On the Relationship of Model Transformations Based on Triple and Plain Graph Grammars (Long Version)

Hartmut Ehrig, Claudia Ermel and Frank Hermann

[ehrig, lieske, frank](at)cs.tu-berlin.de
Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin, Germany

Bericht-Nr. 2008/05
ISSN 1436-9915

On the Relationship of Model Transformations Based on Triple and Plain Graph Grammars (Long Version)

Hartmut Ehrig, Claudia Ermel and Frank Hermann

[ehrig, lieske, frank](at)cs.tu-berlin.de
Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin, Germany

Abstract

Triple graph grammars have been applied and implemented as a formal basis for model transformations in a variety of application areas. They convince by special abilities in automatic derivation of forward, backward and several other transformations out of just one specified set of rules for the integrated model defined by a triple of graphs. While many case studies and all implementations, which state that they are using triple graph grammars, do not use triples of graphs, this paper presents the justification for many of them. It shows a one to one correspondence between triple graph grammars and suitable plain graph grammars, thus results and benefits of the triple case can be transferred to the plain case.

Main results show the relationship between both graph transformation approaches, syntactical correctness of model transformations based on triple graph grammars and a sound and complete condition for functional behaviour. Theoretical results are elaborated on an intuitive case study for a model transformation from class diagrams to database models.

Keywords: model transformation, graph transformation, triple graph transformation

1 Introduction

Model transformations are the heart of the model-driven software development approach. Ideally, a model transformation approach supports a visual specification of the transformation with an underlying formal foundation. Additionally, the approach should allow synchronizing different models and keeping them consistent.

Triple Graph Grammars (TGGs) [16] have been defined to consistently co-develop two related structures modeled by graphs. These are connected using a *correspondence graph* together with its embeddings into the two graphs. TGG rules (triples of non-deleting graph rules) generate the language of triple graphs, i.e. they describe the parallel extension of all three graphs. Based on TGGs, triple rules can be decomposed e.g. into source and forward rules.

In recent years, TGGs have shown to be an adequate basis to specify visual, formal and bidirectional model transformations between different domain-specific modeling languages [10, 6, 8]. Under certain conditions (*source consistency*, see [6]), important properties of model transformations based on forward rules can be shown, e.g. the bijective correspondence of composition and decomposition of triple graph transformation sequences and information preservation of a triple graph transformation sequence[6]. Moreover, the relation of model transformation and *model integration* based on integration rules derived from triple rules has been shown in [8].

Sample applications for model transformations using TGGs are a conversion between the two different computer graphics file formats [2], and an approach for view consistency management in the context of multi-view visual languages [4], where views are related to an integrated model by correspondence graphs. Tools developed for typed attributed graph transformation have been used frequently to implement model transformations based on TGGs (such as FUJABA [17], AToM^3[5], and AGG [1]). Therefore, the structure of (typed) triple graphs have been represented as plain graphs (ordinary typed attributed graphs): the three graphs are modeled as a single integrated graph, where the embedding of the correspondence graph is represented by additional edges. Analogously, a single integrated type graph models the original triple type graph.

Up to now, it has not been shown formally that the results which hold for TGGs (e.g. the canonical decomposition of triple graph transformation sequences into source and forward sequences, and their composition [6]) still hold for their representation as plain graph grammars.

In this paper, we define a flattening functor mapping triple graphs to plain graphs. We extend this functor to the translation of triple graph grammars and transformations. Based on the properties of this functor, we can show that important results for triple graph grammars and transformations can be transferred to plain graph grammars and transformations. In particular, we show that TGG properties based on source consistency of triple graph transformations can now be analyzed using the corresponding plain graph transformations. Our first main theorem states that the translation leads to a bijective correspondence between a model transformation based on forward rules in TGGs and the corresponding translated plain model transformation. Hence, results proven for TGGs like composition and decomposition of triple graph transformation sequences, or information preservation under the condition of source consistency, can be transferred to the corresponding plain model transformation sequences. The second main theorem in this paper is concerned with correctness properties of the translated model transformation sequences in plain graphs, such as syntactical correctness and confluence (leading to functional behavior of model transformations).

Our main theorems are illustrated by a model transformation $CD2RDBM$ from UML class diagrams [14] to relational database models, a quasi-standard example which is elaborated in [15, 3].

The paper is structured as follows: In Section 2 we start with a review of plain graphs and graph transformation according to the double-pushout (DPO) approach [9] on the one hand, and triple graphs and triple graph grammars according to [16] on the other hand. Section 3 reviews the derivation of source, target, forward and backward rules from TGG rules and states the source consistency condition for model transformations based on forward rules, which ensures that all parts of a source model are translated completely and not twice. The functor translating triple graph transformations to plain graph transformations is defined in Section 4. In Section 5 we relate properties of model transformations based on forward rules from TGGs to their corresponding model transformations in plain graphs. In particular, we show that our translation functor preserves syntactical correctness and functional behavior of model transformations. We conclude the paper in Section 6, discussing open problems and directions for future research.

2 Review of DPO- and Triple Graph Grammars

Graphs and graph transformations are used in a variety of types for specifying, analyzing and optimizing systems. Here, we focus on graphs with explicit source and target functions for edges. We recall first the main definitions of plain graphs and their transformation, which are necessary for further constructions. For details and extensions such as attribution or type graphs with inheritance, we refer to [9]. Note that these extensions can be applied directly and the results will not be affected.

Definition 1. Graphs and Graph Morphisms: *A graph $G = (G_V, G_E, sG, tG)$ consists of a set G_V of nodes, a set G_E of edges, and two functions $sG, tG : G_E \to G_V$, the source and the target function.*

Given graphs G, H a graph morphism $f : G \to H, f = (f_V, f_E)$ consists of two functions $f_V : G_V \to H_V$ and $f_E : G_E \to H_E$ that preserve the source and the target function, i.e. $f_V \circ sG = sH \circ f_E$ and $f_V \circ tG = tH \circ f_E$. Graphs and graph morphisms define the category **Graphs**. *A graph morphism f is injective if both functions f_V , f_E are injective.*

Example 1. Class diagram as graph: *The graph in Fig. 1 defines a class diagram containing the classes "Company", "Person" and its specialization "Customer", an assoziation "employee" and an attribute "cust_id" of type "int". Its visualization, thus including concrete syntax information, is given in Fig. 2.*

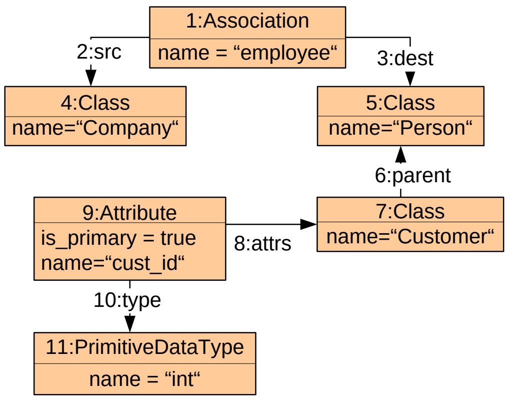

Figure 1: Graph instance for abstract syntax of a class diagram

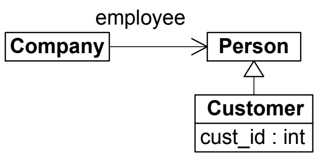

Figure 2: Visualization of graph in Fig. 1

Definition 2. Typing: *Given a distinguished graph TG, called type graph, a typed graph $G = (G, type_G)$ consists of a graph $G = (V, E, s, t)$ together with a type morphism $type_G : G \rightarrow TG$ from G to its type graph TG. A type graph is a distinguished graph TG. A tuple $(G, type_G)$ of a graph G together with a graph morphism $type_G : G \rightarrow TG$ is called a typed graph. Given typed graphs $G = (G, type_G)$ and $H = (H, type_H)$, a typed graph morphism f is a graph morphism $f : G \rightarrow H$, such that $type_H \circ f = type_G$.*

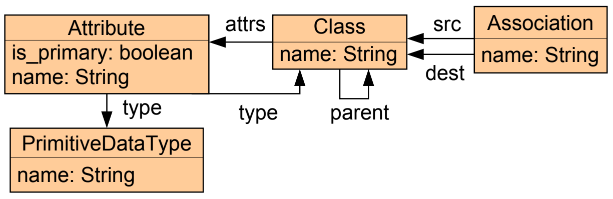

Figure 3: Type graph for class diagrams

Example 2. Typing: *The structure of class diagrams in the CD2RDBM example is given by the type graph in Fig. 3 containing the types "*Class*", "*Attribute*", "*PrimitiveDataType*" and "*Association*" and instances of these node types have to be linked according to the edge types between the node types as well as attributed according to node type attributes.*

Graph transformation rules specify local operations. A graph grammar specifies a graph language as set of all those graphs that can be created by applying transformation rules starting with a given start graph.

Definition 3. Typed Graph Grammar: *A typed graph rule* $p = L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R$ *consists of typed graphs* L, K*, and* R*, called the left-hand side, gluing graph, and the right-hand side respectively, and two injective typed graph morphisms* l *and* r*. A typed graph grammar* $GG = (TG, S, R)$ *consists of a type graph* TG*, a start graph* S *and graph rules* R*. If a rule* p *is applicable to a graph* G *via a morphism* $m : L \to G$*, called match, the transformation* $G \overset{p}{\Longrightarrow} H$ *is defined by two pushouts (DPO diagram):*

$$\begin{tikzcd}
L \arrow[d, "m"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"'] & R \arrow[d, "m*"] \\
G & D \arrow[l, "l*"] \arrow[r, "r*"'] & H
\end{tikzcd}$$

The typed graph language L *of* GG *is defined by* $L = \{G \mid \exists$ *typed graph transformation* $S \Rightarrow^* G\}$.

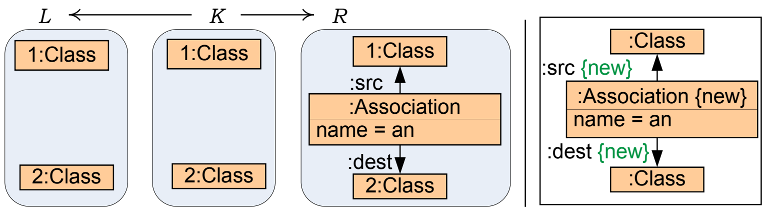

Figure 4: Graph rule insertAssociation() in complete and compact notation

Example 3. Graph Grammar: *One rule of a generating graph grammar for class diagrams is given in Fig. 4. It inserts an association between two existing classes. The rule is presented both in complete form and compact form. The left part of Fig. 4 for complete notation contains the rule components* L*,* K *and* R *as distinct graphs and numbers indicating the morphisms* $l : K \to L, r : K \to R$*. The right part shows the compact form, where elements to be created are labeled with "*{new}*". If there are elements in a rule that shall be deleted, they are labeled with "*{del}*" in the compact form.*

In the following definitions, the concepts of graph transformation are lifted to the case of triple graphs, an extension of plain graphs dividing elements into source, target and correspondence sections. Since triple graph transformations are considered to be used for model transformation they are restricted to non-deleting rules, which can still be seen as a special case of general DPO graph rules.

Definition 4. Triple Graph and Triple Graph Morphism: *Three graphs* SG*,* CG*, and* TG*, called source, connection, and target graph, together with two graph morphisms* $s_G : CG \to SG$ *and* $t_G : CG \to TG$ *form a triple graph* $G = (SG \overset{s_G}{\leftarrow} CG \overset{t_G}{\rightarrow} TG)$*.* G *is called empty, if* SG*,* CG*, and* TG *are empty graphs.*

A triple graph morphism $m = (s, c, t) : G \to H$ *between two triple graphs* $G = (SG \overset{s_G}{\leftarrow} CG \overset{t_G}{\rightarrow} TG)$ *and* $H = (SH \overset{s_H}{\leftarrow} CH \overset{t_H}{\rightarrow} TH)$ *consists of three graph morphisms* $s : SG \to SH$*,* $c : CG \to CH$ *and* $t : TG \to TH$ *such that* $s \circ s_G = s_H \circ c$ *and* $t \circ t_G = t_H \circ c$*. It is injective, if morphisms* s*,* c *and* t *are injective. Triple graphs and triple graph morphisms form the category* **TripleGraphs**.

Example 4. Triple graph: *The graph in Fig. 5 shows a triple graph containing a class diagram together with connecting reference nodes in the connection component visualized by circles pointing to the database model of the target language. References between source and target model denote translation correspondences and exist between classes and tables, associations and foreign keys as well as between attributes and columns. Inheritance information in class diagrams are flattened in corresponding database models.*

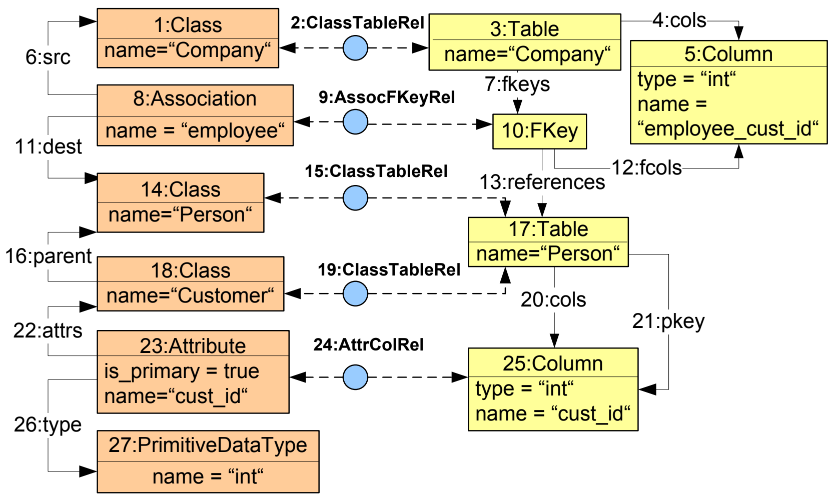

Figure 5: Triple graph for an integrated CD2RDBM model

Definition 5. Triple Graph Grammar:

A triple rule $tr = L \xrightarrow{tr} R$ consists of triple graphs L and R and an injective triple graph morphisms tr. A triple graph grammar $TGG = (TG, S, TR)$ consists of a triple type graph TG, a triple start graph S and triple rules TR typed over TG.

$$\begin{tikzcd}
L \arrow[d, "tr"'] & = & (SL \arrow[d, "s"'] & CL \arrow[l, "s_L"'] \arrow[r, "t_L"] \arrow[d, "c"'] & TL) \arrow[d, "t"] \\
R & = & (SR & CR \arrow[l, "s_R"] \arrow[r, "t_R"'] & TR)
\end{tikzcd}$$

*Given a triple rule $tr = (s, c, t) : L \to R$, a triple graph G and a triple graph morphism $m = (sm, cm, tm) : L \to G$, called triple match m, a triple graph transformation step (*TGT-*step)$G \overset{tr,m}{\Longrightarrow} H$ from G to a triple graph H is given by three objects SH, CH and TH in category* **Graph** *with induced morphisms $s_H : CH \to SH$ and $t_H : CH \to TH$. Morphism $n = (sn, cn, tn)$ is called comatch.*

$$\begin{tikzcd}
 & & SL \arrow[dl, "sm"'] \arrow[dd, dotted] & & CL \arrow[ll] \arrow[rr] \arrow[dl, "cm"'] \arrow[dd, dotted] & & TL \arrow[dl, "tm"'] \arrow[dd] \\
G = \arrow[dd, Rightarrow, "tr"'] & (SG \arrow[dd, "s'"'] & & CG \arrow[ll] \arrow[rr] \arrow[dd, "c'"'] & & TG) \arrow[dd, "t'"'] & \\
 & & SR \arrow[dl, dotted, "sn"] & & CR \arrow[ll, dotted] \arrow[rr, dotted] \arrow[dl, dotted, "cn"] & & TR \arrow[dl, "tn"] \\
H = & (SH & & CH \arrow[ll, "s_H"] \arrow[rr, "t_H"'] & & TH) &
\end{tikzcd}$$

The triple graph language L of TGG is defined by $L = \{G \mid \exists$ triple graph transformation $S \Rightarrow^* G\}$.

Remark 1. *Triple transformation steps are constructed by one pushout in category* **TripleGraphs**, *since we consider non-deleting rules only.*

Example 5. Triple rules: *Rules in Fig. 6 are part of a triple graph grammar that synchronously generates class diagrams and corresponding database models, where the first rule "Class2Table" may create a class and its corresponding table at any time. The second rule "PrimaryAttribute2Column" inserts a primary attribute for a given class, thus creating a corresponding primary key column in the connected table of the database model. Finally, "Subclass2Table" adds a class which inherits from an existing one, thus it is connected to the same corresponding table as the parent class. The rules "Attribute2Column" and "Association2ForeignKey" given in [6] are not shown here.*

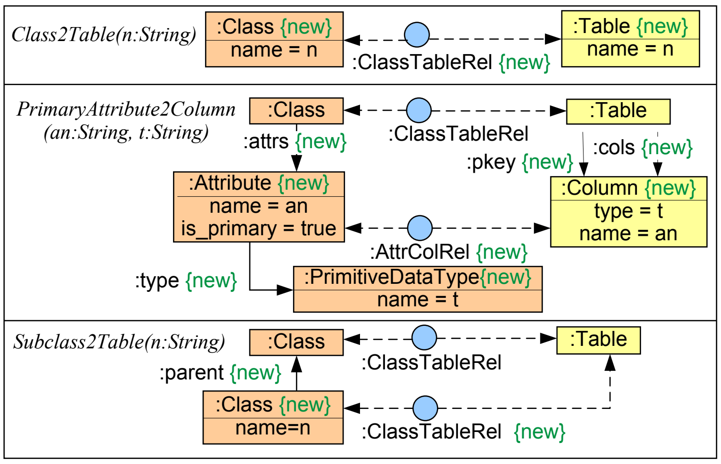

Figure 6: Triple rules for CD2RDBM model transformation

3 Model Transformation based on Triple Graph Grammars

The triple rules TR are defining the language $VL = \{G \,|\, \emptyset \Rightarrow^* G \text{ via } TR\}$ of triple graphs. Source language VL_S and target language are derived by projection to the triple components, i.e. $VL_S = proj_S(VL)$ and $VL_T = proj_T(VL)$, where $proj_X$ is a projection defined by restriction to one of the triple components, i. e. $X \in \{S, C, T\}$.

Definition 6. Derived Triple Rules: *From each triple rule $tr = L \to R$ we have the following source, forward, target and backward rules:*

$$\begin{tikzcd}
(SL \arrow[d, "s"'] & \emptyset \arrow[l] \arrow[r] \arrow[d] & \emptyset) \arrow[d] \\
(SR & \emptyset \arrow[l] \arrow[r] & \emptyset)
\end{tikzcd}
\qquad
\begin{tikzcd}
(\emptyset \arrow[d] & \emptyset \arrow[l] \arrow[r] \arrow[d] & TL) \arrow[d, "t"'] \\
(\emptyset & \emptyset \arrow[l] \arrow[r] & TR)
\end{tikzcd}$$

source rule tr_S   *target rule tr_T*

$$\begin{tikzcd}
(SR \arrow[d, "id"'] & CL \arrow[l, "s \circ s_L"'] \arrow[r, "t_L"] \arrow[d, "c"'] & TL) \arrow[d, "t"] \\
(SR & CR \arrow[l, "s_R"'] \arrow[r, "t_R"] & TR)
\end{tikzcd}
\qquad
\begin{tikzcd}
(SL \arrow[d, "s"'] & CL \arrow[l, "s_L"'] \arrow[r, "t \circ t_L"] \arrow[d, "c"'] & TR) \arrow[d, "id"'] \\
(SR & CR \arrow[l, "s_R"'] \arrow[r, "t_R"] & TR)
\end{tikzcd}$$

forward rule tr_F   *backward rule tr_B*

Source rules allow to create all elements of VL_S as restriction of VL, but they contain less restrictions for matches during transformation in comparison to their corresponding complete triple rules. Thus, they possibly allow to generate more elements than VL_S contains. It is an open problem in which cases the inclusion $VL_S \subseteq VL_{S0} = \{G_S \,|\, \emptyset \Rightarrow^* G_S \text{ via } TR_S\}$ resp. $VL_T \subseteq VL_{T0} = \{G_T \,|\, \emptyset \Rightarrow^* G_T \text{ via } TR_T\}$ is an equality.

Model transformations from elements of the source language VL_{S0} to elements of the target language VL_{T0} can be defined on the basis of forward rules. Vice versa using backward rules - which are dual to forward rules - it is also possible to define backward transformations from target to source graphs and altogether bidirectional model transformations. In [6] we have shown that there is an equivalence between corresponding forward and backward TGT sequences. This equivalence is based on the canonical decomposition and composition result (see Theorem 1 below), which is also the basis for main results of this paper and it uses the following notion of match consistency.

Definition 7. Match and Source Consistency: *Let tr_S^* and tr_F^* be sequences of source rules tri_S and forward rules tri_F, which are derived from the same triple rules tri for $i = 1, \ldots, n$. Let further $G_{00} \xRightarrow{tr_S^*} G_{n0} \xRightarrow{tr_F^*} G_{nn}$ be a* TGT*-sequence with (mi_S, ni_S) being match and comatch of tri_S (respectively (mi, ni) for tri_F) then match consistency of*

$G_{00} \xrightarrow{tr_S^*} G_{n0} \xrightarrow{tr_F^*} G_{nn}$ means that the S-component of the match mi is uniquely determined by the comatch ni_S $(i = 1, \ldots, n)$.
A TGT*-sequence $G_{n0} \xrightarrow{tr_F^*} G_{nn}$ is source consistent, if there is a match consistent sequence $\emptyset \xrightarrow{tr_S^*} G_{n0} \xrightarrow{tr_F^*} G_{nn}$. Note that by source consistency the application of the forward rules is controlled by the source sequence, which generates the given source model.*

Theorem 1. Canonical Decomposition and Composition Result

1. **Decomposition:** *For each* TGT*-sequence*
   (1) *$G_0 \xrightarrow{tr^*} G_n$ there is a canonical match consistent* TGT*-sequence*
   (2) *$G_0 = G_{00} \xrightarrow{tr_S^*} G_{n0} \xrightarrow{tr_F^*} G_{nn} = G_n$ using corresponding source rules tr_S^* and forward rules tr_F^*.*
2. **Composition:** *For each match consistent transformation sequence (2) there is a canonical transformation sequence (1).*
3. **Bijective Correspondence:** *Composition and Decomposition are inverse to each other.*

The proof is given in [6].

Definition 8. Model Transformation based on Forward Rules: *A model transformation sequence $(G_S,\ G_1 \xrightarrow{tr_F^*} G_n, G_T)$ consists of a source graph G_S, a target graph G_T, and a source consistent forward* TGT*-sequence $G_1 \xrightarrow{tr_F^*} G_n$ with $G_S = proj_S(G_1)$ and $G_T = proj_T(G_n)$.*
A model transformation $MT : VL_{S0} \Rrightarrow VL_{T0}$ is defined by model transformation sequences $(G_S, G_1 \xrightarrow{tr_F^*} G_n, G_T)$ with $G_S \in VL_{S0}$ and $G_T \in VL_{T0}$.

Example 6. Model Transformation: *Using the forward rules of the triple rules in Example 5 we obtain a model transformation sequence $(G_S, G_1 \xrightarrow{tr_F^*} G_n, G_T)$, where G is the triple graph in Fig. 5 and $G_S = proj_S(G), G_T = proj_T(G)$ are given by left and right parts of Fig. 5, respectively (see [6] for an explicit construction).*

For each model transformation $MT : VL_{S0} \Rrightarrow VL_{T0}$ based on forward rules we can ensure that it starts at models in VL_S only and also ends at models in VL_T, thus it corresponds directly to the generating sequence of original triple rules, which synchronously generates elements in all three triple components.

Fact 1. Scope of Model Transformation: *For each model transformation sequence $(G_S, G_1 \xrightarrow{tr_F^*} G_n, G_T)$ of $MT : VL_{S0} \Rrightarrow VL_{T0}$ with $G_S \in VL_{S0}$ and $G_T \in VL_{T0}$ we have that G_1 is typed over $(TG_S \leftarrow \emptyset \rightarrow \emptyset)$ with $proj_S(G_1) = G_S \in VL_S$ and $proj_T(G_n) = G_T \in VL_T$, i.e. $MT : VL_S \Rrightarrow VL_T$.*

Proof. Since $G_1 \xrightarrow{tr_F^*} G_n$ is source consistent, we have $\emptyset \xrightarrow{tr_S^*} G_1 \xrightarrow{tr_F^*} G_n$ match consistent and hence, by Theorem 1 above with $G_0 = \emptyset$ we have $\emptyset \xrightarrow{tr^*} G_n$. This implies $G_n \in VL$, $proj_S(G_n) \in VL_S, proj_T(G_n) \in VL_T$ by definition of VL, VL_S and VL_T. Now we have $G_S = proj_S(G_1) = proj_S(G_n) \in VL_S$ and $G_T = proj_T(G_n) \in VL_T$ and $\emptyset \xrightarrow{tr_S^*} G_1$ implies that G_1 is typed over $(TG_S \leftarrow \emptyset \rightarrow \emptyset)$. □

Note that we can check whether a forward transformation sequence $G_1 \xrightarrow{tr_F^*} G_n$ is source consistent, thus we can check whether it defines a model transformation according to Definition 8. If the check is successful, it constructs the corresponding source rule sequence $\emptyset \xrightarrow{tr_S^*} G_1$. Otherwise $G_1 \xrightarrow{tr_F^*} G_n$ is not source consistent. The notation $m|_S$ denotes the source component of match m, $B(m)$ is the boundary of the initial pushout of m (see Chapter 6 in [9]) and $B(m) \subseteq L$ denotes an injective embedding $B(m) \rightarrow L$ compatible with the rule morphism and match m with L being the left hand side of the rule. If the check is successful it leads to the sequence: $\emptyset \xrightarrow{tr1_S} G_{S,1} \ldots \xrightarrow{trk_S} G_{S,k} = G_1 \xrightarrow{tr_F^*} G_n$.

Fact 2. Source Consistency Check: *$G_1 \xRightarrow{tr_F^*} G_n$ with $(tri_F, mi)_{i=1\ldots k}$ is source consistent iff the following source consistency check is successful, visualized by the derivation steps in the S-component beneath.*
Source Consistency Check:
For $i = 0$ to $(k-1)$ repeat the following two steps

1. **construction of** nj_S**:** *$j = k-i$, if $mj|_S(SR_j) \subseteq G_{S,j}$ then restrict $mj|_S : SR_j \to G_S = G_{S,k}$ to $nj_S : SR_j \to G_{S,j}$, otherwise stop*
2. **construction of** $G_{S,j-1}$**:** *stop, if $B(nj_S) \not\subseteq SL_j$, otherwise the gluing condition is satisfied and the following construction is uniquely defined: $G_{S,j-1}$ is pushout complement of $SL_j \xrightarrow{trj_S} SR_j \xrightarrow{nj_S} G_{S,j}$*

Finally, the check is successful if $G_{S,0}$ could be constructed and it is the empty graph $\emptyset$.

$$\begin{tikzcd}
SL_1 \arrow[r, "tr1_S"] \arrow[d] & SR_1 \arrow[dr, "n1_S"] & SL_2 \arrow[d] \arrow[r, dotted, no head] & \cdot \arrow[dr, "n(k-1)_S"] & SL_k \arrow[r, "trk_S"] \arrow[d] & SR_k \arrow[dr, "nk_S"] & \\
\emptyset \arrow[rr] & & G_{S,1} \arrow[rr, dotted, no head] & & G_{S,k-1} \arrow[rr] & & G_{S,k} \\
SR_1 \arrow[r] \arrow[d, "m1|_S"] & SR_1 \arrow[dr] & SR_2 \arrow[d, "m2|_S"] \arrow[r, dotted, no head] & \cdot \arrow[dr] & SR_k \arrow[r] \arrow[d, "mk|_S"] & SR_k \arrow[dr] & \\
G_{S,k} \arrow[rr, "\cdots"] & & G_{S,k} \arrow[rr, dotted, no head] & & G_{S,k} \arrow[rr] & & G_{S,k}
\end{tikzcd}$$

Proof. The consistency check leads to a match consistent sequence $\emptyset \xRightarrow{tr_S^*} G_S \xRightarrow{tr_F^*} G$. Vice versa, if we have a match consistent sequence $\emptyset \xRightarrow{tr_S^*} G_S \xRightarrow{tr_F^*} G_n$ then the S-component leads to the sequence shown above, which shows that the source consistency check is successful. □

The presented check of a given forward transformation sequence $G_S \xRightarrow{tr_F^*} G_n$ is intuitively a parsing of the source model G_S by calculating the intermediate graphs of its construction using source rules. If the check succeeds it shows that the model transformation was complete. Furthermore, if all triple rules create parts in source and target component it is ensured that no model element was translated twice. Considering the forward sequence given in Example 2 of [6] leading to Fig. 5 a check for source consistency succeeds.

4 Translation of Triple Graph Grammars and Transformations

Most case studies and all implementations for triple graph grammars, which we are aware of, use plain graphs as an encoding of triple graphs. But there may arise problems since there are graphs and graph transformations not corresponding to a triple graph transformation, whenever the correspondence component is linked with zero or multiple edges to one of the other component. The following definition of a flattening corresponds to the intuitive translation used in tools and we show that results of the theory can be transferred to the plain case, whenever the graphs and rules are a proper encoding of triple graphs.

Definition 9. Flattening Construction: *Given a triple graph $G = (SG \overset{s_G}{\leftarrow} CG \overset{t_G}{\rightarrow} TG)$ the flattening $\mathcal{F}(G)$ of G is a plain graph defined by the disjoint union $\mathcal{F}(G) = SG + CG + TG + Links_S(G) + Link_T(G)$ with links (additional edges) defined by*
$Links_S(G) = \{(x, y) \mid x \in CG_V, y \in SG_V, s_G(x) = y\}$,
$Link_T(G) = \{(x, y) \mid x \in CG_V, y \in TG_V, t_G(x) = y\}$ with $s_{\mathcal{F}(G)}((x, y)) = x$ and $t_{\mathcal{F}(G)}((x, y)) = y$, $(x, y) \in Links_S \cup Link_T$.

Given a triple graph morphism $f = (f_S, f_C, f_T) : G \to G'$ the flattening $\mathcal{F}(f) : \mathcal{F}(G) \to \mathcal{F}(G')$ is defined by $\mathcal{F}(f) = f_S + f_C + f_T + f_{LS} + f_{LT}$ with $f_{LS} : Links_S(G) \to Links_S(G')$, $f_{LT} : Link_T(G) \to Link_T(G')$ defined by $f_{LS}((x, y)) = (f_C(x), f_S(y))$ and $f_{LT}((x, y)) = (f_C(x), f_T(y))$.

Remark 2. *The mapping of edges in CG_E is disregarded, but in the following we assume $CG_E = \emptyset$ anyhow.*

Example 7. Flattening Construction: *Consider Fig. 5 showing the triple graph* $G = (SG \overset{s_G}{\leftarrow} CG \overset{t_G}{\rightarrow} TG)$ *for our integrated CD2RDBM model. The graph* $\mathcal{F}(G)$*, resulting from applying the flattening functor to* G*, consists of*

- *the subgraphs* SG, CG *and* TG*,*
- *edges* $Link_S(G)$ *corresponding to the morphism* $SG \overset{s_G}{\leftarrow} CG$*, defined by* $Link_S(G) = \{(2,1),(9,8),\ (15,14),19,18),\ (24,23)\}$ *(where the numbers refer to the numbered nodes in Fig. 5), with* $s_{\mathcal{F}(G)}((2,1)) = 2, t_{\mathcal{F}(G)}((2,1)) = 1$ *(analogously for all other edges in* $Link_S(G)$*),*
- *and edges* $Link_T(G)$ *corresponding to the morphism* $CG \overset{t_G}{\rightarrow} TG$*), defined by* $Link_T(G) = \{(2,3),(9,10),\ (15,17),(19,17),(24,25)\}$.

The following Fact 3 ensures several important properties of the flattening construction, which are used later.

Fact 3. Properties of Flattening Construction:

1. *The flattening construction defines a functor* $\mathcal{F} : \mathbf{TripleGraphs} \to \mathbf{Graphs}$*, which preserves pushouts.*
2. *Given a triple type graph* $TG = (STG \leftarrow CTG \rightarrow TTG)$ *with* $CTG_E = \emptyset$ *and flattening* $\mathcal{F}(TG)$*. Then the typed flattening construction is the functor* $\mathcal{F}_{TG} : \mathbf{TripleGraphs}_{TG} \to \mathbf{Graphs}_{\mathcal{F}(TG)}$ *defined by* $\mathcal{F}_{TG}(G,t) = (\mathcal{F}(G), \mathcal{F}(t))$ *and* $\mathcal{F}_{TG}(f) = \mathcal{F}(f)$*. We sometimes write* $\mathcal{F}_{TG} = \mathcal{F}$ *for short.*
3. *The typed flattening* $\mathcal{F}_{TG}$ *is injective on objects and creates morphisms, i.e. for all* $m' : \mathcal{F}_{TG}(L) \to \mathcal{F}_{TG}(G)$ *in* $\mathbf{Graphs}_{\mathcal{F}(TG)}$ *there is a unique* $m : L \to G$ *with* $\mathcal{F}_{TG}(m) = m'$*. Especially we have* $\mathcal{F}_{TG}(A) \cong \mathcal{F}_{TG}(B)$ *iff* $A \cong B$*, and* $\mathcal{F}_{TG}$ *is injective on morphisms.*
4. $\mathcal{F}_{TG}$ *preserves and reflects pushouts, i.e.* (1) *pushout in* $\mathbf{TripleGraphs_{TG}}$ *iff* (2) *is pushout in* $\mathbf{Graphs}_{\mathcal{F}(TG)}$*, and* $\mathcal{F}_{TG}$ *creates pushouts,*

$$\begin{tikzcd} L \arrow[r, "r"] \arrow[d, "m"'] \arrow[dr, phantom, "(1)"] & R \arrow[d, "n"] & \mathcal{F}_{TG}(L) \arrow[r, "\mathcal{F}_{TG}(r)"] \arrow[d, "\mathcal{F}_{TG}(m)"'] \arrow[dr, phantom, "(2)"] & \mathcal{F}_{TG}(R) \arrow[d, "\mathcal{F}_{TG}(n)"] & \mathcal{F}_{TG}(L) \arrow[r, "\mathcal{F}_{TG}(r)"] \arrow[d, "\mathcal{F}_{TG}(m)"'] \arrow[dr, phantom, "(3)"] & \mathcal{F}_{TG}(R) \arrow[d, "n'"] \\ G \arrow[r, "f"'] & G' & \mathcal{F}_{TG}(G) \arrow[r, "\mathcal{F}_{TG}(f)"'] & \mathcal{F}_{TG}(G') & \mathcal{F}_{TG}(G) \arrow[r, "f'"'] & H \end{tikzcd}$$

   i.e. given $r : L \to R$*,* $m : L \to G$ *in* $\mathbf{TripleGraphs_{TG}}$ *and pushout* (3) *with* H, n', f' *in* $\mathbf{Graphs}_{\mathcal{F}(TG)}$ *then there are unique* G', n, f *in* $\mathbf{TripleGraphs_{TG}}$*, s.t.* (1) *is pushout in* $\mathbf{TripleGraphs_{TG}}$ *with* $\mathcal{F}_{TG}(G') = H$*,* $\mathcal{F}_{TG}(n) = n'$*, and* $\mathcal{F}_{TG}(f) = f'$*.*
5. $\mathcal{F}_{TG}$ *preserves, reflects and creates pullbacks.*

Proof. **1.** It is straight forward to show that $\mathcal{F}$ is a well-defined functor $\mathcal{F} : \mathbf{TripleGraphs} \to \mathbf{Graphs}$. Given pushout (1) in **TripleGraphs** we have to show pushout (2) in **Graphs**.

$$\begin{tikzcd} L \arrow[r, "r"] \arrow[d, "m"'] \arrow[dr, phantom, "(1)"] & R \arrow[d, "n"] & \mathcal{F}(L) \arrow[r, "\mathcal{F}(r)"] \arrow[d, "\mathcal{F}(m)"'] \arrow[dr, phantom, "(2)"] & \mathcal{F}(R) \arrow[d, "\mathcal{F}(n)"] \\ G \arrow[r, "f"'] & G' & \mathcal{F}(G) \arrow[r, "\mathcal{F}(f)"'] & \mathcal{F}(G') \end{tikzcd}$$

Pushout (1) in **TripleGraphs** is equivalent to commutativity of double cube (3) in **Graphs** with three vertical pushouts, because pushouts in **TripleGraphs** are constructed componentwise as pushouts in **Graphs**.

$$(3)\qquad\begin{tikzcd}
SL \arrow[rrrr, "r_S"] \arrow[ddd, "m_S"] & & & & SR \arrow[ddd, "n_S"] & & & \\
& & CL \arrow[ull, "s_L"] \arrow[rrrr, "r_C"] \arrow[dr, "t_L"] \arrow[ddd, "m_C"] & & & & CR \arrow[ull, "s_R"] \arrow[dr, "t_R"] \arrow[ddd, "n_C"] & \\
& & & TL \arrow[rrrr, "r_T"] \arrow[ddd, "m_T"] & & & & TR \arrow[ddd, "n_T"] \\
SG \arrow[rrrr, "f_S"] & & & & SG' & & & \\
& & CG \arrow[ull, "s_G"] \arrow[rrrr, "f_C"] \arrow[dr, "t_G"] & & & & CG' \arrow[ull, "s_{G'}"] \arrow[dr, "t_{G'}"] & \\
& & & TG \arrow[rrrr, "f_T"] & & & & TG'
\end{tikzcd}$$

Diagram (2) is pushout in **Graphs** iff the V- and E-components are pushouts in **Sets**. In the following we show this for the E-component, while the proof for the V-component is similar and even simpler, because we have no $Links_S$- and $Link_T$-parts. The E-component of (2) is shown in diagram (4) in **Sets**.

$$\begin{tikzcd}
\boxed{\begin{array}{c}\mathcal{F}(L)_E\\ = SL_E + CL_E + TL_E\\ +Links_S(L) + Link_T(L)\end{array}} \arrow[r, "r_{S,E}+r_{C,E}+r_{T,E}", "+r_{LS}+r_{LT}"'] \arrow[d, "\begin{array}{c}m_{S,E}+m_{C,E}+m_{T,E}\\ +m_{LS}+m_{LT}\end{array}"'] \arrow[dr, phantom, "(4)"] & \boxed{\begin{array}{c}\mathcal{F}(R)_E\\ = SR_E + CR_E + TR_E\\ +Links_S(R) + Link_T(R)\end{array}} \arrow[d, "\begin{array}{c}n_{S,E}+n_{C,E}+n_{T,E}\\ +n_{LS}+n_{LT}\end{array}"] \\
\boxed{\begin{array}{c}\mathcal{F}(G)_E\\ = SG_E + CG_E + TG_E\\ +Links_S(G) + Link_T(G)\end{array}} \arrow[r, "f_{S,E}+f_{C,E}+f_{T,E}", "+f_{LS}+f_{LT}"'] & \boxed{\begin{array}{c}\mathcal{F}(G')_E\\ = SG'_E + CG'_E + TG'_E\\ +Links_S(G') + Link_T(G')\end{array}}
\end{tikzcd}$$

Diagram (4) is the disjoint union of five diagrams, where the S-, C- and T-part are the E-components of corresponding **Graph**-pushouts in (3) and hence, pushouts in **Sets**. We will show that the $Links_S$-part (and similar the $Link_T$-part) is pushout in **Sets**. Since coproducts of pushouts are again pushouts (in any category) (4) is a pushout in **Sets**. First of all, (5) commutes, because $\mathcal{F}$ is a functor and (5) is the LS-part of (2). In order to show the universal properties we assume to have $h_1 : Links_S(R) \to X$ and $h_2 : Links_S(G) \to X$ with $h_1 \circ r_{LS} = h_2 \circ m_{LS}$ and we have to construct a unique $h : Links_S(G') \to X$ such that (6) and (7) commute.

$$\begin{tikzcd}
Links_S(L) \arrow[r, "r_{LS}"] \arrow[d, "m_{LS}"] \arrow[dr, phantom, "(5)"] & Links_S(R) \arrow[d, "n_{LS}"] \arrow[ddr, bend left, "h_1"] & \\
Links_S(G) \arrow[r, "f_{LS}"] \arrow[drr, bend right, "h_2"'] & Links_S(G') \arrow[dr, dotted, "h"] \arrow[r, phantom, "(6)"] \arrow[d, phantom, "(7)"] & \\
& & X
\end{tikzcd}$$

Given $(x, y) \in Links_S(G')$ we have $x \in CG'$ and $y \in SG'$ with $s_{G'}(x) = y$ (where we omit the subindex V). Since CG' in (3) is pushout object we have either $x_1 \in CR$ with $n_C(x_1) = x$ or $x_2 \in CG$ with $f_C(x_2) = x$ leading to $h(x, y)$ defined by

$$h(x,y) = \begin{cases} h_1(x_1, y_1), & \text{for } n_C(x_1) = x \text{ and } y_1 = s_R(x_1) \\ h_2(x_2, y_2), & \text{for } f_C(x_2) = x \text{ and } y_2 = s_G(x_2). \end{cases}$$

$h_1(x_1, y_1)$ is well-defined and (6) commutes, because $(x_1, y_1) \in Links_S(R)$ and $n_S(y_1) = n_S \circ s_R(x_1) = s_{G'}(x) = y$ implies $h \circ n_{LS}(x_1, y_1) = h(n_C(x_1), n_S(y_1)) = h(x, y)$. Similar $h_2(x_2, y_2)$ is well-defined and (7) commutes. It remains to show that h is well-defined. For this purpose it is sufficient to show that for $x = n_C(x_1) = f_C(x_2)$ with $y_1 = s_R(x_1)$ and $y_2 = s_G(x_2)$ we can show $h_1(x_1, y_1) = h_2(x_2, y_2)$. Pushout of the C-component and $n_C(x_1) = f_C(x_2)$ imply existence of $x_0 \in CL$ with $r_C(x_0) = x_1$ and $m_C(x_0) = x_2$, if r_C or n_C is injective. If both are noninjective we have a chain $x_{01}, \ldots, x_{0n}$ connecting x_1 and x_2

and the proof is similar. In the injective case we have $y_0 = s_L(x_0)$ with $(x_0, y_0) \in Links_S(L)$ and $r_{LS}(x_0, y_0) = (r_C(x_0), r_S(y_0)) = (x_1, y_1)$ and similar $m_{LS}(x_0, y_0) = (x_2, y_2)$. Using $h_1 \circ r_{LS} = h_2 \circ m_{LS}$ this implies $h_1(x_1, y_1) = h_1 \circ r_{LS}(x_0, y_0) = h_2 \circ m_{LS}(x_0, y_0) = h_2(x_2, y_2)$.

2. $\mathcal{F}$: **TripleGraphs** $\rightarrow$ **Graphs** defines $\mathcal{F}_{TG}$: **TripleGraphs**$_{TG}$ $\rightarrow$ **Graphs**$_{\mathcal{F}(TG)}$ because for each $(G, t : G \rightarrow TG)$ in **TripleGraphs**$_{TG}$ we have $(\mathcal{F}(G), \mathcal{F}(t) : \mathcal{F}(G) \rightarrow \mathcal{F}(TG))$ in **Graphs**$_{\mathcal{F}(TG)}$ and for each morphism $f : (G, t) \rightarrow (G', t')$ with $f : G \rightarrow G'$ and $t' \circ f = t$ we have $\mathcal{F}(f) : (\mathcal{F}(G), \mathcal{F}(t)) \rightarrow (\mathcal{F}(G'), \mathcal{F}(t'))$ with $\mathcal{F}(f) : \mathcal{F}(G) \rightarrow \mathcal{F}(G')$ and $\mathcal{F}(t') \circ \mathcal{F}(f) = \mathcal{F}(t)$.

3. First we show that $\mathcal{F}_{TG}$ is injective on objects. By construction of $\mathcal{F}(TG)$ we have (in slight abuse of notation) $\mathcal{F}(TG) = STG + CTG + TTG + Links_S(TG) + Link_T(TG)$ with $s\mathcal{F}(TG)(x, y) = x$ and $t\mathcal{F}(TG)(x, y) = y$ and $CTG_E = \emptyset$ and corresponding coproduct embeddings in **Graphs** respectively **Sets**. For (G, t) in **TripleGraphs**$_{TG}$ with $t : G \rightarrow TG$ we have the following pullbacks $(1) - (3)$ in **Graphs** and $(4) - (5)$ in **Sets**, because $\mathcal{F}(t) = t_S + t_C + t_T + t_{LS} + t_{LT}$.

$$\begin{tikzcd}
SG \arrow[r, hook] \arrow[d, "t_S"'] \arrow[dr, phantom, "(1)"] & \mathcal{F}(G) \arrow[d, "\mathcal{F}(t)"] \arrow[dr, phantom, "(2)"] & TG \arrow[l, hook'] \arrow[d, "t_T"] & CG \arrow[r, hook] \arrow[d, "t_C"'] \arrow[dr, phantom, "(3)"] & \mathcal{F}(G) \arrow[d, "\mathcal{F}(t)"] \\
STG \arrow[r, hook] & \mathcal{F}(TG) & TTG \arrow[l, hook'] & CTG \arrow[r, hook] & \mathcal{F}(TG)
\end{tikzcd}$$

$$\begin{tikzcd}
Links_S(G) \arrow[r, hook] \arrow[d, "t_{LS}"'] \arrow[dr, phantom, "(4)"] & \mathcal{F}(G)_E \arrow[d, "\mathcal{F}(t)_E"] \arrow[dr, phantom, "(5)"] & Link_T(G) \arrow[l, hook'] \arrow[d, "t_{LT}"] \\
Links_S(TG) \arrow[r, hook] & \mathcal{F}(TG)_E & Link_T(TG) \arrow[l, hook']
\end{tikzcd}$$

These distinguished pullback constructions with inclusions in the upper and lower rows determine completely (G, t) with $G = (SG \overset{s_G}{\leftarrow} CG \overset{t_G}{\rightarrow} TG)$ and $CG_E = \emptyset$, where for each $e \in Links_S(G) \subseteq \mathcal{F}(G)_E$ with $s\mathcal{F}(G)(e) = x$ and $t\mathcal{F}(G)(e) = y$ we have $s_{G,V}(x) = y$. Vice versa, for each $x \in CG_V, y \in SG_V$ with $s_{G,V}(x) = y$ we have $e = (x, y) \in Links_S(G)$. Similarly, $Link_T(G)$ completely determines $t_{G,V}$, while $s_{G,E}$ and $t_{G,E}$ are empty. This implies for $(G, t), (G', t') \in$ **TripleGraphs**$_{TG}$ with $(\mathcal{F}(G), \mathcal{F}(t)) = (\mathcal{F}(G'), \mathcal{F}(t'))$ already $(G, t) = (G', t')$ and hence injectivity of $\mathcal{F}_{TG}$ on objects. $\mathcal{F}_{TG}$ creates morphisms by Lemma 1. Finally $A \cong B$ implies $\mathcal{F}_{TG}(A) \cong \mathcal{F}_{TG}(B)$, because $\mathcal{F}_{TG}$ is a functor. Vice versa, $\mathcal{F}_{TG}(A) \cong \mathcal{F}_{TG}(B)$ implies isomorphisms $m'_1 : \mathcal{F}_{TG}(A) \overset{\sim}{\rightarrow} \mathcal{F}_{TG}(B)$ and $m'_2 : \mathcal{F}_{TG}(B) \overset{\sim}{\rightarrow} \mathcal{F}_{TG}(A)$ leading to unique morphisms $m_1 : A \rightarrow B$ and $m_2 : B \rightarrow A$ with $\mathcal{F}_{TG}(m_1) = m'_1$ and $\mathcal{F}_{TG}(m_2) = m'_2$, because $\mathcal{F}_{TG}$ creates morphisms. Finally, $m_2 \circ m_1 = id_A$ (and similar $m_1 \circ m_2 = id_B$) and hence $A \cong B$, because $\mathcal{F}_{TG}(m_2 \circ m_1) = \mathcal{F}_{TG}(m_2) \circ \mathcal{F}_{TG}(m_1) = m'_2 \circ m'_1 = id_{\mathcal{F}_{TG}(A)} = \mathcal{F}_{TG}(id_A)$.

4. $\mathcal{F}_{TG}$: **TripleGraphs**$_{TG}$ $\rightarrow$ **Graphs**$_{\mathcal{F}(TG)}$ preserves pushout (1), because $\mathcal{F}$: **TripleGraphs** $\rightarrow$ **Graphs** preserves pushouts by part 1 and pushouts in **TripleGraphs**$_{TG}$ and **Graphs**$_{\mathcal{F}(TG)}$ are based on those in **TripleGraphs** and **Graphs**, respectively. Vice versa, if (2) is pushout in **TripleGraphs**$_{TG}$ and let $(1')$ be pushout in **TripleGraphs**$_{TG}$ then also $(2')$ is pushout in **Graphs**$_{\mathcal{F}(TG)}$.

$$\begin{tikzcd}
L \arrow[r, "r"] \arrow[d, "m"'] \arrow[dr, phantom, "(1')"] & R \arrow[d, "n'"] & \mathcal{F}_{TG}(L) \arrow[r, "\mathcal{F}_{TG}(r)"] \arrow[d, "\mathcal{F}_{TG}(m)"'] \arrow[dr, phantom, "(2')"] & \mathcal{F}_{TG}(R) \arrow[d, "\mathcal{F}_{TG}(n')"] \\
G \arrow[r, "f'"'] & G'' & \mathcal{F}_{TG}(G) \arrow[r, "\mathcal{F}_{TG}(f')"'] & \mathcal{F}_{TG}(G'')
\end{tikzcd}$$

Uniqueness of pushouts implies $\mathcal{F}_{TG}(G') \cong \mathcal{F}_{TG}(G'')$ and hence $G' \cong G''$ by part 3, where $G' \cong G''$ is compatible with n, n' and f, f', respectively, showing that also (1) is pushout. Finally, we show that $\mathcal{F}_{TG}$ creates pushouts. Given $r : L \rightarrow R$ and $m : L \rightarrow G$ in **TripleGraphs**$_{TG}$ and H pushout in (2) of $\mathcal{F}_{TG}(r), \mathcal{F}_{TG}(m)$.

$$\begin{tikzcd}
L \arrow[r, "r"] \arrow[d, "m"'] \arrow[dr, phantom, "(1)"] & R \arrow[d, "n"] \arrow[ddr, bend left, "n''"] & \\
G \arrow[r, "f"'] \arrow[drr, bend right, "f''"'] & G' \arrow[dr, "\sim"] & \\
& & G''
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathcal{F}_{TG}(L) \arrow[r, "\mathcal{F}_{TG}(r)"] \arrow[d, "\mathcal{F}_{TG}(m)"'] \arrow[dr, phantom, "(2)"] & \mathcal{F}_{TG}(R) \arrow[d, "n'"] \arrow[ddr, bend left, "\mathcal{F}_{TG}(n'')"] & \\
\mathcal{F}_{TG}(G) \arrow[r, "f'"'] \arrow[drr, bend right, "\mathcal{F}_{TG}(f'')"'] & H \arrow[dr, "\sim"] & \\
& & \mathcal{F}_{TG}(G'')
\end{tikzcd}$$

Let G'' with n'', f'' be pushout of r and m. Then $\mathcal{F}_{TG}(G'')$ is also pushout and hence, $H \cong \mathcal{F}_{TG}(G'')$. According to the construction in part 3 we can construct G' in $\mathbf{TripleGraphs}_{TG}$ with $\mathcal{F}_{TG}(G') = H$. Note that in this construction $s_{G',V}$ and $t_{G',V}$ are functions defined by $Link_S(G') = type_H^{-1}(Link_S(TG))$ and $Link_T(G') = type_H^{-1}(Link_T(TG))$, respectively, because $H \cong \mathcal{F}_{TG}(G'')$ and this functional property holds for $\mathcal{F}_{TG}$ by construction. Hence, we have $n' : \mathcal{F}_{TG}(R) \to \mathcal{F}_{TG}(G')$ and $f' : \mathcal{F}_{TG}(G) \to \mathcal{F}_{TG}(G')$ and by part 3 we have unique $n : R \to G'$ and $f : G \to G'$ with $\mathcal{F}_{TG}(n) = n'$ and $\mathcal{F}_{TG}(f) = f'$. Now reflection of pushouts implies that (1) is pushout in $\mathbf{TripleGraphs}_{TG}$ with $G' \cong G''$.

5. Similar to part 1 we can show that $\mathcal{F}$ preserves pullbacks, because the $Link_S$- and $Link_T$-diagrams can be shown to be pullbacks and the disjoint union of pullbacks in **Sets** is again a pullback. This allows to show preservation, reflection and creation of pullbacks similar to those of pushouts in part 4. □

Lemma 1. Injectivity of $\mathcal{F}_{TG}$ on morphisms: *Given $m' : \mathcal{F}(L) \to \mathcal{F}(G)$ in $\mathbf{Graphs}_{\mathcal{F}(TG)}$ then there is a unique $m : L \to G$ in $\mathbf{TripleGraphs}_{TG}$ with $\mathcal{F}(m) = m'$. This especially implies injectivity of $\mathcal{F}_{TG}$ on morphisms.*

Proof. Using $\mathcal{F}(L) = SL + CL + TL + Link_S(L) + Link_T(L)$ and $\mathcal{F}(G) = SG + CG + TG + Link_S(G) + Link_T(G)$ we obtain unique $m_S : SL \to SG, m_C : CL \to CG, m_T : TL \to TG, m_{LS} : Link_S(L) \to Link_S(G), m_{LT} : Link_T(L) \to Link_T(G)$, which are type compatible. For example in the case of m_S SG and SL are given by pullbacks $SG = \mathcal{F}(type_G)^{-1}(STG) \subseteq \mathcal{F}(G)$ and $SL = \mathcal{F}(type_L)^{-1}(STG) \subseteq \mathcal{F}(L)$ and m_S is the unique induced morphism leading to type compatibility.

$$\begin{tikzcd}
SL \arrow[rr, hook] \arrow[dd, "type_{L,S}"'] \arrow[dr, dotted, "m_S"] & & \mathcal{F}(L) \arrow[dr, "m'"] \arrow[dd, "\mathcal{F}(type_L)"] & \\
& SG \arrow[rr, hook] \arrow[dl, "type_{G,S}"] & & \mathcal{F}(G) \arrow[dl, "\mathcal{F}(type_G)"] \\
STG \arrow[rr, hook] & & \mathcal{F}(TG) &
\end{tikzcd}$$

The triple graph morphism $m = (m_S, m_C, m_T)$ is given by the following diagram, where commutativity of (1) is shown below for the V-component. It is trivial for the E-component, because $CL_E = CG_E = \emptyset$.

$$\begin{tikzcd}
SL \arrow[d, "m_S"'] \arrow[dr, phantom, "(1)"] & CL \arrow[l, "s_L"'] \arrow[r, "t_L"] \arrow[d, "m_C"] \arrow[dr, phantom, "(2)"] & TL \arrow[d, "m_T"] \\
SG & CG \arrow[l, "s_G"] \arrow[r, "t_G"'] & TG
\end{tikzcd}$$

This construction implies $\mathcal{F}(m) = m_S + m_C + m_T + m_{LS} + m_{LT} = m'$, because $m' : \mathcal{F}(L) \to \mathcal{F}(G)$ is uniquely determined by the S-, C-, T-, $Link_S$- and $Link_T$-components. This also implies uniqueness of m with $\mathcal{F}(m) = m'$. For commutativity of (1) (and similarly for (2)) we use the assumption that $m' : \mathcal{F}(L) \to \mathcal{F}(G)$ is a graph morphism in $\mathbf{Graphs}_{\mathcal{F}(TG)}$ and hence also in **Graphs**. This implies commutativity of (3).

$$\begin{tikzcd}
\boxed{\begin{array}{c}\mathcal{F}(L)_E \\ = SL_E + CL_E + TL_E \\ +Link_S(L) + Link_T(L)\end{array}} \arrow[r, shift left, "s\mathcal{F}(L)"] \arrow[r, shift right, "t\mathcal{F}(L)"'] \arrow[d, "\begin{array}{c}m_{S,E} + m_{C,E} + m_{T,E} \\ +m_{LS} + m_{LT}\end{array}"'] \arrow[dr, phantom, "(3)"] & \boxed{\begin{array}{c}\mathcal{F}(L)_V \\ = SL_V + CL_V + TL_V\end{array}} \arrow[d, "\begin{array}{c}n_{S,E} + n_{C,E} + n_{T,E} \\ +n_{LS} + n_{LT}\end{array}"] \\
\boxed{\begin{array}{c}\mathcal{F}(G)_E \\ = SG_E + CG_E + TG_E \\ +Link_S(G) + Link_T(G)\end{array}} \arrow[r, shift left, "s\mathcal{F}(G)"] \arrow[r, shift right, "t\mathcal{F}(G)"'] & \boxed{\begin{array}{c}\mathcal{F}(G)_V \\ = SG_V + CG_V + TG_V\end{array}}
\end{tikzcd}$$

For all $(x, y) \in Link_S(L)$ we have $m_V \circ s\mathcal{F}(L)(x, y) = s\mathcal{F}(G) \circ m_E(x, y)$. This implies $s\mathcal{F}(G)(m_{LS}(x, y)) = m_{C,V}(x)$, because we have $m_V \circ s\mathcal{F}(L)(x, y) = m_V(x) = m_{C,V}(x)$ and $s\mathcal{F}(G) \circ m_E(x, y) = s\mathcal{F}(G)(m_{LS}(x, y))$. Similar $t\mathcal{F}(G)(m_{LS}(x, y)) = m_{S,V}(y)$, which implies $m_{LS}(x, y) = (m_{C,V}(x), m_{S,V}(y))$.

Now, we show that the V-component of (1) commutes. Given $x \in CL_V$ we have $s_{L,V}(x) = y$ and $(x, y) \in Link_S(L)$. For $(x, y) \in Link_S(L)$ we have $m_{LS}(x, y) = (m_{C,V}(x), m_{S,V}(y)) \in Link_S(G')$. This implies $s_{G,V} \circ m_{C,V}(x) = m_{S,V}(y) = m_{S,V} \circ s_{L,V}(x)$. Finally, $\mathcal{F}_{TG}(f) = \mathcal{F}_{TG}(g)$ implies $f = g$ by the uniqueness of the creatioin property and hence, injectivity of $\mathcal{F}_{TG}$. □

Remark 3. *The typed flattening construction $\mathcal{F}$: **TripleGraphs**$_{TG}$ $\rightarrow$ **Graphs**$_{\mathcal{F}(TG)}$ is in general not surjective and hence defines no isomorphism or equivalence of categories **TripleGraphs**$_{TG}$ and **Graphs**$_{\mathcal{F}(TG)}$. There are graphs $(H, type_H)$ in **Graphs**$_{\mathcal{F}(TG)}$ which are not functional in the sense that for $TG = (STG \leftarrow CTG \rightarrow TTG)$ one node in $CH = type_H^{-1}(CTG)$ is connected in H with zero or more than one node in $SH = type_H^{-1}(STG)$ respectively in $TH = type_H^{-1}(TTG)$. In this case we do not obtain graph morphisms $sH : CH \rightarrow SH$ respectively $tH : CH \rightarrow TH$ and hence no triple graph $(SH \leftarrow CG \rightarrow TH)$. Triple graph applications exist in literature where plain graphs are used which have multiple edges connecting the same correspondence node to various elements of the source and target language [11, 13]. This approach does not correspond to pure morphism-based triple graphs and hence is not covered by our translation construction.*

Using Fact 3 above, the flattening functor can be extended to translate triple graph grammars.

Definition 10. Translation of Triple Graph Grammars: *Given a triple graph grammar $TGG = (TG, S, TR)$ with triple type graph TG as above, start graph S and triple rules $tr : L \rightarrow R$ in **TripleGraphs**$_{TG}$, then the translation $\mathcal{F}(TGG)$ of TGG is the graph grammar $\mathcal{F}(TGG) = (\mathcal{F}(TG), \mathcal{F}(S), \mathcal{F}(TR))$ with type graph $\mathcal{F}(TG)$, start graph $\mathcal{F}(S)$, and rules $\mathcal{F}(TR) = \{\mathcal{F}(tr) : \mathcal{F}(L) \rightarrow \mathcal{F}(R) \,|\, (tr : L \rightarrow R) \in TR\}$.*

Theorem 2. Translation and Creation of Triple Graph Transformations: *Given a triple graph grammar $TGG = (TG, S, TR)$ with translation $\mathcal{F}(TGG) = (\mathcal{F}(TG), \mathcal{F}(S), \mathcal{F}(TR))$ then*

1. *Each triple graph transformation $trafo$:*
   $S \xrightarrow{tr_1, m_1} G_1 \Rightarrow \ldots \xrightarrow{tr_n, m_n} G_n$ in TGG can be translated into a flattened graph transformation
   $\mathcal{F}(trafo) : \mathcal{F}(S) \xrightarrow{\mathcal{F}(tr_1), \mathcal{F}(m_1)} \mathcal{F}(G_1) \Rightarrow \ldots \xrightarrow{\mathcal{F}(tr_n), \mathcal{F}(m_n)} \mathcal{F}(G_n)$ in $\mathcal{F}(TGG)$.
2. *Vice versa, each graph transformation $trafo'$:*
   $\mathcal{F}(S) \xrightarrow{\mathcal{F}(tr_1), m'_1} G'_1 \Rightarrow \ldots \xrightarrow{\mathcal{F}(tr_n), m'_n} G'_n$ in $\mathcal{F}(TGG)$ creates a unique (up to isomorphism) triple graph transformation $trafo : S \xrightarrow{tr_1, m_1} G_1 \Rightarrow \ldots \xrightarrow{tr_n, m_n} G_n$ in TGG with $\mathcal{F}(trafo) = trafo'$, i.e. $\mathcal{F}(m_i) = m'_i$ and $\mathcal{F}(G_i) = G'_i$ for $i = 1 \ldots n$.

Proof. Follows from Fact 3.4.

□

5 Relationship of Model Transformation Concepts

After defining a translation from model transformations using triple graphs to those using plain graphs and showing properties of the translation, we now elaborate properties of model transformations from a more general point of view and again the results of previous sections can be applied to this more abstract setting.

Typed model transformations usually work on integrated models in the translation phase and finally restrict the model to elements of the target language. Thus, the integrated type graph $(TG_S \rightarrow TG_{ST} \leftarrow TG_T)$ consists of all elements of the source and target language and possibly further correspondence structure elements. By $t_S^{>}(G)$ we denote a retyping of a source model G typed over TG_S to a model typed over the integrated type graph TG_{ST} given by the embedding $TG_S \rightarrow TG_{ST}$. Furthermore, $t_T^{<}(G)$ specifies a restriction of G typed over TG_{ST} to a model G' typed over TG_T only, which can be constructed as pullback of $G \rightarrow TG_{ST} \leftarrow TG_T$. In the following general concept graph transformation systems may be equipped with a control condition restricting the possible transformations. Such conditions will be explained exemplarily thereafter.

Definition 11. General Concept of Model Transformations based on graph transformation: *Let* **GRAPHS** *be plain graphs* **Graphs** *or triple graphs* **TripleGraphs***.*

1. *Given visual languages $VL_S \subseteq \mathbf{GRAPHS}_{TG_\mathbf{S}}$ and $VL_T \subseteq \mathbf{GRAPHS}_{TG_\mathbf{T}}$ a* model transformation *$MT : VL_S \Rrightarrow VL_T$ from VL_S to VL_T is given by $MT = (VL_S, VL_T, TG_{ST}, t_S, t_T, GTS)$ where TG_{ST} is an integrated type graph with injective type graph morphisms $(TG_S \overset{t_S}{\rightarrow} TG_{ST} \overset{t_T}{\leftarrow} TG_T)$, and GTS a graph transformation system with non-deleting rules R typed over TG_{ST} and a control condition for GTS-transformations.*
2. *A* model transformation sequence *via MT, short MT-sequence, is given by $(G_S, G_1 \Rightarrow^* G_n, G_T)$, where $G_S \in VL_S, G_T \in VL_T$ and $G_1 \Rightarrow^* G_n$ is a GTS-transformation satisfying the control condition of GTS with $G_1 = t_S^{>}(G_S)$ and $G_T = t_T^{<}(G_n)$, defined above.*
3. *The* model transformation relation *$MT_R \subseteq VL_S \times VL_T$ defined by MT is given by: $(G_S, G_T) \in MT_R \Leftrightarrow \exists\ MT - sequence\ (G_S, G_1 \Rightarrow^* G_n, G_T)$.*
4. *$MT : VL_S \Rrightarrow VL_T$ is called*
   (a) *syntactically correct, if for all GTS-transformations $G_1 \Rightarrow^* G_n$ satisfying the control condition with $G_1 = t_S^{>}(G_S)$ and $G_S \in VL_S$ we have $G_T = t_T^{<}(G_n) \in VL_T$*
   (b) *functional, if MT_R is right unique*
   (c) *total, if MT_R is left total*
   (d) *surjective, if MT_R is right total*

Most examples of model transformations based on plain graph transformation considered in the literature fit into this general concept. A typical example is the model transformation $SC2PN$ from state charts to Petri nets (see Chapter 14 of [9] with restriction construction instead of deleting rules): The control condition is given by layers, where the rules with negative application conditions are applied as long as possible in one layer, and suitable termination criteria have to be satisfied, before switching to the next layer. But also model transformations based on triple rules fit into this concept as shown for forward rules in the next example. Of course this can be done in a similar way for backward rules.

Example 8. Model Transformation based on Forward Rules: *Given triple rules TR with source rules TR_S and forward rules TR_F defining the triple graph language VL_S and VL_T. Let TR be typed over $TG = (TG_S \leftarrow TG_C \rightarrow TG_T)$ with $t_S : (TG_S \leftarrow \emptyset \rightarrow \emptyset) \rightarrow TG$ and $t_T : (\emptyset \leftarrow \emptyset \rightarrow TG_T) \rightarrow TG$ type graph embeddings and $GTS = TR_F$ with "source consistency" as control condition, i.e. $G_1 \xRightarrow{tr_F^*} G_n$ satisfies the control condition, if it is source consistent. Then the model transformation $MT : VL_S \Rrightarrow VL_T$ based on forward rules is given by $MT = (VL_S, VL_T, TG, t_S, t_T, TR_F)$*

As pointed out before, by source consistency the application of the forward rules is controlled by the source sequence, which generates the given source model. Model transformations based on forward rules are source consistent, which additionally ensures syntactical correctness as well as totality and surjectivity stated below. Moreover, functional behaviour can be characterized.

Theorem 3. Correctness Properties of Model Transformation based on Forward Rules: *Let $MT : VL_S \Rrightarrow VL_T$ with $MT = (VL_S, VL_T, TG_{ST}, t_S, t_T, TR_F)$ be a model transformation based on forward rules TR_F then*

- *each model transformation sequence $(G_S,\ G_1 \xRightarrow{tr_F^*} G_n,\ G_T)$ in the sense of Def. 8 is a model transformation sequence in the sense of Def. 11 and vice versa.*
- *Moreover, MT is syntactically correct, total and surjective.*
- *MT is functional if and only if the language VL of $(\emptyset, TR)$ has the S-T projection property, i.e. for all $G, G' \in VL$ we have $proj_S(G) = proj_S(G') \Rightarrow proj_T(G) = proj_T(G')$.*

Proof. By Fact 1 each model transformation sequence in the sense of Def. 8 is also one in the sense of Def. 11 and vice versa, because $G_1 = t_S^{>}(G_S)$ is equivalent to G_1 typed over $(TG_S \leftarrow \emptyset \rightarrow \emptyset)$ and $proj_S(G_1) = G_S$ and $G_T = t_T^{<}(G_n)$ is equivalent to $proj_T(G_n) = G_T$.

MT is syntactically correct, because we have for each source consistent $G_1 \xRightarrow{tr_F^*} G_n$ with $G_1 = t_S^{>}(G_S)$ and $G_S \in VL_S$ already $G_T = t_T^{<}(G_n) \in VL_T$ by Fact 1 and the equivalences above.

MT is total, because for each $G_S \in VL_S$ we have by definition $G \in VL$ with $proj_S(G) = G_S$. $G \in VL$ implies $\emptyset \xRightarrow{tr^*} G$ and hence, by Thm. 1 a match consistent sequence $\emptyset \xRightarrow{tr_S^*} G_1 \xRightarrow{tr_F^*} G$. This implies a model transformation sequence $(G_S,\ G_1 \xRightarrow{tr_F^*} G, G_T)$ with $proj_S(G_1) = proj_S(G) = G_S$ and hence, $(G_S, G_T) \in MT_R$ which implies that MT is total. Similarly we find for each $G_T \in VL_T$, triple graphs $G \in VL$ and $G_S = proj_S(G)$ with $(G_S, G_T) \in MT_R$ showing that MT is surjective.

Assume now that we have the S-T projection property and MT-sequences $(G_S,\ G_1 \xRightarrow{tr_F^*} G_n, G_T)$ and $(G_S,\ G_1 \xRightarrow{tr'^*_F} G'_m, G'_T)$ we have to show $G_T = G'_T$. By source consistency we have match consistent sequences $\emptyset \xRightarrow{tr_S^*} G_1 \xRightarrow{tr_F^*} G_n$ with $proj_S(G_n) = proj_S(G_1) = G_S$ and $\emptyset \xRightarrow{tr'^*_S} G'_1 \xRightarrow{tr'^*_F} G'_m$ with $proj_S(G'_m) = G_S$. By Thm. 1 we have $\emptyset \xRightarrow{tr^*} G_n$, $\emptyset \xRightarrow{tr'^*} G'_m$ and hence $G_n, G'_m \in VL$ with $proj_S(G_n) = G_S = proj_S(G'_m)$. This implies $G_T = proj_T(G_n) = proj_T(G'_m) = G'_T$ by S-T projection property. Similar we can show that MT being functional implies the S-T projection property. □

Example 9. Non-functional Model Transformation *Consider a triple transformation sequence based on the triple rules in Fig. 6, where at first rule Class2Table is applied, generating a Class connected to a Table. In a second step, rule Subclass2Table is applied, inserting a subclass of the existing Class node and connecting the new Class also with the existing Table node. In the third step, yet another Class node is created as subclass of the existing subclass, also connected to the Table node. This 3-step triple transformation can be divided into three target steps, where the Table node is created in the first step, and nothing happens in the second and third step, and three backward transformation steps based on the backward rules in Fig. 7, which are generated from the triple rules Class2Table and Subclass2Table according to Def. 6.*

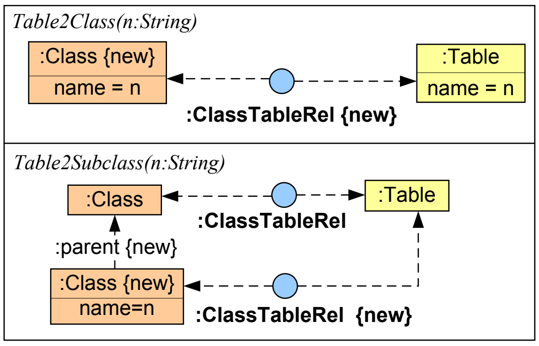

Figure 7: Backward rules generated from the triple rules $Class2Table$ and $Subclass2Table$

The first backward transformation step creates a Class node and connects it to the existing Table node. The second backward step creates a second Class node as subclass of the first one *and connects it to the Table node. For the third backward step, there are now two matches possible (see Fig. 8): either the third new Class node becomes a subclass of the existing superclass (graph G), or it becomes a subclass of the existing subclass (graph G').*

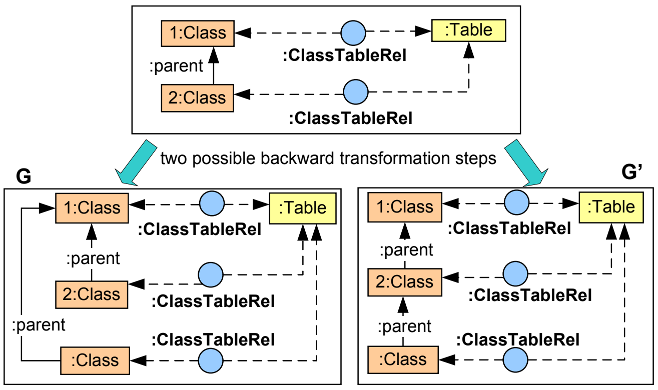

Figure 8: Non-functional (backward) model transformation

Both transformation sequences are match-consistent and target-consistent (the dual property to source-consistent), but they are not functional since different results are possible when applying the same rule. In particular, the T-S *projection property (corresponding to the* S-T *projection property in Thm. 3) is violated because we have $proj_T(G) = proj_T(G')$ but $proj_S(G) \neq proj_S(G')$.*

Finally, we present the translation of model transformation MT based on forward rules into plain model transformation $\mathcal{F}(MT)$ and show that the correctness properties of MT are preserved for $\mathcal{F}(MT)$.

Definition 12. Translation of Model Transformation based on Forward Rules: *Given $MT = (VL_S, VL_T, TG, t_S, t_T, TR_F)$ as in Example 8 then the translated model transformation $\mathcal{F}(MT)$ is a plain model transformation defined by $\mathcal{F}(MT) = (\mathcal{F}(VL_S), \mathcal{F}(VL_T), \mathcal{F}(TG), \mathcal{F}(t_S), \mathcal{F}(t_T), \mathcal{F}(TR_F))$ where $\mathcal{F} : \mathbf{TripleGraphs}_{TG} \to \mathbf{Graph}_{\mathcal{F}(TG)}$ is the typed flattening functor (see Def. 10) and a graph transformation sequence $trafo' : G'_0 \xRightarrow{\mathcal{F}(tr_{1,F}),m'_1} G'_1 \Rightarrow \ldots \xRightarrow{\mathcal{F}(tr_{n,F}),m'_n} G'_n$ satisfies the plain control condition, if $G'_0 = \mathcal{F}(G_0)$ and the uniquely created triple graph transformation $trafo : G_0 \xRightarrow{tr_{1,F},m_1} G_1 \Rightarrow \ldots \xRightarrow{tr_{n,F},m_n} G_n$ (by Thm. 2) is source consistent.*

Theorem 4. Properties of Translation *Given a model transformation $MT = (VL_S, VL_T, TG_{ST}, t_S, t_T, TR_F)$ based on forward rules TR_F of TR and $\mathcal{F}(MT)$ the translated plain model transformation then*

1. *there is a bijective correspondence between MT- and $\mathcal{F}(MT)$-model transformation sequences,*
2. *$\mathcal{F}(MT)$ is syntactically correct, total and surjective and*
3. *$\mathcal{F}(MT)$ being functional is equivalent to MT being functional.*

Proof. Given an MT-model transformation sequence $(G_S, G_1 \xRightarrow{tr_F^*} G, G_T)$ we obtain by Def. 12 the $\mathcal{F}(MT)$-model transformation sequence $(\mathcal{F}(G_S), \mathcal{F}(G_1) \xRightarrow{\mathcal{F}(tr_F^*)} \mathcal{F}(G_n), \mathcal{F}(G_T))$, because $\mathcal{F}(tr_F^*)$ satisfies the plain control condition and $t_S^{<}(G_S) = G_1$ implies $\mathcal{F}(t_S)^{>}(\mathcal{F}(G_S)) = \mathcal{F}(G_1)$ and $t_T^{<}(G_n) = G_T$ implies $\mathcal{F}(t_T)^{<}(\mathcal{F}(G_T)) = \mathcal{F}(G_n)$ because $\mathcal{F}$ preserves pullbacks by Fact 3. Vice versa, each $\mathcal{F}(MT)$-model transformation sequence creates a unique MT-model transformation sequence using again Def. 12 and the fact that $\mathcal{F}$ creates pushouts and pullbacks by Fact 3. Injectivity of $\mathcal{F}$ by Fact 3 implies that we have a bijective correspondence between MT- and $\mathcal{F}(MT)$-model transformation sequences. This implies by Thm. 3 that $\mathcal{F}(MT)$ is syntactically correct, total and surjective and $\mathcal{F}(MT)$ functional is equivalent to MT functional. □

6 Discussion

In this paper we address the problem that frequently TGGs and model transformations based on forward rules are implemented using plain graphs and transformations. In order to clarify how the results from the theory of TGGs do relate to corresponding plain graph representations, a formal translation from triple graphs and transformations according to [16] to plain graphs and transformations according to [9] is presented. Based on this translation functor, we relate properties of model transformations by forward rules to their corresponding model transformations in plain graphs and show in particular that the functor preserves syntactical correctness and functional behavior of model transformations.

Our main results show that for each model transformation based on forward rules there is an equivalent model transformation based on plain rules, defined by the translation functor. Analogously, such an equivalence exists for model transformation based on (target-consistent) backward rules. Hence, model transformations based on triple rules are bidirectional but not necessarily functional in one or both directions.

There are model transformations based on plain rules which are unidirectional and cannot be represented by equivalent model transformations based on triple rules.

The control criterion for forward rules is source consistency which is different from usual control criteria for plain rules, like layered graph transformation with termination criteria for rules with/without negative application conditions (NACs) [12] or graph constraints [7]. Note that NACs are not necessary for termination of forward transformations in our setting, since by source consistency the source sequence, which generates the given source model, controls the application of the forward rules.

Correctness properties are mainly given for triple case in Thm. 3 which are inherited by translated plain model transformations, but there remain the following open questions:

1. What are suitable syntactical criteria for functionality of model transformations?
2. Extending the approach to triple rules with NACs – what is the relationship between the source consistency control condition for forward rules on the one hand and NAC-consistency together with termination of model transformations on the other hand?

Furthermore, practical applications sometimes use plain graphs, which do not correspond to triple graphs according to [16], because morphisms between triple components cannot be defined accordingly. For instance edges and attributes occur in the component graph but have no connection to source and target model elements, thus morphisms of the triple graph cannot be total. These constructions are used to store information about the history and dependencies of executed integration steps, and thus they do not directly belong to the integrated model and can be stored separately. In other examples one node of the correspondence component may be connected to multiple nodes of the source resp. target component, which again does not correspond to triple graphs (e.g. [11, 13]). It remains unclear whether the theoretical results including those of [6] and [8] for triple graph transformations can be transferred to plain graph grammars, which do not directly correspond to TGGs.

References

[1] AGG Homepage. http://tfs.cs.tu-berlin.de/agg.

[2] N. Aschenbrenner and L. Geiger. Transforming scene graphs using Triple Graph Grammars - A practice report. In *Proc. Third International Workshop and Symposium on Applications of Graph Transformation with Industrial Relevance (AGTIVE 2007)*, Universität Kassel, Germany, October 2007.

[3] J. Bézivin, B. Rumpe, A. Schürr, and L. Tratt. Model transformations in practice workshop. In J.-M. Bruel, editor, *MoDELS Satellite Events*, volume 3844 of *Lecture Notes in Computer Science*, pages 120–127. Springer, 2005.

[4] J. de Lara and E. Guerra. Model View Management with Triple Graph Grammars. In A. Corradini et al., editor, *Proceedings of the Third International Conference on Graph Transformation (ICGT 2006)*, volume 4178 of *LNCS*, Natal, Brazil, 2006. Springer.

[5] J. de Lara and H. Vangheluwe. *AToM3: A Tool for Multi-formalism and Meta-Modelling*, 2007. http://atom3.cs.mcgill.ca/.

[6] H. Ehrig, K. Ehrig, C. Ermel, F. Hermann, and G. Taentzer. Information preserving bidirectional model transformations. In M. B. Dwyer and A. Lopes, editors, *Fundamental Approaches to Software Engineering*, volume 4422 of *LNCS*, pages 72–86. Springer, 2007.

[7] H. Ehrig, K. Ehrig, A. Habel, and K.-H. Pennemann. Theory of Constraints and Application Conditions: From Graphs to High-Level Structures . *Fundamenta Informaticae*, 74(1):135–166, 2006.

[8] H. Ehrig, K. Ehrig, and F. Hermann. From Model Transformation to Model Integration based on the Algebraic Approach to Triple Graph Grammars. In J. de Lara, C. Ermel, and R. Heckel, editors, *Workshop on Graph Transformation and Visual Modelling Techniques (GT-VMT'08)*. Electronic Communications of the EASST, 2008. accepted for publication.

[9] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation.* EATCS Monographs. Springer Verlag, 2006.

[10] H. Giese and R. Wagner. Incremental Model Synchronization and Transformation with Triple Graph Grammars. In O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, editors, *Proc. of the 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, LNCS, Genoa, Italy, October 2006. Springer.

[11] J. Greenyer and E. Kindler. Reconciling TGGs with QVT. In G. Engels, B. Opdyke, D. Schmidt, and F. Weil, editors, *10th International Conference on Model Driven Engineering Languages and Systems (MoDELS'07)*, volume 4735 of *LNCS*, pages 16–30. Springer, 2006.

[12] A. Habel, R. Heckel, and G. Taentzer. Graph Grammars with Negative Application Conditions. *Special issue of Fundamenta Informaticae*, 26(3,4):287–313, 1996.

[13] E. Kindler and R. Wagner. Triple graph grammars: Concepts, extensions, implementations, and application scenarios. Technical Report Technical Report tr-ri-07-284, Software Engineering Group, Department of Computer Science, University of Paderborn, 2007.

[14] Object Management Group. *OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2*, November 2007.

[15] OMG. *MOF QVT Final Adopted Specification (05-11-01). http://www.omg.org/docs/ptc/05-11-01.pdf*, 2005.

[16] A. Schürr. Specification of Graph Translators with Triple Graph Grammars. In G. Tinhofer, editor, *WG94 20th Int. Workshop on Graph-Theoretic Concepts in Computer Science*, volume 903 of *Lecture Notes in Computer Science*, pages 151–163, Heidelberg, 1994. Springer Verlag.

[17] Software Engineering Group, University of Paderborn. *Fujaba Tool Suite*, 2007. http://wwwcs.uni-paderborn.de/cs/ag-schaefer/Lehre/PG/Fujaba/projects/tgg/index.html.