the behavior of transformations needs to be deterministic. However, there are several possibilities to achieve deterministic transformations, for example rule priorities. The rules are executed in a specific order and the first rule that matches is applied. MediniQVT[15] uses another approach. It transforms an element multiple times if there is more than one rule that matches. The ATLAS Transformation Language [15] (ATL) is a widely used language for unidirectional model transformations. ATL does not allow conflicts among transformation rules. The ATL engine reports an error in that case, which can be considered a serious drawback, since the problem of conflict resolution is shifted from the designer to the user of the model transformation. We in contrast ensure determinism, independently of the order in which operational rules are applied, by restricting the set of valid TGG rules. This can be checked statically, which was pointed out as one of the open issues in [25].

7 Conclusion & Future Work

In this paper we have closed the gap between the formal semantics of TGG and our implementation. However, this does not only ensure that a valid rule set results in a unique and semantically correct outcome, it also permits to decide whether a TGG can be applied in both directions. It also links a practical implementation with a suitable formal semantics such that based on this sound foundation and former work [9, 2, 18] we can now study the verification

[15] http://projects.ikv.de/qvt/

of model transformations exploiting the identified criteria. The provided bridge only closes the gap between the formal semantics of TGGs and the implementation at the level of abstraction related to the standard graph transformation system semantics. In an additional step we plan to also cover several omitted additional optimization tricks employed in the implementation that go beyond this abstraction. These are in particular the strategy to avoid searching for matches in the whole source graph and the way control flow constructs are used in the implementation to realize the transformation on top of the graph transformation language Story Diagrams [8]. Finally, it is planned to also cover the sophisticated model synchronization schemes that have been developed [12, 10, 11] in the same manner to prove their correctness, define required constraints, and maybe also identify further potential for optimization.

References

1. Amelunxen, C., Klar, F., Königs, A., Rötschke, T., Schürr, A.: Metamodel-Based Tool Integration with MOFLON. In: ICSE '08: Proceedings of the 30th ICSE. pp. 807–810. ACM, New York, NY, USA (2008)
2. Becker, B., Giese, H.: Incremental Verification of Inductive Invariants for the Run-Time Evolution of Self-Adaptive Software-Intensive Systems. In: Proc. 23rd IEEE/ACM International Conference on Automated Software Engineering - Workshops. pp. 33–40. IEEE Computer Society Press (2008)
3. Burmester, S., Giese, H., Niere, J., Tichy, M., Wadsack, J.P., Wagner, R., Wendehals, L., Zündorf, A.: Tool Integration at the Meta-Model Level within the FUJABA Tool Suite. International Journal on Software Tools for Technology Transfer (STTT) 6(3), 203–218 (August 2004)
4. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs in Theoretical Computer Science, Springer (2006)
5. Ehrig, H., Ermel, C., Hermann, F.: On the Relationship of Model Transformations based on Triple and Plain Graph Grammars. In: GRaMoT '08: Proceedings of the third International Workshop on Graph and Model Transformations. pp. 9–16. ACM, New York, NY, USA (2008)
6. Ehrig, H., Ermel, C., Hermann, F., Prange, U.: On-the-Fly Construction, Correctness and Completeness of Model Transformations Based on Triple Graph Grammars. In: Proc. Models 2009 Model Driven Engineering Languages and Systems. LNCS, vol. 5795/2009, pp. 241–255. Springer Berlin / Heidelberg (2009)
7. Ehrig, H., Prange, U.: Formal Analysis of Model Transformations Based on Triple Graph Rules with Kernels. In: ICGT '08: Proceedings of the 4th International Conference on Graph Transformation. pp. 178–193. Springer-Verlag, Berlin, Heidelberg (2008)
8. Fischer, T., Niere, J., Torunski, L., Zündorf, A.: Story Diagrams: A New Graph Rewrite Language Based on the Unified Modeling Language and Java. In: TAGT'98: Selected papers from the 6th International Workshop on Theory and Application of Graph Transformations. Lecture Notes in Computer Science (LNCS), vol. 1764/2000, pp. 296–309. Springer-Verlag, London, UK (16-20 November 2000)
9. Giese, H., Glesner, S., Leitner, J., Schäfer, W., Wagner, R.: Towards Verified Model Transformations. In: Hearnden, D., Süß, J.G., Baudry, B., Rapin, N. (eds.) Proc.

of the 3^rd International Workshop on Model Development, Validation and Verification (MoDeVa), Genova, Italy. pp. 78–93. Le Commissariat à l'Energie Atomique - CEA (October 2006)

10. Giese, H., Hildebrandt, S.: Incremental Model Synchronization for Multiple Updates. In: Proceedings of the 3rd International Workshop on Graph and Model Transformations, May 12, 2008, Leipzig, Germany. pp. 1–8. ACM Press (2008)
11. Giese, H., Hildebrandt, S.: Efficient Model Synchronization of Large-Scale Models. Tech. Rep. 28, Hasso Plattner Institute at the University of Potsdam (2009)
12. Giese, H., Wagner, R.: From Model Transformation to Incremental Bidirectional Model Synchronization. Software and Systems Modeling (SoSyM) 8(1) (28 March 2009)
13. Greenyer, J., Kindler, E.: Comparing Relational Model Transformation Technologies: Implementing Query/View/Transformation with Triple Graph Grammars. Software and Systems Modeling 9(1), 21–46 (2010)
14. International Telecommunication Union, I.: ITU-T Recommendation Z.100: Specification and Description Language (SDL) (2002)
15. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Satellite Events at the MoDELS 2005 Conference. LNCS, vol. 3844, pp. 128–138. Springer Verlag, Berlin (2006)
16. Kindler, E., Rubin, V., Wagner, R.: An Adaptable TGG Interpreter for In-Memory Model Transformation. In: Schürr, A., Zündorf, A. (eds.) Proc. of the 2nd International Fujaba Days 2004, Darmstadt, Germany. Technical Report, vol. tr-ri-04-253, pp. 35–38. University of Paderborn (2004)
17. Lambers, L., Ehrig, H., Prange, U., Orejas, F.: Embedding and Confluence of Graph Transformations with Negative Application Conditions. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) Proc. International Conference on Graph Transformation (ICGT'08). LNCS, vol. 5214, pp. 162–177. Springer, Heidelberg (2008)
18. Lambers, L.: Certifying Rule-Based Models using Graph Transformation. Ph.D. thesis, Technische Universität Berlin (2010)
19. de Lara, J., Vangheluwe, H.: AToM3 as a Meta-CASE Environment (DFD to SC). In: Proceedings of the 4th International Conference on Enterprise Information Systems (2002)
20. Object Management Group: MOF 2.0 QVT 1.0 Specification (2008)
21. Orejas, F., Guerra, E., de Lara, J., Ehrig, H.: Correctness, Completeness and Termination of Pattern-Based Model-to-Model Transformation. In: Kurz, A., Lenisa, M., Tarlecki, A. (eds.) CALCO. LNCS, vol. 5728, pp. 383–397. Springer (2009)
22. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) Proc. of the 20^th International Workshop on Graph-Theoretic Concepts in Computer Science. LNCS, vol. 903, pp. 151–163. Spinger Verlag, Herrsching, Germany (June 1994)
23. Schürr, A., Klar, F.: 15 Years of Triple Graph Grammars : Research Challenges, New Contributions, Open Problems. In: 4th International Conference of Graph Transformation, ICGT 2008, Leicester, United Kingdom, September 7-13, 2008. LNCS, vol. 5214, pp. 411–425. Springer, Berlin / Heidelberg (2008)
24. Sendall, S., Kozaczynski, W.: Model Transformation: The Heart and Soul of Model-Driven Software Development. IEEE Software pp. 42–45 (2003)
25. Stevens, P.: Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions. Software and Systems Modeling 9(1), 7–20 (2010)

26. Taentzer, G., Ermel, C., Rudolf, M.: The AGG-Approach: Language and Tool Environment. In: Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G. (eds.) Handbook of Graph Grammars and Computing by Graph Transformation, volume 2: Applications, Languages and Tools. pp. 551–603. World Scientific (1999)
27. Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., Becker, B.: Incremental Model Synchronization for Efficient Run-Time Monitoring. In: Ghosh, S. (ed.) Models in Software Engineering, Workshops and Symposia at MODELS 2009, Denver, CO, USA, October 4-9, 2009, Reports and Revised Selected Papers, LNCS, vol. 6002, pp. 124–139. Springer-Verlag (April 2010)

A Triple Graphs and TGGs

We reintroduce the definition of graphs and graph morphisms.

Definition 7 (graph and graph morphism). *A* graph $G = (G_N, G_E, src, tgt)$ *consists of a set G_N of nodes, a set G_E of edges and two mappings $src, tgt : G_E \to G_N$, assigning to each edge $e \in G_E$ a source $src(e) \in G_N$ and target $tgt(e) \in G_N$. A* graph morphism $f : G_1 \to G_2$ *between two graphs $G_i = (G_{i,N}, G_{i,E}, src_i, tgt_i)$, $(i = 1, 2)$ is a pair $f = (f_N : G_{1,N} \to G_{2,N}, f_E : G_{1,E} \to G_{2,E})$ of mappings, such that $f_N \circ src_1 = src_2 \circ f_E$ and $f_N \circ tgt_1 = tgt_2 \circ f_E$.*

The category having graphs as objects and graph morphisms as arrows is called **Graphs**.

Triple graphs are graphs typed over a distinguished graph, called $TRIPLE$.

Definition 8 (triple graph and morphism). *The graph $TRIPLE$ is a graph with three nodes s, c, t and four edges l_s, e_{cs}, e_{ct} and l_t such that $src(l_s) = s$, $tgt(l_s) = s$, $src(e_{cs}) = s$, $tgt(e_{cs}) = c$, $src(e_{ct}) = c$, $tgt(e_{ct}) = t$ and $src(l_t) = t$, $tgt(l_t) = t$:*

$$\begin{tikzcd} TRIPLE: & s \arrow[loop left, "l_s"] & c \arrow[l, "e_{cs}"'] \arrow[r, "e_{ct}"] & t \arrow[loop right, "l_t"] \end{tikzcd}$$

A triple graph $(G, triple)$ *is a graph G equipped with a morphism $triple : G \to TRIPLE$. Consider triple graphs $(G_1, triple_1)$ and $(G_2, triple_2)$, a* triple graph morphism $f : (G_1, triple_1) \to (G_2, triple_2)$ *is a graph morphism $f : G_1 \to G_2$ such that $triple_1 = triple_2 \circ f$.*

The category having triple graphs as objects and triple graph morphisms as arrows is called **TripleGraphs**. Note that the category **TripleGraphs** is equal to the slice category **Graphs**/$TRIPLE$.

We say that $TRIPLE_S$, the graph consisting of node s and loop l_s, $TRIPLE_C$, the graph consisting of nodes s, t, c, edge e_{cs} and edge e_{ct}, $TRIPLE_T$, the graph consisting of node T and loop l_T, are the *source component, correspondence component*, and *target component* of $TRIPLE$, respectively.

We say that $(G_S, triple_{|G_S})$, the restriction of $(G, triple)$ to $TRIPLE_S$, is the *source component*, $(G_C, triple_{|G_C})$, the restriction of $(G, triple)$ to $TRIPLE_C$, is the *correspondence component*, and $(G_T, triple_{|G_T})$, the restriction of $(G, triple)$

to $TRIPLE_T$, is the *target component of* $(G, triple)$. Given a triple graph morphism $f : (G_1, triple_1) \to (G_2, triple_2)$, we say that $f_S = f_{|G_{1,S}}$ is the source component, $f_C = f_{|G_{1,C}}$ is the correspondence component, and $f_T = f_{|G_{1,T}}$ is the target component of f.

We may denote a triple graph $(G, triple)$ as a combination of three capitals (with index), as for example $S_G C_G T_G$, where the first capital, here S_G, denotes the source, the second capital, here C_G, denotes the correspondence, and the third capital, here T_G denotes the target component of $(G, triple)$.

We introduce typed triple graphs as triple graphs typed over a distinguished triple graph, called type triple graph.

Definition 9 (typed triple graph and morphism). *A* type triple graph *$(T, triple_T)$ is a distinguished triple graph.*

A typed triple graph *$((G, triple_G), type)$ is a triple graph $(G, triple_G)$ equipped with a triple graph morphism $type : (G, triple_G) \to (T, triple_T)$.*

Consider typed triple graphs $((G_1, triple_1), type_1)$ and $((G_2, triple_2), type_2)$, a typed triple graph morphism *$f : ((G_1, triple_1), type_1) \to ((G_2, triple_2), type_2)$ is a triple graph morphism $f : (G_1, triple_1) \to (G_2, triple_2)$ such that $type_2 \circ f = type_1$.*

Note that each typed graph $(G_1, type_1)$ and typed graph morphism $f : (G_1, type_1) \to (G_2, type_2)$ typed via $type_1 : G_1 \to T$ and $type_2 : G_2 \to T$ over T such that $type_1 = type_2 \circ f$, where $(T, triple_T)$ is a type triple graph, corresponds uniquely to a typed triple graph $((G_1, triple_T \circ type_1), type_1)$ and typed triple graph morphism $f : ((G_1, triple_T \circ type_1), type_1) \to ((G_2, triple_T \circ type_2), type_2)$, respectively.

We denote the type triple graph $(T, triple_T)$ also as $S_T C_T T_T$, where S_T is its source component, C_T its correspondence component and T_T its target component. For the rest of this appendix every triple graph SCT is typed, although not explicitly mentioned. Analogously, a morphism between typed triple graphs is typed, although not explicitly mentioned. In particular, this means that S is typed over S_T, C is typed over C_T, and T is typed over T_T.

The category having typed triple graphs as objects and typed triple graph morphisms as arrows is called $\mathbf{TripleGraphs}_{S_T C_T T_T}$. Note that $\mathbf{TripleGraphs}_{S_T C_T T_T}$ is the slice category $\mathbf{TripleGraphs}/S_T C_T T_T$ of the slice category $\mathbf{Graphs}/TRIPLE$. Consequently, it follows from [4] that the category $\mathbf{TripleGraphs}_{S_T C_T T_T}$ of typed triple graphs together with the set $\mathcal{M}$ of monomorphisms forms an adhesive HLR category [4].

For simplicity reasons, we do not reintroduce attributed graphs as given in [4]. However, it follows from [4] that, analogous to the case without attribution, we can define the category $\mathbf{ATripleGraphs}_{S_T C_T T_T}$, having typed attributed triple graphs as objects and typed attributed triple graph morphisms as arrows such that with the set $\mathcal{M}$ of injective morphisms with isomorphism on the data part it is an adhesive HLR category [4].

Given the adhesive HLR category $\mathbf{TripleGraphs}_{S_T C_T T_T}$ ($\mathbf{ATripleGraphs}_{S_T C_T T_T}$), a triple graph grammar (TGG) as introduced in Section 2 is an adhesive HLR grammar with non-deleting rules. Forward, backward and correspondence rules and transformations corresponding to a given TGG are distinguished rules (as introduced for the forward case in Section 3 analogous to [22]) and transformations in the adhesive HLR category $\mathbf{TripleGraphs}_{S_T C_T T_T}$ ($\mathbf{ATripleGraphs}_{S_T C_T T_T}$).

B Triple Graph Transformation extended with Bookkeeping

In Section 3, we extend triple graphs and forward rules with a bookkeeping mechanism allowing us to formulate in Section 4 sufficient criteria (determinism criteria) that ensure uniqueness of the transformation result.

In order to be able to apply critical pair analysis as described in [4], we need to show that triple graphs and forward rules extended with bookkeeping can be defined as objects and adhesive HLR rules for a suitable adhesive HLR category. To this extent, we slightly adapt the category $\mathbf{TripleGraphs}_{S_T C_T T_T}$ ($\mathbf{ATripleGraphs}_{S_T C_T T_T}$) to $\mathbf{TripleGraphs}_{BS_T C_T T_T}$ ($\mathbf{ATripleGraphs}_{BS_T C_T T_T}$) by extending $TRIPLE$ to $BTRIPLE$ and accordingly $S_T C_T T_T$ to $BS_T C_T T_T$.

First, since for the bookkeeping of edge translation we use bookkeeping edges from the bookkeeping node b to edges in the source or target graph, we introduce a graph notion where edges from nodes to edges are allowed.

Definition 10 (extended graph and graph morphism). *An* extended graph *$G = (G_N, G_E, G_{E'}, src, tgt, src', tgt')$ consists of a set G_N of nodes, a set G_E of edges, two mappings $src, tgt : G_E \to G_N$, assigning to each edge $e \in G_E$ a source $src(e) \in G_N$ and target $tgt(e) \in G_N$, and two mappings $src' : G_{E'} \to G_N$ and $tgt' : G_{E'} \to G_E$ assigning to each edge $e' \in G_{E'}$ a source $src(e') \in G_N$ and target $tgt(e') \in G_E$. A* extended graph morphism *$f : G_1 \to G_2$ between two extended graphs $G_i = (G_{i,N}, G_{i,E}, G_{i,E'}, src_i, tgt_i, src'_i, tgt'_i)$, $(i = 1, 2)$ is a triple $f = (f_N : G_{1,N} \to G_{2,N}, f_E : G_{1,E} \to G_{2,E}, f_{E'} : G_{1,E'} \to G_{2,E'})$ of mappings such that $f_N \circ src_1 = src_2 \circ f_E$, $f_N \circ tgt_1 = tgt_2 \circ f_E$, $f_N \circ src'_1 = src'_2 \circ f_{E'}$, and $f_E \circ tgt'_1 = tgt'_2 \circ f_{E'}$.*

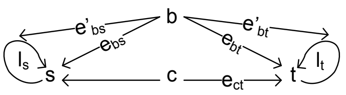
As explained in [4], the category **Graphs** is isomorphic to the functor category $[\mathcal{S}, \mathbf{Sets}]$, where the schema category $\mathcal{S}$ is given by the schema: $\begin{tikzcd} \bullet \arrow[r, shift left] \arrow[r, shift right] & \bullet \end{tikzcd}$. Analogously, the category **ExtGraphs** of extended graphs and extended graph morphisms is isomorphic to the functor category $[\mathcal{ES}, \mathbf{Sets}]$, where the schema category $\mathcal{ES}$ is given by the schema:

$$\begin{tikzcd} & \bullet \arrow[dl, dotted] \arrow[dr, dotted] & \\ \bullet \arrow[rr, shift left] \arrow[rr, shift right] & & \bullet \end{tikzcd}.$$

Therefore, it follows that not only the category **Graphs**, but also the category **ExtGraphs** with $\mathcal{M}$, the set of extended graph monomorphisms is adhesive HLR.

Definition 11 (triple graph and morphism with bookkeeping). *The graph $BTRIPLE$ is an extended graph and can be constructed from the graph $TRIPLE$ as follows: we add a node b, edges e_{bs} with $src(e_{bs}) = b$ and $tgt(e_{bs}) = s$, e_{bt} with $src(e_{bt}) = b$ and $tgt(e_{bt}) = t$, and special edges, e'_{bs} with $src(e'_{bs}) = b$ and $tgt(e'_{bs}) = l_s$, e'_{bt} with $src'(e'_{bt}) = b$ and $tgt'(e'_{bt}) = l_t$.*

BTRIPLE:



We define **TripleGraphs**$_B$ *as the slice category* **ExtGraphs**$/BTRIPLE$*. It consists of so-called triple graphs and morphisms with bookkeeping.*

Consequently, triple graphs with bookkeeping may have incoming edges from the bookkeeping node b[16] to its source or target nodes or edges, expressing which nodes and edges have not been translated yet.

Definition 12 (typed triple graph and morphism with bookkeeping). *Given a triple type graph $S_TC_TT_T$, then we can construct its corresponding triple type graph $BS_TC_TT_T$ with bookkeeping types from $S_TC_TT_T$ as follows: add the node b of type b in $BTRIPLE$ to the node set of $S_TC_TT_T$, add for each node n in S_T (T_T) an edge from b to n of type e_{bs} (resp. e_{bt}), and finally, add for each edge e in S_T (T_T) an edge from b to e of type e'_{bs} (resp. e'_{bt}).*

We define **TripleGraphs**$_{BS_TC_TT_T}$ *as the slice category* **TripleGraphs**$/BS_TC_TT_T$ *of the slice category* **ExtGraphs**$/BTRIPLE$*. It consists of so-called typed triple graphs and morphisms with bookkeeping.*

It follows from [4] that the category **TripleGraphs**$_{BS_TC_TT_T}$ of typed triple graphs with bookkeeping, together with the set $\mathcal{M}$ of monomorphisms forms an adhesive HLR category. Moreover, it follows from [18] that this category with $\mathcal{M} = \mathcal{M}' = \mathcal{Q}'$ is also NAC-adhesive HLR. This is a prerequisite for being able to do critical pair analysis as presented in [4, 18] on bookkeeping rules, since in particular these rules hold NACs.

For the attributed case, it follows in an analogous way that the category **ATripleGraphs**$_{BS_TC_TT_T}$ of typed attributed triple graphs with bookkeeping, $\mathcal{M}$ the set of monomorphisms with isomorphism on the data part, and suitable $\mathcal{M}'$, $\mathcal{Q}$ and $\mathcal{E}'$ morphisms (see [4, 18]) is a NAC-adhesive HLR category.

Given the adhesive HLR category **TripleGraphs**$_{BS_TC_TT_T}$ (**ATripleGraphs**$_{BS_TC_TT_T}$), bookkeeping forward, backward and correspondence rules and transformations corresponding to a given TGG are distinguished rules (as introduced for the bookkeeping forward case in Section 3) and transformations in the NAC-adhesive HLR category **TripleGraphs**$_{BS_TC_TT_T}$ (**ATripleGraphs**$_{BS_TC_TT_T}$).

C Proof of Lemma 1 and Theorem 1

Lemma 1 (partial consistency). *For a TGG and its operationalization $CON = OP_{FT}(TGG)$ holds that $B_{init}S \rightarrow_{a^{BF}} B_ASC_AT_A \rightarrow^*_{CON} B_iSC_iT_i \wedge Trans^F(B_iSC_iT_i) = S_i$ implies $S_AC_AT_A \rightarrow^*_{TGG} S_iC_iT_i$ via the related TGG rules.*

[16] As an additional constraint, we assume that there is exactly one bookkeeping node, i.e. node of type b in $BTRIPLE$, in each triple graph with bookkeeping. Therefore, we denote the bookkeeping node in each triple graph with bookkeeping also by b.

Proof. First, we show that for the bookkeeping forward transformation

$$t^{BF} : B_{init}S \rightarrow_{m_A,a^{BF}} B_A SC_A T_A \rightarrow^*_{CON} B_i SC_i T_i$$

we can construct the following source transformation via corresponding source rules[17]

$$t^S : \emptyset \rightarrow_{a^S} Trans^F(B_A SC_A T_A) \rightarrow^*_{SRC} Trans^F(B_i SC_i T_i)$$

To this extent, first, we show that $B_{init}S \rightarrow_{m_A,a^{BF}} B_A SC_A T_A$ leads to a source transformation $\emptyset \rightarrow_{a^S} Trans^F(B_A SC_A T_A)$ with co-match $m_{A|S_A}$. This is because in $B_{init}S \rightarrow_{m_A,a^{BF}} B_A SC_A T_A$, m_A matches all elements of S_A to elements in S with incoming bookkeeping edges from b and deletes these edges such that $m_A(S_A)$ equals $Trans^F(B_A SC_A T_A)$.

Secondly, we show that for the forward transformation

$$t'^{BF} : B_A SC_A T_A \rightarrow_{m_1,r_1^{BF}} B_1 SC_1 T_1 \ldots \rightarrow_{m_i,r_i^{BF}} B_i SC_i T_i$$

a source transformation

$$t'^S : Trans^F(B_A SC_A T_A) \rightarrow_{r_1^S,m_{1|S_{1,L}}} Trans^F(B_1 SC_1 T_1) \ldots$$

$$\ldots \rightarrow_{r_i^S,m_{i|S_{i,L}}} Trans^F(B_i SC_i T_i)$$

exists. We argue by induction over i, the number of transformation steps.

The rule application $B_A SC_A T_A \rightarrow_{m_1,r_1^{BF}} B_1 SC_1 T_1$ implicates that we have the transformation $Trans^F(B_A SC_A T_A) \rightarrow_{r_1^S,m_{1|S_{1,L}}} Trans^F(B_1 SC_1 T_1)$ with co-match $m_{1|S_{1,R}}$. This is because m_1 matches all elements in $S_{1,L}$ to elements in $Trans^F(B_A SC_A T_A)$ because otherwise NAC^{BF} of rule r_1^{BF} would not be fulfilled. Moreover, m_1 matches all elements in $S_{1,R} \backslash S_{1,L}$ to elements in $B_A SC_A T_A \backslash Trans^F(B_A SC_A T_A)$ because these elements should have incoming bookkeeping edges from b. Exactly these elements are then added to $Trans^F(B_A SC_A T_A)$, obtaining $Trans^F(B_1 SC_1 T_1)$.

Suppose that we have for the bookkeeping forward transformation

$$t''^{BF} : B_A SC_A T_A \rightarrow_{m_1,r_1^{BF}} B_1 SC_1 T_1 \ldots \rightarrow_{m_{i-1},r_{i-1}^{BF}} B_{i-1} SC_{i-1} T_{i-1}$$

a corresponding source transformation

$$t''^S : Trans^F(B_A SC_A T_A) \rightarrow_{r_1^S,m_{1|S_{1,L}}} Trans^F(B_1 SC_1 T_1) \ldots$$

$$\ldots \rightarrow_{r_{i-1}^S,m_{i-1|S_{i-1,L}}} Trans^F(B_{i-1} SC_{i-1} T_{i-1})$$

Then,

$$Trans^F(B_{i-1} SC_{i-1} T_{i-1}) \rightarrow_{r_i^S,m_{i|S_{i,L}}} Trans^F(B_i SC_i T_i)$$

[17] Given a TGG, then SRC denotes the set of source rules that can be derived from the rules in TGG.

exists with co-match $m_i|_{S_{i,R}}$. This is because m_i matches all elements in $S_{i,L}$ to elements in $Trans^F(B_{i-1}SC_{i-1}T_{i-1})$ because otherwise NAC^{BF} of rule r_i^{BF} would not be fulfilled. Moreover, m_i matches all elements in $S_{i,R} \backslash S_{i,L}$ to elements in $B_{i-1}SC_{i-1}T_{i-1} \backslash Trans^F(B_{i-1}SC_{i-1}T_{i-1})$ because these elements should have incoming bookkeeping edges from b. Exactly these elements are then added to $Trans^F(B_{i-1}SC_{i-1}T_{i-1})$ obtaining $Trans^F(B_iSC_iT_i) = S_i \subseteq S$.

Summarizing, we have obtained the source sequence:

$$t^S : \emptyset \rightarrow_{a^S} Trans^F(B_ASC_AT_A) \rightarrow^*_{SRC} Trans^F(B_iSC_iT_i)$$

for the corresponding bookkeeping forward transformation

$$t^{BF} : B_{init}S \rightarrow_{m_A, a^{BF}} B_ASC_AT_A \rightarrow^*_{CON} B_iSC_iT_i$$

By type restriction, we can derive from t^{BF} the forward transformation without bookkeeping via the corresponding forward rules in FOR:

$$t^F : S \rightarrow_{a^F} SC_AT_A \rightarrow^*_{FOR} SC_iT_i$$

Moreover, we can restrict t^F to the transformation:

$$t_i^F : S_i \rightarrow_{a^F} S_iC_AT_A \rightarrow^*_{FOR} S_iC_iT_i$$

This is because the elements belonging to $S \setminus S_i$ (with $Trans^F(B_iSC_iT_i) = S_i \subseteq S$) are not mapped by any match in t^F such that we can apply the Restriction Theorem [4, 18] to t^F obtaining t_i^F. Consider the transformation t^S followed by t_i^F:

$$\emptyset \rightarrow_{a^S} Trans^F(B_ASC_AT_A) \rightarrow_{r_1^S} Trans^F(B_1SC_1T_1) \ldots$$
$$\ldots \rightarrow_{r_i^S} Trans^F(B_iSC_iT_i) \rightarrow_{a^F} S_iC_AT_A \rightarrow_{r_1^F} S_1C_1T_1 \ldots \rightarrow_{r_i^F} S_iC_iT_i$$

We can switch each source transformation step via r_k^S ($1 \leq k \leq i$) in t^S with each forward transformation step via a^F and r_j^F with $1 \leq j < k \leq i$, respectively. This is because these transformations steps are sequentially independent, i.e. nothing is produced in source transformation step k what is used by a forward transformation step via a^F and r_j^F, respectively, since each source transformation step via r_k^S produces graph elements used for the first time by a forward transformation via rule r_k^F. By switching all source transformation steps in t^S with forward transformation steps in t_i^F, starting with switching $Trans^F(B_{i-1}SC_{i-1}T_{i-1}) \rightarrow_{r_i^S} Trans^F(B_iSC_iT_i) \rightarrow_{a^F} S_iC_AT_A$, as much as possible to the right, we obtain a transformation

$$t : \emptyset \rightarrow_{a^S} S_A \rightarrow_{a^F} S_AC_AT_A \rightarrow_{r_1^S} S_1C_AT_A \rightarrow_{r_1^F} S_1C_1T_1 \ldots$$
$$\ldots S_iC_{i-1}T_{i-1} \rightarrow_{r_i^F} S_iC_iT_i$$

We can build concurrent rules of the source and forward transformations such that using the Concurrency Theorem [4], we obtain:

$$S_AC_AT_A \rightarrow_{r_1^S * r_1^F} S_1C_1T_1 \ldots \rightarrow_{r_i^S * r_i^F} S_iC_iT_i$$

Consequently, $S_iC_iT_i$ is an element of $\mathcal{L}(TGG)$ with $S_i = Trans^F(B_iSC_iT_i)$. $\square$

Theorem 1 (conformance). *For a TGG and its operationalization $CON = OP_{FT}(TGG)$, it holds that $FT_{TGG}(S) = FT_{CON}(S)$. In particular, $B_{init}S \rightarrow_{a^{BF}} B_A S C_A T_A \rightarrow^*_{CON} B_i S C_i T_i$ and $Trans^F(B_i S C_i T_i) = S$ if and only if $S_A C_A T_A \rightarrow^*_{TGG} S C_i T_i$ via the related TGG rules.*

Proof. *$FT_{CON}(S) \subseteq FT_{TGG}(S)$ (consistency) follows from Lemma 1 for the special case that $Trans^F(B_i S C_i T_i) = S$.*

$FT_{CON}(S) \supseteq FT_{TGG}(S)$ (completeness) holds because of the following argumentation: Each transformation

$$t : \emptyset \rightarrow_a S_A C_A T_A \rightarrow_{r_1} S_1 C_1 T_1 \ldots \rightarrow_{r_i} S_i C_i T_i$$

where $S_i = S$, can be decomposed via the Concurrency Theorem [4] into a transformation

$$t' : \emptyset \rightarrow_{a^S} S_A \rightarrow_{a^F} S_A C_A T_A \rightarrow_{r_1^S} S_1 C_A T_A \rightarrow_{r_1^F} S_1 C_1 T_1 \ldots$$

$$\ldots \rightarrow_{r_i^S} S_i C_{i-1} T_{i-1} \rightarrow_{r_i^F} S_i C_i T_i$$

The forward transformation step via a^F can be switched with each source transformation step via r_j^S $(1 \leq j \leq i)$ since these steps are sequentially independent. This is because a^F produces only correspondence and target elements that are not used by some source transformation step via r_j^S $(1 \leq j \leq i)$. Moreover, each forward transformation step via r_k^F $(1 \leq k \leq i-1)$ can be switched with each source transformation step via r_j^S $(1 \leq k < j \leq i)$ since these steps are sequentially independent. This is because each r_k^F only produces correspondence and target elements not used by some source transformation step via r_j^S. By switching like this in t' each forward transformation step with each source transformation step, starting with switching $S_{i-1} C_{i-2} T_{i-2} \rightarrow_{r_{i-1}^F} S_{i-1} C_{i-1} T_{i-1} \rightarrow_{r_i^S} S_i C_{i-1} T_{i-1}$, as much as possible to the right, we obtain a source transformation t^S where first, S_i is generated via the source rules in SRC with

$$t^S : \emptyset \rightarrow_{a^S} S_A \rightarrow_{r_1^S, m_{1|S_{1,L}}} S_1 \ldots \rightarrow_{r_i^S, m_{i|S_{i,L}}} S_i$$

and co-matches $m_{A|S_A}, m_{1|S_{1,R}}, \ldots, m_{i|S_{i,R}}$, respectively, together with a forward transformation t^F transforming S_i into $S_i C_i T_i$ via the corresponding forward rules in FOR

$$t^F : S_i \rightarrow_{a^F, m_A} S_i C_A T_A \rightarrow_{r_1^F, m_1} S_i C_1 T_1 \ldots \rightarrow_{r_i^F, m_i} S_i C_i T_i$$

Thereby note, that after this switching, the elements produced by the first source transformation step via a^S in t^S are matched for the first time in t^F by m_A in the first forward transformation step via a^F. Moreover, the elements produced by the k-th $(1 \leq k \leq i)$ source transformation step via r_k^S in t^S are matched for the first time in t^F by $m_k(S_{k,R} \setminus S_{k,L})$ in the k-th forward transformation step via the corresponding forward rule r_k^F. Therefore, we can enrich t^F to a bookkeeping forward transformation

$$t^{BF} : B_{init} S_i \rightarrow_{a^{BF}} B_A S_i C_A T_A \rightarrow_{r_1^{BF}} B_1 S_i C_1 T_1 \ldots \rightarrow_{r_i^{BF}} B_i S_i C_i T_i$$

via the corresponding rules in CON as follows: First, we add bookkeeping edges to all elements in S_i, obtaining $B_{init}S_i$. We enrich the forward transformation step via rule a^F to a forward transformation step with bookkeeping via a^{BF} such that it deletes exactly those bookkeeping edges pointing to elements in S_i produced by a^S. We then have $Trans^F(B_ASC_AT_A) = S_A$. Moreover, we enrich each forward transformation step via r_k^F $(1 \leq k \leq i)$ to a forward transformation step with bookkeeping via r_k^{BF} such that it deletes exactly those bookkeeping edges pointing to elements in S_i produced by the corresponding source transformation step via r_k^S such that in the end $Trans^F(B_iSC_iT_i) = S_i = S$. $\square$

ISBN 978-3-86956-078-6
ISSN 1613-5652