Under consideration for publication in Math. Struct. in Comp. Science

Formal Analysis of Model Transformations Based on Triple Graph Grammars

FRANK HERMANN[1,2,†], HARTMUT EHRIG[1,†], ULRIKE GOLAS[3]
AND FERNANDO OREJAS[4,‡]

[1] *[frank,ehrig]@cs.tu-berlin.de, Institut für Softwaretechnik und Theoretische Informatik, Technische Universität Berlin, Germany*
[2] *Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg*
[3] *golas@zib.de, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Germany*
[4] *orejas@lsi.upc.edu, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Barcelona, Spain*

Received September 2011

Triple Graph Grammars (TGGs) are a well-established concept for the specification and execution of bidirectional model transformations within model driven software engineering. Their main advantage is an automatic generation of operational rules for forward and backward model transformations, which simplifies specification and enhances usability as well as consistency. This article presents several important results for analysing model transformations based on the formal categorical foundation of TGGs within the framework of attributed graph transformation systems.
In our first main result, we show that the crucial properties correctness and completeness are ensured for model transformations. In order to analyse functional behaviour, we generate a new kind of operational rules – called *forward translation rules*. We apply existing results for the analysis of local confluence for attributed graph transformation systems. As additional main results we provide sufficient criteria for the verification of functional behaviour as well as a necessary and sufficient condition for strong functional behaviour. In fact, these conditions imply polynomial complexity for the execution of the model transformation. Moreover, we analyse information and complete information preservation of model transformations, i.e. the problem whether a source model can be reconstructed (uniquely) from the target model computed by the model transformation. We illustrate the results for the well-known model transformation example from class diagrams to relational database models.

Contents

† The participation of these authors is supported by the DFG project BEHAVIOUR-GT.
‡ The participation of this author is supported by the MEC project FORMALISM (ref. TIN2007-66923).

1. Introduction

1.1. *Main Challenges for Model Transformations*

Model transformations are a key concept for modular and distributed model driven development. They are used thoroughly for model optimization and other forms of model evolution. Moreover, model transformations are used to map models between different domains in order to perform code generation or to apply analysis techniques. In this multi domain context, triple graph grammars have been applied in several case studies and they show a convenient combination of formal and intuitive specification abilities.

In this paper we consider a number of important properties for model transformations. More precisely, assuming that we have specified a class of transformations using a triple graph grammar, we study the following properties of forward transformations from the class of source models to the class of target models:

1 *Syntactical Correctness and Completeness:* Syntactical correctness of a transformation method means that if we can transform any source model G^S into a model G^T using the method, then the model G^T is a valid target model and, moreover, the pair (G^S, G^T) is consistent with respect to the specification of the model transformation provided by the triple graph grammar. Completeness, on the other hand, means that if for any consistent pair (G^S, G^T) according to the specification, then our transformation method will be able to build G^T from G^S.

2 *Functional and Strong Functional Behaviour:* Functional behaviour means that for each source model G^S each forward transformation starting with G^S leads to a unique valid target model G^T. Strong functional behaviour means, in addition, that also the forward transformation from G^S to G^T is essentially unique, i.e. unique up to switchings of independent transformation steps.

3 *Information and Complete Information Preservation:* In case of bidirectional model transformations, information preservation means that for each forward transformation from G_S to G_T there is also a backward transformation from G_T to G_S. Complete information preservation means in addition that each backward transformation starting with G_T leads to the same G_S.

It is the main aim of this paper to analyse under which conditions the properties defined above can be guaranteed and how these conditions can be checked with suitable tool support. Additional important properties, like semantical correctness, are not considered in this paper, but the interested reader is referred to (Bisztray et al.2009; Hermann et al.2010d). Semantical correctness of a forward transformation from G^S to G^T means that G^S and G^T are semantically equivalent in a suitable sense.

1.2. *Model Transformations Based on TGGs and Main Results*

Model transformations based on triple graph grammars (TGGs) have been introduced by Schürr (Schürr1994) and are used, among others, for the specification and execution of bidirectional model transformations between domain specific languages (DSLs). The power of bidirectional model transformations is based on the simultaneous support of transformations in both forward and backward direction. In addition to the general advantages of bidirectional model transformations, TGGs simplify the design of model transformations. A single set of triple rules is sufficient to generate the operational rules for the forward and backward model transformations. The key idea for the execution of model transformations via TGGs is to preserve the given source model and to add the missing target and correspondence elements in separate but connected components. For this reason, the transformation rules add new structures and do not necessarily need to delete existing elements. The resulting target model is obtained by type restriction. Indeed, non-deleting triple rules are sufficient for many case studies. However, in general it may be very difficult, if not impossible, to specify a model transformation whose validity depends on some global properties of the given models. An example may be automata minimization, where we transform a finite automaton into an automaton with the same behaviour, but with the smallest possible set of states. In this case, the transformation should translate any two states with the same behaviour into a single state. However, knowing if two states have the same behaviour is a global property of the given automaton. Nevertheless, a possible solution to simplify the model transformation is to perform some additional pre-processing of the source model or post-processing of the target model. For this reason and as it is common praxis for TGGs, we consider transformation rules that are non-deleting.

Based on (Ehrig et al.2009a), we show in our first main result in Thm. 1 that syntactical correctness and completeness are ensured for model transformations based on the formal control condition of source consistency. Moreover, we can ensure termination for the execution of a model transformation by the static condition that all TGG-rules are creating on the source component, which can be checked automatically.

In the general context of transformation systems, it is well-known that termination and local confluence implies confluence and hence functional behaviour, where local confluence can be checked by analysing all critical pairs between pairs of transformation rules. However, in the context of model transformations a weaker notion of confluence is sufficient, because uniqueness of results is required for successful executions of the model transformation only. Moreover, the control condition source consistency has to be included in the analysis as well. For this purpose, we generate a new kind of operational rules, called forward translation rules, which extend forward rules by additional attributes keeping track of the elements that have been translated already.

In our main results for the analysis of functional behaviour we extend the presented contributions in (Hermann et al.2010a; Hermann et al.2010c) and we additionally provide a less restrictive condition for functional behaviour. We show by Thm. 2 that functional behaviour of model transformations is ensured by strict confluence of all significant critical pairs of the forward translation rules. This means that several critical pairs can be neglected if they are not significant. Moreover, we analyse strong functional behaviour of model transformations, where uniqueness is also required for the transformation sequences up to switch-equivalence. By Thm. 3, we characterize strong functional behaviour by the absence of significant critical pairs. The results for functional behaviour are additionally used for improving efficiency of the execution, such that we can ensure polynomial time complexity if the provided sufficient conditions are satisfied (see Sec. 5.1).

Finally, we analyse information preservation of model transformations, i.e. the problem whether a source model can be reconstructed from the target model computed by the model transformation. Here, we extend our results presented in (Ehrig et al.2007) to TGGs with application conditions and to the notion of complete information preservation. We show by Thm. 4 that model transformations based on forward rules always ensure information preservation, which requires that there is a backward transformation sequence starting at the derived target model and resulting again in the given source model. Thereafter, we provide by Thm. 5 a sufficient condition for complete information preservation, i.e. that any reconstructed source model coincides with the given one.

1.3. *Mathematical Framework*

The mathematical background for this paper is the algebraic theory of graph transformations (Rozenberg1997), especially the double pushout approach for graphs introduced in (Ehrig et al.1973; Rozenberg1997; Ehrig et al.2006). This approach has been generalized from graphs to adhesive, adhesive HLR and M-adhesive categories (Lack and Sobociński2005; Ehrig et al.2006; Ehrig et al.2010). These are categorical frameworks where specific constructions like pushouts and pullbacks exist and are compatible with each other. This allows an instantiation of the categorical theory not only for graphs, but also for several other high-level structures, like typed and attributed graphs, hypergraphs and different kinds of Petri nets.

In our approach to model transformations, the abstract syntax of models is given by typed attributed graphs in the sense of (Ehrig et al.2006). In fact, main parts of the theory can be presented in adhesive or $\mathcal{M}$-adhesive categories (Lack and Sobociński2005; Ehrig et al.2010) as shown in (Hermann et al.2010c). But for simplicity, the construction of forward translation rules in this paper is based on attributed graphs, called graphs for short. However, the corresponding category of typed attributed graphs (see App. A) is an important example of an $\mathcal{M}$-adhesive category.

In this paper we assume basic knowledge of the algebraic theory of graph transformations as presented e.g. in Part I of (Ehrig et al.2006), but not of general category theory. For a summary of main results on an informal level and potential applications - going beyond our running example - we refer to Sec. 5.

1.4. *Structure of the Paper*

In Sec. 2, we present model transformations based on forward rules and forward translation rules as well as our first main result concerning correctness and completeness of model transformations. In Sec. 3, we provide our main results for analysing functional behaviour as well as information preservation, i.e. whether and how source models can be (completely) reconstructed from target models. Thereafter, Secs. 4 and 5 discuss related work and provide a conclusion. Finally, App. A recalls the technical details of the $\mathcal{M}$-adhesive category of typed attributed graphs. App. B provides the proofs of some auxiliary facts, while the proofs of the main results in Thms. 1-5 are given in the main part of the paper.

2. Model Transformation Based on Triple Graph Grammars

Triple graph grammars are a technique developed by Schürr ((Schürr1994)) that allows us to specify (bidirectional) model transformations. In particular, a triple graph grammar describes a class of triple graphs, consisting of pairs of models together with the relation between their elements. More precisely, a triple graph $G =(G_S \xleftarrow{s_G} G_C \xrightarrow{t_G} G_T)$ consists of a source graph G_S and a target graph G_T, which are related via a correspondence graph G_C and two graph morphisms $s_G : G_C \to G_S$ and $t_G : G_C \to G_T$ specifying how source elements correspond to target elements. In this context, the target graph of G may be considered the *forward* transformation of its source graph and the source graph may be considered the *backward* transformation of its target graph. Moreover, a given set of triple graphs can be seen as a class of model transformations, and the triple graph grammar that generates this set may be considered its specification.

$$\begin{tikzcd}
G = (G^S \arrow[d, "m"'] & G^C \arrow[l, "s_G"'] \arrow[r, "t_G"] \arrow[d, "m^C"'] & G^T) \arrow[d, "m^T"'] \\
R = (H^S & H^C \arrow[l, "s_H"] \arrow[r, "t_H"'] & H^T)
\end{tikzcd}$$
m^S

Triple graphs are related by means of triple graph morphisms which, as we would expect, are formed by three graph morphisms. More precisely, a triple graph morphism $m = (m_S, m_C, m_T) : G \to H$ consists of $m_S : G_S \to H_S$, $m_C : G_C \to H_C$ and $m_T : G_T \to H_T$ such that $m_S \circ s_G = s_H \circ m_C$ and $m_T \circ t_G = t_H \circ m_C$.

We can use any kind of graphs inside triple graphs, as long as they form an adhesive (or $\mathcal{M}$-adhesive) category (Lack and Sobociński2005; Ehrig et al.2006; Ehrig et al.2010). This means that we can have triple graphs (or, better, triple structures) consisting of many kinds of graphical structures. In this paper, we use attributed triple graphs based on E-graphs as presented in (Ehrig et al.2007). Moreover, our triple graphs are assumed to be typed over a given triple type graph TG. As usual, the typing is done by a triple graph morphism $type_G : G \to TG$.

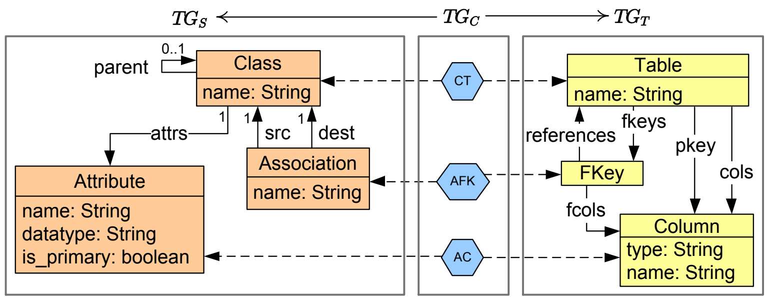

Fig. 1. Triple type graph for *CD2RDBM*

Example 2.1 (Triple Type Graph). Fig. 1 shows the type graph TG of the triple graph grammar TGG for our example model transformation *CD2RDBM* from class diagrams to database models. The source component TG_S defines the structure of class diagrams while in its target component the structure of relational database models is specified. Classes correspond to tables, attributes to columns, and associations to foreign keys. Throughout the example, originating from (Ehrig et al.2007), elements are arranged left, center, and right according to the component types source, correspondence and target. Attributes of structural nodes and edges are depicted within their containing structural nodes respectively edges. Formally, attribute values are edges to additional data value nodes (see App. A). Note that the correspondence component is important for the relation of the the source elements to their aligned target elements. For this reason, it is used in practical scenarios to navigate via the traceability links from source structures to target structures or vice versa. The morphisms between the three component are visualized by dashed arrows. The depicted multiplicity constraints are ensured by the triple rules of the grammar shown in Figs. 2-4. Moreover, the source language contains only those class diagrams in which the classes have unique primary attributes.

A rule tr in a triple graph grammar, called a triple rule, is an injective triple graph morphism $tr = (tr^S, tr^C, tr^T) : L \to R$ and without loss of generality we assume tr to be an inclusion. A triple rule is applied to a triple graph

$$\begin{tikzcd}
L = (L^S \arrow[d, "tr^S"'] & L^C \arrow[l, "s_L"'] \arrow[r, "t_L"] \arrow[d, "tr^C"'] & L^T) \arrow[d, "tr^T"'] & L \arrow[r, hook, "tr"] \arrow[d, "m"'] \arrow[dr, phantom, "(PO)"] & R \arrow[d, "n"] \\
R = (R^S & R^C \arrow[l, "s_R"] \arrow[r, "t_R"'] & R^T) & G \arrow[r, hook, "t"'] & H
\end{tikzcd}$$

G by matching L to some sub triple graph of G. Technically, a match is a morphism $m : L \to G$. The result of this application is the triple graph H, where L is replaced by R in G. Technically, the result of the transformation is defined by a pushout diagram with comatch $n : R \to H$ and transformation inclusion $t : G \hookrightarrow H$, as depicted on the right. This triple graph transformation (TGT) step is denoted by $G \xRightarrow{tr,m} H$. A grammar $TGG = (TG, S, TR)$ consists of a triple type graph TG, a triple start graph S and a set TR of triple rules.

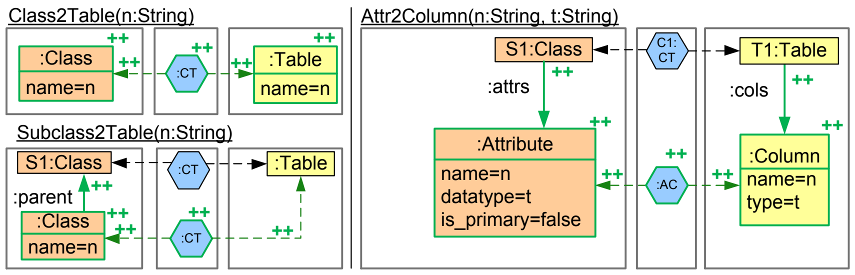

Fig. 2. Rules for the model transformation *CD2RDBM*, Part 1

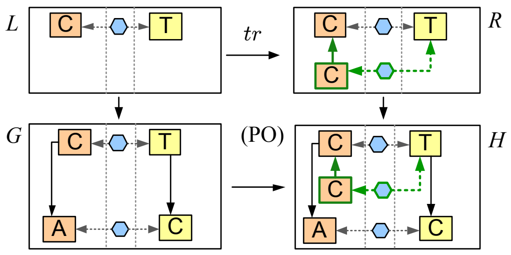

Fig. 3. Triple graph transformation step via rule "*Subclass2Table*"

Example 2.2 (Triple Rules and Triple Transformation Step). The triple rules in Fig. 2 are part of the rules of the grammar *TGG* for the model transformation *CD2RDBM*. They are presented in short notation, i.e. left and right hand side of a rule are depicted in one triple graph. Elements which are created by the rule are labelled with "++" and additionally marked by green line colouring. The rule "*Class2Table*" synchronously creates a class with name "n" together with the corresponding table in the relational database. Accordingly, subclasses are connected to the tables of its super classes by rule "*Subclass2Table*". Note that this rule creates the new class node together with an edge of type parent implying that our compact case study does not handle the case of multiple inheritance. Finally, rule "*Attr2Column*" creates attributes with type "t" together with their corresponding columns in the database component. Figure 3 shows a triple graph transformation step $G \xRightarrow{tr,m} H$ via rule $tr =$"*Subclass2Table*", where we ommitted the attribute values of the nodes and reduced the node types to the starting letters. The top line shows the rule with its left and right hand sides and the bottom line shows the given triple graph G and the resulting triple graph H. The effect of this step is the addition of a new subclass that is related to the existing table corresponding to the existing class.

From the application point of view a model transformation should be injective on the structural part, i.e. the transformation rules are applied along matches that do not identify structural elements. But it would be too restrictive to require injectivity of the matches also on the data and variable nodes, because we must allow that two different variables can be mapped to the same data value. For this reason we introduce the notion of almost injective matches, which requires that matches are injective except for the data value nodes. This way, attribute values can still be specified as terms within a rule and matched non-injectively to the same value. For the rest of this paper we generally require almost injective matching for the transformation sequences.

Definition 2.3 (Almost Injective Match). An attributed triple graph morphism $m : L \to G$ is called *almost injective match*, if it is non-injective at most for the set of variables and data values.

In graph transformation, negative application conditions (in short, NACs) allow us to restrict the application of transformation rules when certain structures are present in the given object graph (see, for instance, (Ehrig et al.2006)). In this paper, we consider NACs for triple rules, following (Ehrig et al.2009a). Moreover, for most case studies of model transformations source-target NACs, i.e. either source or target NACs, are sufficient and we regard them as the standard case. These NACs prohibit the existence of certain structures either in the source or in the target part only, while general NACs may prohibit both at once, or even structures in the correspondence graph. For model transformations with more general application conditions we refer to (Golas et al.2011).

Definition 2.4 (Triple Rule with Negative Application Conditions). Given a triple rule $tr = (L \to R)$, a negative application condition (NAC) $(n : L \to N)$ consists of a triple graph N and a triple graph morphism n. A NAC with $n = (n^S, id_{L^C}, id_{L^T})$ is called *source NAC* and a NAC with $n = (id_{L^S}, id_{L^C}, n^T)$ is called *target NAC*.

A match $m : L \to G$ is NAC consistent if there is no injective $q : N \to G$ such that $q \circ n = m$ for each NAC $L \xrightarrow{n} N$. A triple transformation $G \overset{*}{\Rightarrow} H$ is NAC consistent if all matches are NAC consistent.

For the rest of this paper we only consider source and target NACs and almost injective matches, which is sufficient in many practical case studies.

Given a triple type graph TG, a set of triple rules TR and a start graph $\varnothing = (\varnothing \leftarrow \varnothing \to \varnothing)$ (usually, the empty triple graph), we denote by VL the set of integrated models (i.e. triple models including elements in the source, target and correspondence component) that are generated from $\varnothing$ using the rules in TR. Then, the source language VL_S and target language VL_T of VL are derived by projections to the triple components, i.e. $VL_S = proj_S(VL)$ and $VL_T = proj_T(VL)$. Moreover, we denote the set of all models typed over the source component TG^S of the triple type graph TG by $VL(TG^S)$ implying directly that $VL_S \subseteq VL(TG^S)$. Analogously, by $VL(TG^T)$ we denote the set of all target models typed over TG^T and have that $VL_T \subseteq VL(TG^T)$.

Example 2.5 (Triple Rules with NACs). The remaining triple rules of the model transformation "*CD2RDBM*" are shown in Fig. 4. Rule "*PrimaryAttr2Column*" extends "*Attr2Column*" from Ex. 2.2 by creating additionally a link of type "*pkey*" for the column and by setting the attribute value "is_primary=true". This rule contains NACs, which are specified in short notation. The NAC-only elements are specified by red line colouring and additionally, with a surrounding frame with label "NAC". A complete NAC is obtained by composing the left hand side of a rule with the marked NAC-only elements. The

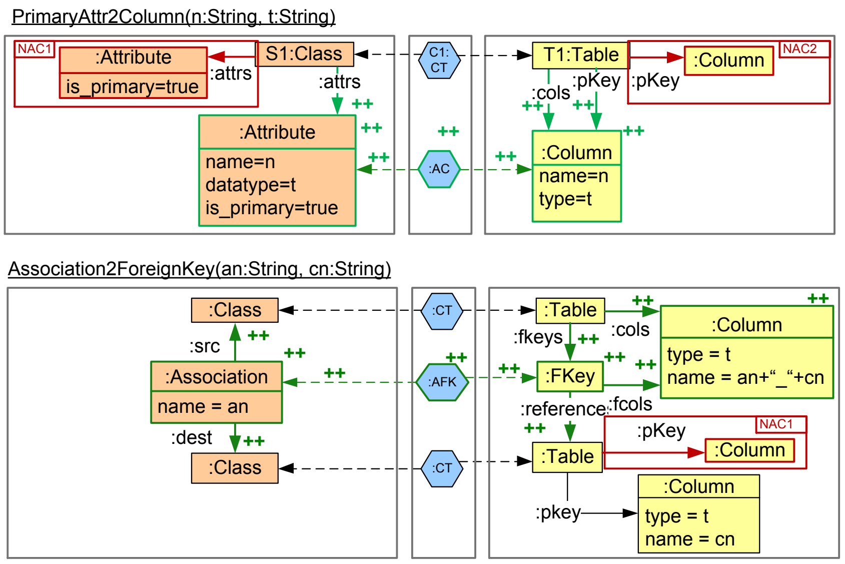

Fig. 4. Rules for the model transformation *CD2RDBM*, Part 2

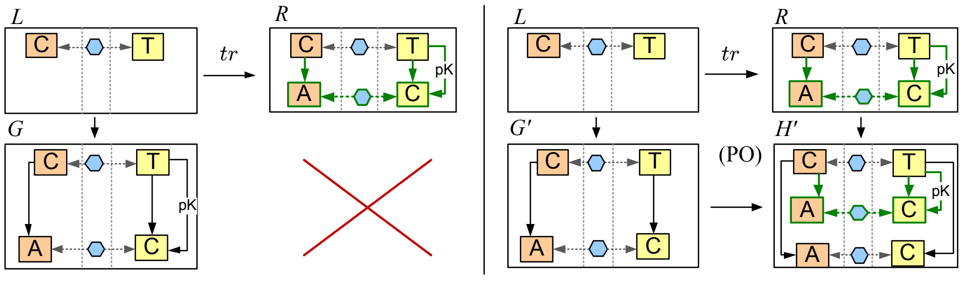

Fig. 5. Vioaltion of NAC and satisfaction of NAC for rule "*PrimaryAttr2Column*"

source and a target NACs ensure that there is neither a primary attribute in the class diagram nor a primary key in the data base model present when applying the rule. More formally, the depicted NACs are actually NAC schemata (see Rem. 2.6 below). The rule "*Association2ForeignKey*" creates an association between two classes and the corresponding foreign key, where the parameters "an" and "cn" are used to set the names of the association and column nodes. The target NAC ensures that the used primary key for the foreign key in the data base component is unique. The left component of Fig. 5 shows a violation of the target NAC for rule "*PrimaryAttr2Column*", whose target NAC forbids the presence of an existing primary key at the matched table. In its right component, the figure shows a NAC consistent transformation step, where no primary key and also no primary is present and also the existing attribute is assumed to be not a primary one. Analogously to Fig. 3 we use a compact notation for the transformation steps.

Remark 2.6 (NACs for almost injective matches). In order to simplify the specification of NACs for systems with almost injective matches we interpret all specified NACs in a TGG as NAC schemata according to (Hermann et al.2011a). A match $m : L \to G$ satisfies a NAC schema $n : L \to N$, effectively if there is no almost injective morphism $q : N \to G$, such that $q \circ n = m$. The difference to standard NACs is that the morphism q is allowed to identify data values. According to Fact 2.14 in (Hermann et al.2011a), a NAC schema is equivalent to the set of all instantiated NACs, which are given by a structural copy of the NAC but with an adapted data part for each possible data evaluation. Due to this equivalence, we can provide the formal results in this paper using the standard notion of NAC satisfaction with injective morphism $q : N \to G$ according to Def. 2.4.

Triple graph grammars specify model transformations, but they do not directly solve the problem of, given a source model (or a target model) how to build its forward transformation (respectively, its backward transformation). However, as we will see in the next two subsections, from a triple graph grammar we can derive its associated operational rules that are used for this task. In particular, in Subsec. 2.1, we present model transformation in terms of forward (and backward) transformation rules, describing the main results (Schürr and Klar2008; Ehrig et al.2009a). Then, in Subsec. 2.2, we present a more elaborate kind of rules, called forward (and backward) translation rules, based on the notion of translation attributes. These rules are the basis for the analysis of functional behaviour and information preservation in Sec. 3.

2.1. *Model Transformation Based on Forward Rules*

As said above, in order to describe how given source models can be transformed into corresponding target models, we use the so-called operational rules, which are derived from the triple rules TR as shown below. From each triple rule tr we derive a source rule tr_S and a forward rule tr_F for forward transformation sequences for the parsing and, respectively, the construction of a model of the source language. As we can see, source rules essentially consist of the source part of triple rules. As a consequence they may be used to generate or parse the valid source graphs. However, we must notice that the set of graphs that can be generated by the source rules includes, but in general does not coincide with VL_S, the source part of the triple graphs that are generated by the triple rules. That is, there may be models generated by the source rules that do not have a valid transformation, according to the triple rules. The reason is that, at a certain moment, it may be impossible to apply a given triple rule, because we can not match the target or the correspondence part of its left-hand side, but it may be possible to match just the source part of the rule (i. e., its associated source rule).

$$\begin{tikzcd}
(L^S \arrow[d, "tr^S"'] & \varnothing \arrow[l] \arrow[r] \arrow[d] & \varnothing) \arrow[d] \\
(R^S & \varnothing \arrow[l] \arrow[r] & \varnothing)
\end{tikzcd}$$

source rule tr_S

$$\begin{tikzcd}
(\varnothing \arrow[d] & \varnothing \arrow[l] \arrow[r] \arrow[d] & L^T) \arrow[d, "tr^T"'] \\
(\varnothing & \varnothing \arrow[l] \arrow[r] & R^T)
\end{tikzcd}$$

target rule tr_T

$$\begin{tikzcd}
(R^S \arrow[d, "id"'] & L^C \arrow[l, "tr^S \circ s_L"'] \arrow[r, "t_L"] \arrow[d, "tr^C"'] & L^T) \arrow[d, "tr^T"] \\
(R^S & R^C \arrow[l, "s_R"'] \arrow[r, "t_R"] & R^T)
\end{tikzcd}$$

forward rule tr_F

The intuition behind forward rules is quite simple. Given a certain source model, G_S, we are trying to find a target model G_T such that there is a triple graph $(G_S \xleftarrow{s_G} G_C \xrightarrow{t_G} G_T)$ that can be generated by the given set of triple rules. This means that G_T can be generated by the target part of the triple rules. However, the problem is to know which target rules should be used. Instead, we use forward rules that restrict the choice of the possible rules to use in this construction. In particular, given a triple rule tr, in its associated forward rule tr_F, the source part of its left-hand side, R_S, coincides with the source part of the right-hand side of tr. This means that, if there is a match of tr_F in G_S then its source part could have been generated by tr or, conversely, if there is no match of the source part of tr_F in G_S then we would be unable to use tr to generate the triple graph $(G_S \xleftarrow{s_G} G_C \xrightarrow{t_G} G_T)$. In addition, using forward rules, we not only are able to build G_T, but also the correspondence between G_S and G_T.

If the given triple rules include NACs, then these NACs are inherited by the operational rules as follows. Each forward rule tr_F inherits the target NACs of its associated triple rule tr, since target NACs restrict the construction of target models. Conversely, source NACs restrict the construction of source models. For this reason they are inherited by source rules. By TR_S and TR_F we denote the sets of all source and forward rules derived from TR. Analogously, we derive a target rule tr_T and a backward rule tr_B for the construction and transformation of a model of the target language leading to the sets TR_T and TR_B.

As introduced in (Ehrig et al.2007; Ehrig et al.2009a) the derived operational rules provide the basis to define model transformations based on forward transformation sequences that are executed via the formal control condition source consistency, which we briefly explain in the following. We know that G^T is the transformation of G^S if the triple graph $G = (G^S \leftarrow G^C \rightarrow G^T)$ is in the class defined by the TGG, i.e. if there is a sequence of transformations $\varnothing \xRightarrow{tr_1} G_1 \Rightarrow \ldots \xRightarrow{tr_n} G_n = G$. But, as we can see in Fact 2.9, this sequence of transformations can be decomposed into a sequence of transformations using the associated source rules, followed by a sequence of transformations using the associated forward rules $\varnothing \xRightarrow{tr_{1S}} G_{10} \Rightarrow \ldots \xRightarrow{tr_{nS}} G_{n0} = (G^S \leftarrow \varnothing \rightarrow \varnothing) \xRightarrow{tr_{1F}} G_{n1} \Rightarrow \ldots \xRightarrow{tr_{nF}} G_{nn} = G$, where the source and forward sequences are *match consistent*, meaning that the matches of the corresponding source and forward steps are compatible. Technically, source and forward match are compatible if they coincide for each mapped element on their source component, i.e., $m_S^S(x) = m_F^S(x)$ assuming that the trace morphisms of the transformation sequences are inclusions. Moreover, Fact 2.9 also tells us that for every match consistent sequence of transformations $\varnothing \xRightarrow{tr_{1S}} G_{10} \Rightarrow \ldots \xRightarrow{tr_{nS}} G_{n0} \xRightarrow{tr_{1F}} G_{n1} \Rightarrow \ldots \xRightarrow{tr_{nF}} G_{nn} = G$ there is a corresponding sequence of triple rule transformations $\varnothing \xRightarrow{tr_1} G_1 \Rightarrow \ldots \xRightarrow{tr_n} G_n = G$. This means that, if we want to compute the transformation of a certain source model G^S, what we can do is to find a sequence of forward transformations $(G^S \leftarrow \varnothing \rightarrow \varnothing) \xRightarrow{tr_{1F}} G_{n1} \Rightarrow \ldots \xRightarrow{tr_{nF}} G_{nn} = G$, such that the corresponding sequence of match consistent source transformations generates G^S, i.e. $\varnothing \xRightarrow{tr_{1S}} G_{10} \Rightarrow \ldots \xRightarrow{tr_{nS}} (G^S \leftarrow \varnothing \rightarrow \varnothing)$. These forward sequences are called *source consistent.* In principle, to find a source consistent forward sequence we must first parse the source model, i.e. we must find the match consistent source sequence that generates G^S. However, in (Ehrig et al.2009a) it was shown that source and forward sequences can be constructed simultaneously.

Let us now see some of these concepts in more detail.

Definition 2.7 (Model Transformation based on Forward Rules). A *model transformation sequence* $(G^S, G_0 \xRightarrow{tr_F^*} G_n, G^T)$ consists of a source graph G^S, a target graph G^T, and a source consistent forward TGT-sequence $G_0 \xRightarrow{tr_F^*} G_n$ with $G^S = G_0^S$ and $G^T = G_n^T$.
A *model transformation* $MT : VL(TG^S) \Rrightarrow VL(TG^T)$ is defined by all model transformation sequences $(G^S, G_0 \xRightarrow{tr_F^*} G_n, G^T)$ with $G^S \in VL(TG^S)$ and $G^T \in VL(TG^T)$. All the corresponding pairs (G^S, G^T) define the *model transformation relation* $MTR_F \subseteq VL(TG^S) \times VL(TG^T)$ based on TR_F.

In (Ehrig et al.2007; Ehrig et al.2009a) we have proved that source consistency ensures (syntactical) correctness and completeness of model transformations based on forward rules with respect to the language *VL* of integrated models. Syntactical correctness means that every model transformation sequence $(G^S, G_0 \xRightarrow{tr_F^*} G_n, G^T)$ leads to an integrated model $G_n = (G^S \leftarrow G^C \rightarrow G^T) \in VL$. In other words, that source consistent forward transformations generate correct model transformations, according to the class of transformations specified by the given TGG. Completeness means that for any integrated model $G = (G^S \leftarrow G^C \rightarrow G^T) \in VL$, there is a corresponding model transformation sequence $(G^S, G_0 \xRightarrow{tr_F^*} G, G^T)$. Intuitively, that any valid transformation specified by a TGG can be implemented by a source consistence forward transformation.

Note that the model transformation relation MTR_F is in general not a function from $VL(TG^S)$ to $VL(TG^T)$, but we study functional behaviour in Sec. 3.

Definition 2.8 (Syntactical Correctness and Completeness). A model transformation $MT : VL(TG^S) \Rrightarrow VL(TG^T)$ based on forward rules is

— *syntactically correct*, if for each model transformation sequence $(G^S, G_0 \xRightarrow{tr_F^*} G_n, G^T)$ there is $G \in VL$ with $G = (G^S \leftarrow G^C \rightarrow G^T)$ implying further that $G^S \in VL_S$ and $G^T \in VL_T$, and it is
— *complete*, if for each $G^S \in VL_S$ there is $G = (G^S \leftarrow G^C \rightarrow G^T) \in VL$ with a model transformation sequence $(G^S, G_0 \xRightarrow{tr_F^*} G_n, G^T)$ and $G_n = G$. Vice versa, for each $G^T \in VL_T$ there is $G = (G^S \leftarrow G^C \rightarrow G^T) \in VL$ with a model transformation sequence $(G^S, G_0 \xRightarrow{tr_F^*} G_n, G^T)$ and $G_n = G$.

For showing syntactical correctness and completeness for model transformations based on TGGs by Thm. 1 we use the following composition and decomposition result for TGT-sequences, which is shown in (Ehrig et al.2007) and (Ehrig et al.2009b) for the case of rules without and with NACs, respectively.

Fact 2.9 (Composition and Decomposition of TGT-Sequences).

1 **Decomposition**: For each TGT-sequence $G_0 \xRightarrow{tr_1} G_1 \Rightarrow \ldots \xRightarrow{tr_n} G_n$ (1)
based on triple rules there is a corresponding match consistent TGT-sequence

$$G_0 = G_{00} \xRightarrow{tr_{1S}} G_{10} \Rightarrow \ldots \xRightarrow{tr_{nS}} G_{n0} \xRightarrow{tr_{1F}} G_{n1} \Rightarrow \ldots \xRightarrow{tr_{nF}} G_{nn} = G_n \qquad (2)$$

based on corresponding source and forward rules.

2 **Composition:** For each match consistent transformation sequence (2) there is a corresponding transformation sequence (1).

3 **Bijective Correspondence:** Composition and decomposition are inverse to each other (up to isomorphism).

Theorem 1 (Syntactical Correctness and Completeness). Each model transformation $MT : VL(TG^S) \Rrightarrow VL(TG^T)$ based on forward rules is syntactically correct and complete.

Proof.

1 (Syntactical Correctness)

Given a model transformation sequence $(G^S, G_0 \xRightarrow{tr_F^*} G_n, G^T)$, then source consistency of $G_0 \xRightarrow{tr_F^*} G_n$ implies a match consistent sequence $\varnothing \xRightarrow{tr_S^*} G_0 \xRightarrow{tr_F^*} G_n$. Using the composition part of Fact 2.9 we have a corresponding TGT-sequence $\varnothing \xRightarrow{tr^*} G_n$. This implies for $G = G_n$ that $G \in VL$ with $G = (G^S \leftarrow G^C \rightarrow G^T)$ and hence, also $G^S \in VL_S$ and $G^T \in VL_T$.

2 (Completeness)

Given $G^S \in VL_S$ we have by definition of VL_S some $G = (G^S \leftarrow G^C \rightarrow G^T) \in VL$. This means we have a TGT-sequence $\varnothing \xRightarrow{tr^*} G$ and by the decomposition part of Fact. 2.9 we have a match consistent sequence $\varnothing \xRightarrow{tr_S^*} G_0 \xRightarrow{tr_F^*} G$, which defines a model transformation sequence $(G^S, G_0 \xRightarrow{tr_F^*} G, G^T)$ using $G = (G^S \leftarrow G^C \rightarrow G^T)$. Vice versa, we use Rem. 2.10. □

Remark 2.10 (Composition and Decomposition for Backward Case). For each TGT-sequence $G_0 \xRightarrow{tr^*} G_n$ there is also a corresponding match consistent backward TGT-sequence $G_0 = G_{00} \xRightarrow{tr_{1,T}} G_{01} \Rightarrow \ldots \xRightarrow{tr_{n,T}} G_{0n} \xRightarrow{tr_{1,F}} G_{1n} \Rightarrow \ldots \xRightarrow{tr_{n,F}} G_{nn} = G_n$ based on target and backward rules leading to a backward model transformation $MT_B : VL(TG^T) \Rrightarrow VL(TG^S)$ with similar results as in the forward case.

Termination of model transformations is considered in Fact 3.11 in Sec. 3.1.

2.2. *Model Transformation Based on Forward Translation Rules*

A main difficulty in implementing the techniques described in the previous subsection is related to how to check source consistency in a reasonably efficient way. In this section we show an approach, introduced in (Hermann et al.2010c), that solves this problem in a relatively simple way. Moreover, this approach sets the basis for the analysis of model transformations in Sec. 3.1.

The basic idea is to use what we call *translation attributes* that tell us, as described in Ex. 2.13, which elements of the given source model have been already translated or used to build the target and the correspondence models. More precisely, given a source model G^S, if we think of building in parallel the match consistent sequences of source and forward transformations, $\varnothing \xRightarrow{tr_{1S}} G_{10} \Rightarrow \ldots \xRightarrow{tr_{nS}} G_{n0} = (G^S \leftarrow \varnothing \rightarrow \varnothing)$ and $(G^S \leftarrow \varnothing \rightarrow \varnothing) \xRightarrow{tr_{1F}} G_{n1} \Rightarrow \ldots \xRightarrow{tr_{nF}} G_{nn} = G$, at any point i, when we would be going to apply the source transformation $G_{(i-1)0} \xRightarrow{tr_{iS}} G_{i0}$ and the forward transformation $G_{n(i-1)} \xRightarrow{tr_{iF}} G_{ni}$, all the elements in the source graph which are included in $G_{(i-1)0}$ would have their translation attributes set to true and the rest of them would be set to false, since the elements in $G_{(i-1)0}$ are the elements that have been translated by the forward transformations $(G^S \leftarrow \varnothing \rightarrow \varnothing) \xRightarrow{tr_{1F}} G_{n1} \Rightarrow \ldots \xRightarrow{tr_{(i-1)F}} G_{n(i-1)}$.

This means that, first, we must enrich the given source graph with translation attributes assigning one translation attribute to each element (i.e. each node, edge and attribute) of the source graph. Second, before starting the transformation process, we must set all translation attributes to false, since initially no element has already been used in building the target model. Third, when applying the forward rule tr_{iF} we would need to check, on the one hand, that the associated source rule, tr_{iS}, could be applied using a consistent match. This is equivalent to checking if all the elements of the source graph that are matched by the left-hand side of the source rule have their translation attributes set to true and all the elements that would be added by the source rule have their translation attributes set to false. Moreover, if that source rule includes some NAC, we would need to check that the subgraph of the source graph, consisting of the elements with true translation attributes, satisfies that NAC with respect to the given match. On the other hand, we have to set to true all the translation attributes of the elements of the source graph that would have been added by tr_{iS}. Finally, to check that the source model has been completely transformed into a target (and a correspondence) model, we would need to check that, at the end of the transformation, all the translation attributes of the source model are set to true. In this case, we say that the transformation sequence is *complete.*

The above explanation of how we use translation attributes may give the impression that the management of translation attributes (i.e. checking if we can apply a transformation rule and updating the attributes after each transformation step) is external to the transformation process, in the sense that the model transformation process is still done using forward rules, but checking and updating the translation attributes is done in some metaprocess. Actually, this is not true. A second key idea of our approach is that we can integrate the management of translation attributes into the transformation process. We do this by using a variant of forward rules that we call forward translation rules. More precisely, given a triple rule $tr = (L \rightarrow R)$, its associated forward translation rule, tr_{FT}, as described in Example 2.16, enriches its associated forward rule in the following aspects:

— In the source part of the left-hand side of the rule, every element in L_S has an associated translation attribute set to true, and every element in $L_R \setminus L_S$ has an associated translation attribute set to false. In this way, the matching of the rule takes care that, in the given source graph, all elements that are expected to have been already created by previous source transformations have a true translation attribute and all the elements that are supposed to be translated in this transformation step have a false translation attribute.

— In the source part of the right-hand side of the rule, every element in R_S has an associated translation attribute set to true. In this way, the transformation defined by the rule takes care of updating translation attributes of the elements that are supposed to be translated in this transformation step.[†]

— Every NAC $n : L \to N$ of tr (not only target NACs) is included in tr_{FT}, but all the elements in the source part of the NAC (either in L or in N) are included with an associated translation attribute set to true.

The main result in this section shows that model transformations based on source consistent forward TGT-sequences are equivalent to those based on complete forward translation TGT-sequences as stated by Fact 2.20. The control condition source consistency is ensured by the completeness of forward translation TGT-sequences, which are based on the generated forward translation rules. For this reason, the check of source consistency for forward TGT-sequences is reduced to a check whether the model is completely translated, i.e. all translation attributes are set to true.

Next, we provide the technical details of this approach, together with some examples. Even if the basic ideas, as we have seen, are relatively simple, some basic definitions are a bit involved because of the details when handling the translation attributes. For this reason, in a superficial reading of this part , the reader may prefer to skip these definitions.

In our notation, the translation attribute of each node, edge and attribute of a graph is labelled with the prefix "`tr`". Notice that we use different font shapes for a triple rule tr (italic) and for the prefix of translation attributes "`tr`" (typewriter) in order to emphasize the difference. Given an attributed graph $AG = (G, D)$ and a family of subsets $M \subseteq |G|$ for the domains $|G|$ of G, we call AG' a graph with translation attributes over AG if it extends AG with one new Boolean-valued attribute $\texttt{tr\_}x$ for each element x (node or edge) in M and one new Boolean-valued attribute $\texttt{tr\_}x\texttt{\_}a$ for each attribute a associated to such an element x in M. The family M together with all these additional translation attributes is denoted by Att_M. Note that we use the attribution concept of E-Graphs as presented in (Ehrig et al.2006), where attributes are possible for nodes and edges. Attributed graphs consist of a graph for the structural part, an algebra for the data values and the attributes are edges between the structural elements (nodes and edges) and the data values. Roughly spoken, an attribution edge of a node points to the assigned value for the specific attribute. For more details on attributed graphs see App. A.

Definition 2.11 (Family with Translation Attributes). Given an attributed graph $AG = (G, D)$ we denote by $|G| = (V_G^G, V_G^D, E_G^G, E_G^{NA}, E_G^{EA})$ the underlying family of sets

[†] We may note that forward translation rules are not just inclusions (i.e. non-deleting rules), since some translation attributes are modified by the rule. As a consequence, as we may see in Definition 2.14, forward translation rules are spans of inclusions as it happens in the general case of graph transformation rules.

containing all nodes and edges. Let $M \subseteq |G|$ with $(V_M^G, V_M^D, E_M^G, E_M^{NA}, E_M^{EA})$, then a *family with translation attributes* for (G, M) extends M by additional translation attributes and is given by $Att_M = (V_M^G, V_M^D, E_M^G, E^{NA}, E^{EA})$ with:

— $E^{NA} = E_M^{NA} \cup \{\texttt{tr\_}x \mid x \in V_M^G\} \cup \{\texttt{tr\_}x\texttt{\_}a \mid a \in E_M^{NA}, src_G^{NA}(a) = x \in V_G^G\}$,
— $E^{EA} = E_M^{EA} \cup \{\texttt{tr\_}x \mid x \in E_M^G\} \cup \{\texttt{tr\_}x\texttt{\_}a \mid a \in E_M^{EA}, src_G^{EA}(a) = x \in E_G^G\}$.

Definition 2.12 (Graph with Translation Attributes). Given an attributed graph $AG = (G, D)$ and a family of subsets $M \subseteq |G|$ with $\{\mathbf{T}, \mathbf{F}\} \subseteq V_M^D$ and let Att_M be a family with translation attributes for (G, M) according to Def. 2.11. Then, $AG' = (G', D)$ is a *graph with translation attributes* over AG, where the domains $|G'|$ of G' are given by the gluing via pushout of $|G|$ and Att_M over M and the source and target functions of G' are defined as follows:

$$\begin{tikzcd} M \arrow[r, hook] \arrow[d, hook] \arrow[dr, phantom, "(PO)"] & Att_M \arrow[d] \\ {|G|} \arrow[r] & {|G'|} \end{tikzcd}$$

— $src_{G'}^G = src_G^G$, $trg_{G'}^G = trg_G^G$,

— $src_{G'}^X(z) = \begin{cases} src_G^X(z) & z \in E_G^X \\ x & z = \texttt{tr\_}x \text{ or } z = \texttt{tr\_}x\texttt{\_}a \end{cases}$ for $X \in \{NA, EA\}$,

— $trg_{G'}^X(z) = \begin{cases} trg_G^X(z) & z \in E_G^X \\ \mathbf{T} \text{ or } \mathbf{F} & z = \texttt{tr\_}x \text{ or } z = \texttt{tr\_}x\texttt{\_}a \end{cases}$ for $X \in \{NA, EA\}$.

Att_M^v, where $v = \mathbf{T}$ or $v = \mathbf{F}$, denotes a family with translation attributes where all attributes are set to v. Moreover, we denote by $AG \oplus Att_M$ that AG is extended by the translation attributes in Att_M, i.e. $AG \oplus Att_M = (G', D)$ for $AG' = (G', D)$ as defined above. Analogously, we use the notion $AG \oplus Att_M^v$ for translation attributes with value v and we use the short notation $Att^v(AG) := AG \oplus Att_{|G|}^v$.

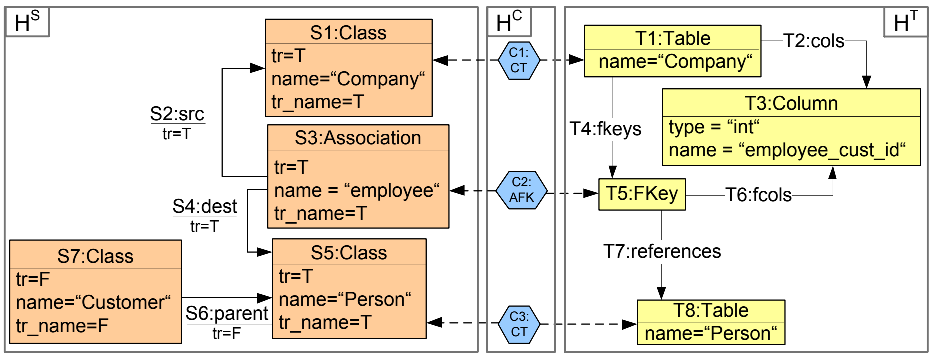

Fig. 6. Triple graph with translation attributes

Example 2.13 (Triple Graph with Translation Attributes). Fig. 6 shows the triple graph $H = (H^S \leftarrow H^C \rightarrow H^T)$ which is extended by some translation attributes in the source component. The translation attributes with value "**T**" indicate that the owning elements have been translated during a model transformation sequence using forward translation rules, which are defined in Def. 2.14 hereafter. The remaining elements (edge S6, node S7 and the attribute "name" of S7) in the source component are still marked with translation attributes set to “**F**”. These elements can still be matched and will become translated at later steps. The translation attributes are used to explicitly specify the elements which have been translated up to a specific step during the execution of a model transformation.

The concept of forward translation rules, which we introduced in (Hermann et al.2010c), extends the construction of forward rules by additional translation attributes in the source component. As described in Ex. 2.13, the translation attributes are used to keep track of the elements that have been translated so far. Since triple rules may create new attributes for existing nodes by definition, we also have to keep track of the translation of the attributes. The separate handling of nodes and their attributes is used, e.g., in synchronization scenarios (Hermann et al.2011b). At the beginning, the source model of a model transformation sequence is extended by translation attributes that are all set to “**F**” and, step by step, they are set to “**T**” when their containing elements are translated by a forward translation rule.

Definition 2.14 (Forward Translation Rule). Given a triple rule $tr = (L \to R)$, the *forward translation rule* of tr is given by $tr_{FT} = (L_{FT} \xleftarrow{l_{FT}} K_{FT} \xrightarrow{r_{FT}} R_{FT})$ defined as follows using the forward rule $(L_F \xrightarrow{tr_F} R_F)$ and the source rule $(L_S \xrightarrow{tr_S} R_S)$ of tr, where we assume w.l.o.g. that tr is an inclusion:
— $L_{FT} = L_F \oplus Att^{\mathbf{T}}_{L_S} \oplus Att^{\mathbf{F}}_{R_S \setminus L_S}$
— $K_{FT} = L_F \oplus Att^{\mathbf{T}}_{L_S}$
— $R_{FT} = R_F \oplus Att^{\mathbf{T}}_{L_S} \oplus Att^{\mathbf{T}}_{R_S \setminus L_S} = R_F \oplus Att^{\mathbf{T}}_{R_S}$,
— l_{FT} and r_{FT} are the induced inclusions.

Moreover, for each NAC $n : L \to N$ of tr we define a forward translation NAC $n_{FT} : L_{FT} \to N_{FT}$ of tr_{FT} as inclusion with $N_{FT} = (L_{FT} +_L N) \oplus Att^{\mathbf{T}}_{N_S \setminus L_S}$.

Remark 2.15. Note that $(L_{FT} +_L N)$ is the union of L_{FT} and N with shared L (formally a pushout) and for a target NAC n the forward translation NAC n_{FT} does not contain any additional translation attributes because $N_S = L_S$. Given a set of triple rules TR we denote by TR_{FT} the set of all tr_{FT} with $tr \in TR$.

Example 2.16 (Derived Forward Translation Rules). The rule “*Subclass2Table*$_{FT}$” in Fig. 7 is the derived forward translation rule of the triple rule “*Subclass2Table*” in Fig. 2. Note that we abbreviate “`tr_`x” for an item (node or edge) x by “`tr`” and “`tr_`x`_`a” by “`tr_`$type(a)$” in the figures to increase readability. The compact notation of forward translation rules specifies the modification of translation attributes by “[**F** ⇒ **T**]”, meaning that the attribute is matched with the value “**F**” and set to “**T**” during the transformation step. The detailed complete notation of a forward translation rule is shown on the right of Fig. 7 for “*Subclass2Table*$_{FT}$”.

Fig. 8 shows the forward translation rule with NACs “*PrimaryAttr2Column*$_{FT}$” derived from the triple “*PrimaryAttr2Column*” in Fig. 4. According to Def. 2.14 the source elements of the triple rule are extended by translation attributes and changed by the rule from “**F**” to “**T**”, if the owning elements are created by the triple rule. Furthermore, the forward translation rule contains both, the source and the target NACs of the triple rule,

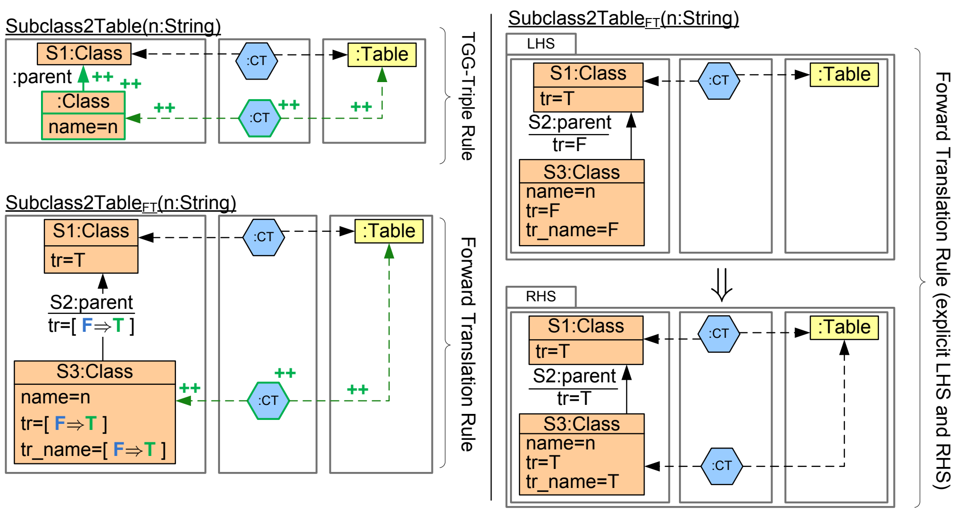

Fig. 7. Forward translation rule $Subclass2Table_{FT}(n : String)$

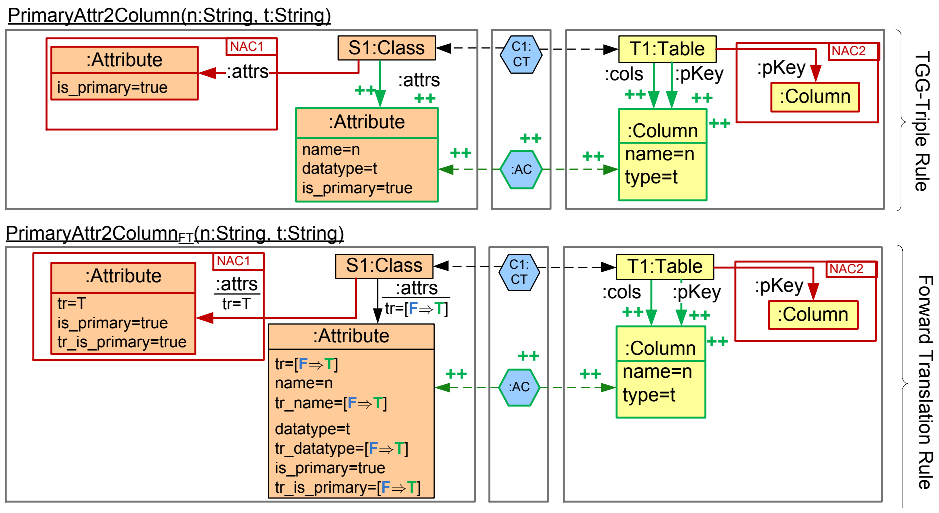

Fig. 8. Forward translation rule with NACs

where the NAC-only elements in the source NACs are extended by translation attributes set to "**T**". Thus, a source NAC concerns only elements that have been translated so far.

Since forward translation rules are deleting only on attribution edges, each NAC-consistent match is applicable according to Fact 1 in (Hermann et al.2010a). Note that in the general case of deleting rules the additional gluing condition has to be checked (Ehrig et al.2006), in order to ensure, e.g., that edges do not become dangling due to the deletion of nodes.

Now, we define model transformations based on forward translation rules in the same way as for forward rules in Def. 2.7, where source consistency of the forward sequence is replaced by completeness of the forward translation sequence.

Definition 2.17 (Complete Forward Translation Sequence). A forward translation sequence $G_0 \xRightarrow{tr^*_{FT}} G_n$ with almost injective matches is called *complete* if no further forward translation rule is applicable and G_n is *completely translated*, i.e. all translation attributes of G_n are set to true ("**T**").

Definition 2.18 (Model Transformation Based on Forward Translation Rules). A *model transformation sequence* $(G^S,\, G'_0 \xRightarrow{tr^*_{FT}} G'_n, G^T)$ based on forward translation rules TR_{FT} consists of a source graph G^S, a target graph G^T, and a complete TGT-sequence $G'_0 \xRightarrow{tr^*_{FT}} G'_n$ typed over $TG' = TG \oplus Att^{\mathbf{F}}_{|TG^S|} \oplus Att^{\mathbf{T}}_{|TG^S|}$ based on TR_{FT} with $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$ and $G'_n = (Att^{\mathbf{T}}(G^S) \leftarrow G^C \rightarrow G^T)$.
A *model transformation* $MT : VL(TG^S) \Rrightarrow VL(TG^T)$ based on TR_{FT} is defined by all model transformation sequences as above with $G^S \in VL(TG^S)$ and $G^T \in VL(TG^T)$. All the corresponding pairs (G^S, G^T) define the *model transformation relation* $MTR_{FT} \subseteq VL(TG^S) \times VL(TG^T)$ based on TR_{FT}. The model transformation is *terminating* if there are no infinite TGT-sequences via TR_{FT} starting with $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$ for some source graph G^S.

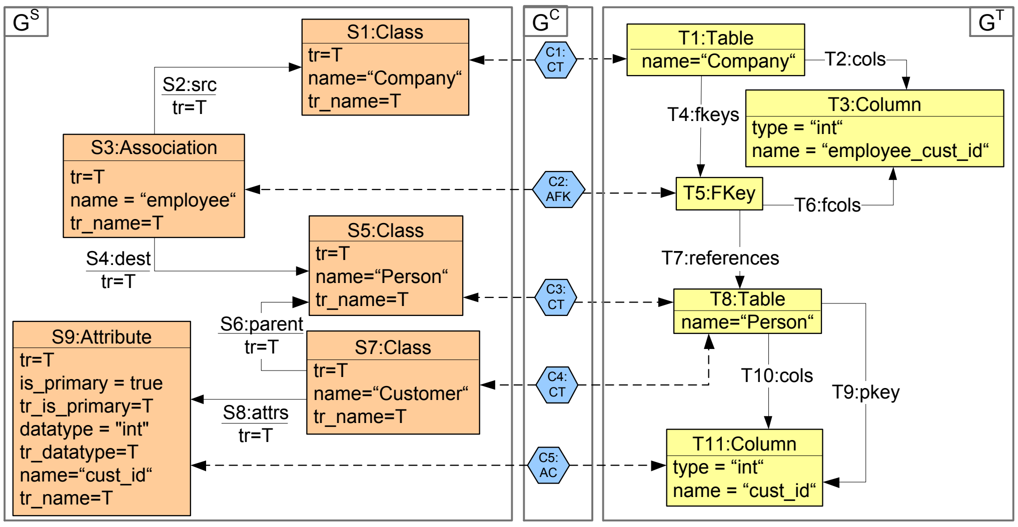

Fig. 9. Triple graph instance with translation attributes for *CD2RDBM*

Example 2.19 (Model Transformation via Forward Translation Rules). Fig. 9 shows the resulting triple graph of a forward translation sequence. The execution starts by extending the source model G^S with translation attributes according to Def. 2.18, i.e. $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$. We can execute the forward translation sequence via the following sequence of forward translation steps. $G'_0 \xrightarrow{Class2Table_{FT}} G'_1 \xrightarrow{Class2Table_{FT}} G'_2 \xrightarrow{Subclass2Table_{FT}} G'_3 \xrightarrow{PrimaryAttr2Col_{FT}} G'_4 \xrightarrow{Association2FKey_{FT}} G'_5$, with G'_5 being the graph G in Fig. 9. Now, the triple graph G'_5 is completely translated, because all translation attributes are set to "**T**". No further forward translation rule is applicable and we derive the resulting target model G^T by restricting G'^5 to its target component, i.e. $G^T = G'^T_5$. According to the equivalence of the model transformation concepts based on forward and forward translation rules in Fact 2.20 below we can further conclude that G^T can be equivalently obtained via a source consistent forward transformation sequence based on forward rules without translation attributes.

By Fact 2.20 below we show that the model transformation sequences based on forward translation rules are one-to-one with model transformation sequences based on forward rules, i.e. based on source consistent forward sequences. For this reason, we can equivalently use both concepts and chose one of them depending on the particular needs. While the concept based on source consistency shows advantages in formal proofs, the concept based on forward translation rules shows advantages concerning analysis and efficiency as we will show in Sec. 3.1. The proof of Fact 2.20 is given in App. B.

Fact 2.20 (Equivalence of Forward Transformation and Forward Translation Sequences). Given a source model $G^S \in VL(TG^S)$, the sets of forward rules TR_F and corresponding forward translation rules TR_{FT}, then the following are equivalent for almost injective matches.

1 There is a model transformation sequence $(G^S, G_0 \xrightarrow{tr^*_F} G_n, G^T)$ based on TR_F with $G_0 = (G^S \leftarrow \varnothing \rightarrow \varnothing)$ and $G_n = (G^S \leftarrow G^C \rightarrow G^T)$

2 There is a model transformation sequence $(G^S, G'_0 \xrightarrow{tr^*_{FT}} G'_n, G^T)$ based on TR_{FT} with $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$ and $G'_n = (Att^{\mathbf{T}}(G^S) \leftarrow G^C \rightarrow G^T)$.

Moreover, the model transformation relation MTR_F for the model transformation based on forward rules coincides with the model transformation relation MTR_{FT} for the model transformation based on forward translation rules, i.e. $MTR_F = MTR_{FT}$.

3. Analysis of Functional Behaviour and Information Preservation

As shown in Sec. 2 before, we can ensure syntactical correctness and completeness for model transformations based on forward rules and equivalently for those based on forward translation rules using Fact 2.20. This section concentrates on the analysis of functional behaviour and information preservation.

3.1. *Functional Behaviour and Efficient Execution*

Functional behaviour of a model transformation means that each model of the source domain specific language (DSL) $\mathcal{L}_S$ is transformed into a unique model of the target language, where we require $\mathcal{L}_S \subseteq VL_S$ in order to ensure correctness and completeness by Thm. 1. The source DSL can form any subset of VL_S and it can be specified by the type graph TG^S together with additional well-formedness constraints. In many cases, model transformations should ensure the crucial property of functional behaviour. Moreover, in order to ensure efficient executions of model transformations, backtracking should be reduced or eliminated, respectively. Backtracking is necessary due to the possible choice of a suitable forward rule and match used for the translation of a particular source element. Therefore, backtracking is performed, if a transformation sequence terminates and is not completed successfully, because some parts of the source model have not been translated. This means, an execution of MT requires backtracking, if there are terminating TGT-sequences $(Att^F(G^S) \leftarrow \varnothing \rightarrow \varnothing) \xRightarrow{tr^*_{FT}} G'_n$ with $G'^S_n \neq Att^T(G^S)$. Termination of a forward translation sequence means that the construction of this sequence ends at a graph to which no further forward translation rule is applicable. As we will show by Thms. 2 and 3, functional behaviour and elimination of backtracking are closely related topics.

Definition 3.1 (Functional Behaviour of Model Transformations). Given a source DSL $\mathcal{L}_S \subseteq VL_S$, then a model transformation MT based on forward translation rules has *functional behaviour* if each execution of MT starting at a source model $G^S \in \mathcal{L}_S$ leads to a unique target model $G^T \in VL_T$.

$$\begin{tikzcd}
& K \arrow[dl, Rightarrow, "p_1{,}o_1"'] \arrow[dr, Rightarrow, "p_2{,}o_2"] & \\
P_1 \arrow[dr, Rightarrow, "*"'] & & P_2 \arrow[dl, Rightarrow, "*"] \\
& K' &
\end{tikzcd}$$

The standard way to analyse functional behaviour is to check whether the underlying transformation system is confluent, i.e. all diverging derivation paths starting at the same model finally meet again. According to Newman's Lemma (Newman1942), confluence can be shown by proving local confluence and additionally ensuring termination. More precisely, local confluence means that whenever a graph K can be transformed in one step into two graphs P_1 and P_2, these graphs can be transformed into a graph K', as shown in the diagram on the right.

Local confluence can be shown by checking confluence of all *critical pairs* $(P_1 \Leftarrow K \Rightarrow P_2)$, which represent the minimal objects where a confluence conflict may occur. A critical pair describes a minimal conflict, where minimality means that only overlappings of the rule components are considered for graph K. The technique is based on two results (see (Ehrig et al.2006)). On the one hand, the *completeness* of critical pairs implies that for every confluence conflict given by a pair of diverging transformation steps $(G_1 \Leftarrow G \Rightarrow G_2)$ there is a critical pair $(P_1 \Leftarrow K \Rightarrow P_2)$ which can be embedded into $(G_1 \Leftarrow G \Rightarrow G_2)$. On the other hand, the transformations $(P_1 \overset{*}{\Rightarrow} K' \overset{*}{\Leftarrow} P_2)$ obtained by confluence of the critical pair can be embedded into transformations $(G_1 \overset{*}{\Rightarrow} G' \overset{*}{\Leftarrow} G_2)$ that solve the original confluence conflict.

However, as shown by Plump (Plump1993; Plump2005), confluence of critical pairs is not sufficient for this purpose, but a slightly stronger version, called strict confluence is necessary, which additionally requires that the preserved elements of the given steps are preserved in the merging steps. This means that elements, which are not deleted by one of the original transformations steps, have to be preserved by the additional transformations which lead to confluence to ensure the applicability of the rules in the bigger context. This is necessary, because when extending such a transformation, a preserved node may be adjacent to an edge such that the deletion of this node would lead to dangling edges, i.e.

$$\begin{tikzcd}
 & K \arrow[dl, Rightarrow, "p_1{,}o_1"'] \arrow[dr, Rightarrow, "p_2{,}o_2"] & \\
P_1 \arrow[dr, Rightarrow, "*"'] & & P_2 \arrow[dl, Rightarrow, "*"] \\
 & K' &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & & K & & \\
 & N_1 \arrow[ur, "v_1"] \arrow[dl, "w_1"'] & (1) & N_2 \arrow[ul, "v_2"'] \arrow[dr, "w_2"] & \\
P_1 & (2) & N \arrow[ul, "z_1"] \arrow[ur, "z_2"'] \arrow[dl, "z_3"] \arrow[dr, "z_4"'] & (3) & P_2 \\
 & N_3 \arrow[ul, "v_3"] \arrow[dr, "w_3"'] & (4) & N_4 \arrow[ur, "v_4"'] \arrow[dl, "w_4"] & \\
 & & K' & &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & G & \\
 & K \arrow[u, "k_0"'] \arrow[dl, Rightarrow, "p_1{,}o_1"'] \arrow[dr, Rightarrow, "p_2{,}o_2"] & \\
P_1 \arrow[dr, Rightarrow, "t_1"', "*"] & & P_2 \arrow[dl, Rightarrow, "t_2", "*"'] \\
 & K' &
\end{tikzcd}$$

Fig. 10. NAC-strict confluence

the additional transformations were not applicable in the larger context. This result is also valid for typed attributed graph transformation systems (Ehrig et al.2006; Lambers2009) and we apply them to show functional behaviour of model transformations.

Furthermore, in the presence of NACs, we also have to ensure that NAC-consistency of the merging steps is implied by the NAC-consistent diverging steps of the critical pair. Again, this property ensures that the confluent transformations of the critical pair can be embedded into a larger context. NAC-consistency of an embedding $k : G \to G'$ for a transformation step $G \xRightarrow{p,m} H$ implies that there is a transformation step $G' \xRightarrow{p,m'} H'$ with $m' = k \circ m$ which satisfies the NACs of p. Concerning a transformation sequence, NAC-consistency can be checked by constructing the concurrent rule of the sequence (Lambers2009), which combines the involved NACs in a suitable way. If an embedding morphism fulfills the NACs of the original rules, the critical pair can be embedded into the larger context. To ensure the embedding of the additional transformations, the embedding morphism has to fulfill all occurring NACs to ensure the applicability of the transformation. Otherwise, we may have an embedding of a critical pair that is not confluent.

Let us recall the basic notions for confluence of critical pairs according to (Ehrig et al.2006; Lambers2009).

Definition 3.2 (NAC-strict Confluence of Critical Pairs). A critical pair $CP = (P_1 \xLeftarrow{p_1,o_1} K \xRightarrow{p_2,o_2} P_2)$ is called *strictly confluent*, if we have the following:

1 *Confluence*: the critical pair is confluent, i.e. there are transformations $t_1 : P_1 \overset{*}{\Rightarrow} K'$ and $t_2 : P_2 \overset{*}{\Rightarrow} K'$ with derived spans $der(t_i) = (P_i \xleftarrow{v_{i+2}} N_{i+2} \xrightarrow{w_{i+2}} K')$ for $i = 1, 2$.

2 *Strictness*: Let $der(K \xRightarrow{p_i,o_i} P_i) = (K \xleftarrow{v_i} N_i \xrightarrow{w_i} P_i)$ for $i = 1, 2$, and let N be the pullback object of the pullback (1). Then, there are morphisms z_3 and z_4 such that (2), (3), and (4) in Fig. 10 commute.

3 *NAC-consistency*: For every injective morphism $k_0 : K \to G$ that is NAC consistent with respect to $K \xRightarrow{p_1,o_1} P_1$ and $K \xRightarrow{p_2,o_2} P_2$ in Fig. 10 it follows that k_0 is also NAC consistent with respect to t_1 and t_2 .

However, while termination of model transformations based on forward rules resp. forward translation rules can be ensured quite easily by checking that all *TGG*-triple rules are creating on the source component, this is not the case for local confluence. In fact, the system of forward translation rules of our case study *CD2RDBM* is not locally confluent, but we can show in Ex. 3.15 that the model transformation has functional

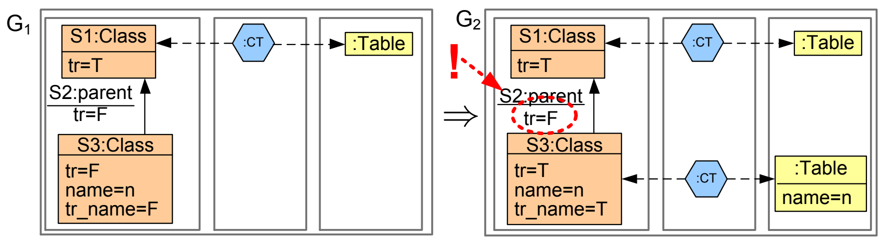

Fig. 11. Step $G_1 \xrightarrow{Class2Table_{FT}} G_2$ with misleading graph G_2

behaviour. Indeed, functional behaviour of a model transformation does not require general confluence of the underlying system of operational rules. Confluence only needs to be ensured for transformation paths which lead to completely translated models. More precisely, derivation paths leading to a point for backtracking do not influence the functional behaviour. For this reason, we introduce so-called *filter NACs* that extend the model transformation rules in order to avoid misleading paths that cause backtracking, such that the backtracking for the extended system is reduced substantially. By Fact 3.9 we ensure that the overall behaviour of the model transformation with respect to the model transformation relation is still preserved. As first important result we show by Thm. 2 that functional behaviour of a model transformation is ensured by termination and strict confluence of all significant critical pairs of the system of forward translation rules enriched by filter NACs, where significant critical pairs are a subset of all critical pairs. Furthermore, we are able to characterize strong functional behaviour of a terminating model transformation based on forward translation rules with filter NACs in Thm. 3 by the condition that there is no significant critical pair at all. Compared with functional behaviour we additionally ensure by strong functional behaviour that the model transformation sequences are unique up to switch equivalence.

The addition of filter NACs therefore has two advantages. On the one hand, the analysis of functional behaviour is improved, because the possible conflicts between the transformation rules are reduced and we will show in this section that filter NACs allow us to verify functional behaviour for our case study *CD2RDBM*. On the other hand, filter NACs improve the efficiency of the execution by cutting off possible backtracking paths. Filter NACs are based on the following notion of misleading graphs, which can be seen as model fragments that are responsible for the backtracking of a model transformation.

Definition 3.3 (Translatable and Misleading Graphs). A triple graph with translation attributes G is *translatable* if there is a transformation sequence $G \xrightarrow{tr^*_{FT}} H$ via forward translation rules such that H is completely translated (see Def. 2.17). A triple graph with translation attributes G is *misleading*, if every triple graph G' with translation attributes and $G' \supseteq G$ is not translatable.

Example 3.4 (Misleading Graph). Consider the transformation step shown in Fig. 11. The resulting graph G_2 is misleading according to Def. 3.3, because the edge $S2$ is labelled with a translation attribute set to "**F**", but there is no rule which may change this attribute in any bigger context at any later stage of the transformation. The only rule which changes the translation attribute of a "parent"-edge is "*Subclass2Table*$_{FT}$", but it requires that the source node $S3$ is labelled with a translation attribute set to "**F**". However, forward translation rules do not modify translation attributes if they are set to "**T**" already and additionally do not change the structure of the source component.

Definition 3.5 (Filter NAC). A filter NAC n for a forward translation rule $tr_{FT} : L_{FT} \leftarrow K_{FT} \rightarrow R_{FT}$ is given by a morphism $n : L_{FT} \rightarrow N$, such that there is a TGT step $N \xRightarrow{tr_{FT},n} M$ with M being misleading. The extension of tr_{FT} by some set of filter NACs is called forward translation rule tr_{FN} with filter NACs.

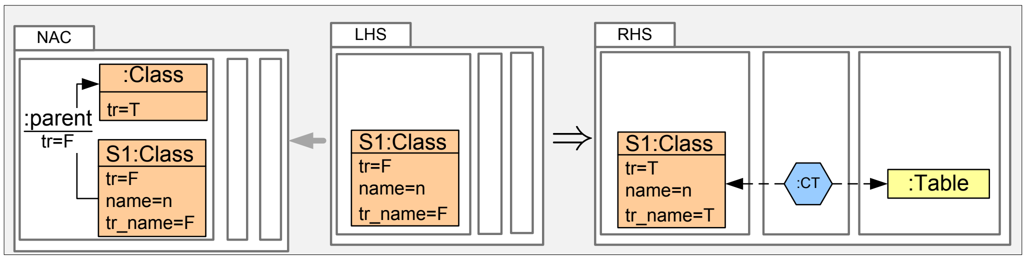

Fig. 12. A forward translation rule with filter NAC: *Class2Table*$_{FN}$

Example 3.6 (Forward Translation Rule with Filter NACs). The rule *Class2Table*$_{FT}$ is extended by a filter NAC in Fig. 12, which is obtained from the graph G_1 of the transformation step $G_1 \xRightarrow{Class2Table_{FT}} G_2$ in Fig. 11, where G_2 is misleading according to Ex. 3.4. In Fact 3.7 below we present how such filter NACs are generated automatically. In Ex. 3.15 we will extend the rule by a further similar filter NAC with "$tr = \mathbf{T}$" for node "$S1$".

A direct construction of filter NACs according to Def. 3.5 would be inefficient, because the size of the considered graphs to be checked is unbounded. For this reason we now present efficient techniques which support the generation of filter NACs and allow us to bound the size without losing generality. At first we present an automated technique for a subset of filter NACs and thereafter, an interactive generation technique leading to a much larger set of filter NACs. The first procedure in Fact 3.7 below is based on a sufficient criterion for checking the misleading property. Concerning our example this automated generation leads to the filter NAC shown in Fig. 12 for the rule *Class2Table*$_{FT}$ for an incoming edge of type "*parent*".

Fact 3.7 (Automated Generation of Filter NACs). Given a triple graph grammar, then the following procedure applied to each triple rule $tr \in TR$ generates filter NACs for the derived forward translation rules TR_{FT} leading to forward translation rules TR_{FN} with filter NACs:

— Outgoing Edges: Check whether the following properties hold

  - tr creates a node $(x : T_x)$ in the source component and the type graph allows outgoing edges of type " T_e" for nodes of type " T_x", but tr does not create an edge $(e : T_e)$ with source node x.
  - Each rule in TR which creates an edge $(e : T_e)$ also creates its source node.
  - Extend L_{FT} to N by adding an outgoing edge $(e : T_e)$ at x together with a target node. Add a translation attribute for e with value **F**. The inclusion $n : L_{FT} \to N$ is a NAC-consistent match for tr.

  For each node x of tr fulfilling the above conditions, the filter NAC $(n : L_{FT} \to N)$ is generated for tr_{FT} leading to tr_{FN}.
- Incoming Edges: Dual case, this time for an incoming edge $(e : T_e)$.
- TR_{FN} is the extension of TR_{FT} by all filter NACs constructed above.

Proof. See App. B. □

The following interactive technique for deriving filter NACs is based on the generation of critical pairs, which define conflicts of rule applications in a minimal context. By the completeness of critical pairs (Lemma 6.22 in (Ehrig et al.2006)) we know that for each pair of two parallel dependent transformation steps there is a critical pair which can be embedded. If a critical pair $P_1 \xLeftarrow{tr_{1,FT}} K \xRightarrow{tr_{2,FT}} P_2$ contains a misleading graph P_1, we use the overlapping graph K as a filter NAC of the rule $tr_{1,FT}$. However, checking the misleading property needs manual interaction. But in some cases, these manual results of identified misleading graphs can be reused for more general static conditions. Indeed, the conditions used in Fact 3.7 were inspired by first performing the interactive method to our case study. Moreover, we are currently working on a technique that uses a sufficient criteria to check the misleading property automatically, and we are confident that this approach will provide a powerful generation technique.

Fact 3.8 (Interactive Generation of Filter NACs). Given a set of forward translation rules, then generate the set of critical pairs $P_1 \xLeftarrow{tr_{1,FT},m_1} K \xRightarrow{tr_{2,FT},m_2} P_2$. If P_1 (or similarly P_2) is misleading, we generate a new filter NAC $m_1 : L_{1,FT} \to K$ for $tr_{1,FT}$ leading to $tr_{1,FN}$, such that $K \xRightarrow{tr_{1,FN},m_1} P_1$ violates the filter NAC. Hence, the critical pair for $tr_{1,FT}$ and $tr_{2,FT}$ is no longer a critical pair for $tr_{1,FN}$ and $tr_{2,FT}$. But this construction may lead to new critical pairs for the forward translation rules with filter NACs. The procedure is repeated until no further filter NAC can be found or validated. This construction starting with TR_{FT} always terminates if the structural part of each graph of a rule is finite.

Proof. See App. B. □

Based on the flattening construction presented in (Ehrig et al.2008) we derive an equivalent plain graph transformation system from the system of forward translation rules. Since the system of forward translation rules ensures source consistency for complete transformation sequences by construction, the derived flattened grammar also ensures source consistency for complete transformation sequences. For this reason, we do not need to extend the analysis techniques for critical pairs and can use the critical pair analysis engine of AGG (AGG2011).

Concerning our case study *CD2RDBM*, the interactive generation terminates after the second round, which is typical for practical applications, because the amount of already translated elements in the new occurring critical pairs usually decreases. Furthermore, several NACs can be combined if they differ only on some translation attributes. According to Fact 3.9 below, filter NACs do not change the behaviour of model transformations. The only effect is that they filter out derivation paths, which would lead to misleading graphs, i.e. to backtracking for the computation of the model transformation sequence. This means that the filter NACs filter out backtracking paths.

Fact 3.9 (Equivalence of Transformations with Filter NACs). Given a triple graph grammar $TGG = (TG, \varnothing, TR)$ with forward translation rules TR_{FT} and filter NACs leading to TR_{FN}. Let $G_0 = (G^S \leftarrow \varnothing \rightarrow \varnothing)$ be a triple graph typed over TG and $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$, then the following are equivalent for almost injective matches:

1 There is a complete TGT-sequence $G'_0 \xRightarrow{tr^*_{FT}, m^*_{FT}} G'$ via TR_{FT}.

2 There is a complete TGT-sequence $G'_0 \xRightarrow{tr^*_{FN}, m^*_{FT}} G'$ via TR_{FN}.

Proof. See App. B. □

Concerning termination of a system of forward translation rules according to Def. 3.10, we have the following Fact 3.11 according to Thm. 1 in (Hermann et al.2010a).

Definition 3.10 (Termination). A system of forward translation rules TR_{FT} is terminating, if each transformation sequence via TR_{FT} is terminating, i.e. the sequence ends at a graph to which no further forward translation rule is applicable.

Fact 3.11 (Termination). Given TR_{FN} and TR_{FT} as in Fact 3.9, then TR_{FN} is terminating if TR_{FT} is terminating. A sufficient condition for termination of TR_{FT} is that all graphs are finite on the graph part and each rule modifies at least one translation attribute from false to true. Termination of TR_{FN} with strict confluence of critical pairs implies unique normal forms by the local confluence theorem in (Lambers2009).

In order to analyse functional behaviour we generate the critical pairs for the system of forward translation rules and show by Thm. 2 that strict confluence of "significant" critical pairs ensures functional behaviour. A critical pair is significant if it can be embedded into two transformation sequences via forward translation rules that start at the same source model G^S, which belongs to the source domain specific language $\mathcal{L}_S$. This implies that a critical pair containing a misleading graph automatically is not significant. For this reason, some of the non-significant critical pairs can be eliminated already with the presented automatic and interactive techniques for generating filter NACs in Facts 3.7 and 3.8.

Definition 3.12 (Significant Critical Pair). A critical pair $(P_1 \xLeftarrow{tr_{1,FN}} K \xRightarrow{tr_{2,FN}} P_2)$ for a set of forward translation rules with filter NACs TR_{FN} is called significant if it can be embedded into a parallel dependent pair $(G'_1 \xLeftarrow{tr_{1,FN}} G' \xRightarrow{tr_{2,FN}} G'_2)$ such that there is $G^S \in \mathcal{L}_S \subseteq VL_S$ and $G'_0 \xRightarrow{tr^*_{FN}} G'$ with $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$.

$$\begin{tikzcd}
G'_0 \arrow[r, Rightarrow, "tr^*_{FN}"] & G' \arrow[r, Rightarrow, "tr_{1,FN}"] \arrow[dr, Rightarrow, "tr_{2,FN}"'] & G1' \\
& & G'_2
\end{tikzcd}$$

Theorem 2 (Functional Behaviour). Let MT_{FT} be a model transformation based on forward translation rules TR_{FT} with model transformation relation MTR_{FT} and source DSL $\mathcal{L}_S$. Furthermore, let TR_{FN} extend TR_{FT} with filter NACs such that TR_{FN} is terminating and all significant critical pairs are strictly confluent. Then, MT_{FT} has functional behaviour. Moreover, the model transformation MT_{FN} based on TR_{FN} does not require backtracking and MT_{FN} defines the same model transformation relation, i.e. $MTR_{FN} = MTR_{FT}$.

Proof of Thm. 2. For functional behaviour of the model transformation we have to show that each source model $G^S \in \mathcal{L}_S$ is transformed into a unique (up to isomorphism) completely translated target model G^T, which means that there is a completely translated triple model G' with $G'^T = G^T$, and furthermore $G^T \in VL_T$.

For $G^S \in \mathcal{L}_S \subseteq VL_S$ we have by definition of VL that there is a $G^T \in VL_T$ and a TGT-sequence $\varnothing \xRightarrow{tr^*} (G^S \leftarrow G^C \rightarrow G^T)$ via TR and using the decomposition theorem with NACs in (Ehrig et al.2009b) we obtain a match consistent TGT-sequence $\varnothing \xRightarrow{tr^*_S} (G^S \leftarrow \varnothing \rightarrow \varnothing) \xRightarrow{tr^*_F} (G^S \leftarrow G^C \rightarrow G^T)$ and by Fact 2.20 a complete TGT-sequence $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing) \xRightarrow{tr^*_{FT}} (Att^{\mathbf{T}}(G^S) \leftarrow G^C \rightarrow G^T) = G'$. This means that $(G^S, G'_0 \xRightarrow{tr^*_{FT}} G', G^T)$ is a model transformation sequence based on TR_{FT}. Assume that we also have a complete forward translation sequence $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing) \xRightarrow{\overline{tr^*_{FT}}} (Att^{\mathbf{T}}(G^S) \leftarrow \overline{G^C} \rightarrow \overline{G^T}) = \overline{G}'$. By Fact. 3.9 we also have the complete TGT-sequences $(G^S, G'_0 \xRightarrow{tr^*_{FN}} G', G^T)$ and $G'_0 \xRightarrow{tr^*_{FN}} G'$ and $G'_0 \xRightarrow{tr^*_{FN}} \overline{G}'$. Using the precondition that TR_{FN} is terminating and all significant critical pairs are strictly confluent we show that all diverging transformation sequences can be merged again. Consider the possible transformation sequences starting at G'_0 (which form a graph of transformation steps) and two diverging steps $(G'_{i+1} \xLeftarrow{p_1,m_1} G'_i \xRightarrow{p_2,m_2} G''_{i+1})$. If they are parallel independent, we can apply the local Church-Rosser theorem (LCR) (Lambers2009) and derive the merging steps $(G'_{i+1} \xRightarrow{p_2,m'_2} H \xLeftarrow{p_1,m'_1} G''_{i+1})$. If they are parallel dependent diverging steps we know by completeness of critical pairs (see Thm. 3.7.6 in (Lambers2009)) that there is a critical pair and by Def. 3.12 we know that this pair is significant, because we consider transformations sequences starting at G'_0. This pair is strictly confluent by precondition. Therefore, these steps can be merged again. Now, any new diverging situation can be merged by either LCR for parallel independent steps or by strict confluence of critical pairs for parallel dependent steps. By precondition the system is terminating. In combination, this implies that $G' \cong \overline{G}'$ and hence, $G^T \cong \overline{G}^T$.

Backtracking is not required, because termination of TR_{FN} with strict confluence of significant critical pairs implies unique normal forms as shown above. Therefore, any terminating TGT-sequence $(Att^F(G^S) \leftarrow \varnothing \rightarrow \varnothing) \xRightarrow{tr^*_{FN}} G'_n$ leads to a unique G'_n up to isomorphism and by correctness and completeness (Thm. 1 and Fact 2.20) we have that ${G'}_n^S = Att^T(G^S)$.

The model transformation relation is the same, because we have by Fact 3.9 the equivalence of the model transformation sequences. □

If the set of generated critical pairs of a system of forward translation rules with filter NACs TR_{FN} is empty, we can directly conclude from Thm. 2 that the corresponding system TR_{FT} without filter NACs has functional behaviour. Moreover, from an efficiency point of view, the set of rules should be compact in order to minimize the effort for pattern matching. In the optimal case, the rule set ensures that each transformation sequence of the model transformation is itself unique up to switch equivalence meaning that it is unique up to the order of sequentially independent steps. For this reason, we introduce the notion of strong functional behaviour with respect to a given source domain specific language (DSL) $\mathcal{L}_S$. Note, that two transformation sequences are called switch-equivalent, if they can be obtained from each other by switching consecutive sequentially independent transformation steps, which is possible according to the Local Church-Rosser Theorem (Ehrig et al.2006; Lambers2009).

Definition 3.13 (Strong Functional Behaviour of Model Transformations). A model transformation based on forward translation rules TR_{FN} with filter NACs and the source DSL $\mathcal{L}_S \subseteq VL_S$ has *strong functional behaviour* if for each $G^S \in \mathcal{L}_S$ there is a $G^T \in VL_T$ and a model transformation sequence $(G^S, G'_0 \xRightarrow{tr^*_{FN}} G'_n, G^T)$ based on forward translation rules, and moreover,

— any partial TGT-sequence $G'_0 \xRightarrow{tr^{i,*}_{FN}} G'_i$ terminates, i.e. there are finitely many extended sequences $G'_0 \xRightarrow{tr^{i,*}_{FN}} G'_i \xRightarrow{tr^{j,*}_{FN}} G'_j$, and

— each two TGT-sequences $G'_0 \xRightarrow{tr^*_{FN}} G'_n$ and $G'_0 \xRightarrow{\overline{tr}^*_{FN}} \overline{G}'_m$ with completely translated graphs G'_n and $\overline{G}'_m$ are switch-equivalent up to isomorphism.

Remark 3.14 (Strong Functional Behaviour).

1 The sequences are terminating means that no rule in TR_{FN} is applicable any more. However, it is not required that the sequences are complete, i.e. that G'_n and $\overline{G}'_m$ are completely translated.

2 Strong functional behaviour implies functional behaviour, because G'_n and $\overline{G}'_m$ completely translated implies that $G'_0 \xRightarrow{tr^*_{FN}} G'_n$ and $G'_0 \xRightarrow{\overline{tr}^*_{FN}} \overline{G}'_m$ are terminating TGT-sequences.

3 Two sequences $t1 : G_0 \Rightarrow^* G_1$ and $t2 : G_0 \Rightarrow^* G_2$ are called switch-equivalent, written $t1 \approx t2$, if $G_1 = G_2$ and $t2$ can be obtained from $t1$ by switching sequential independent steps according to the local Church-Rosser theorem with NACs (Lambers2009). The sequences $t1$ and $t2$ are called switch-equivalent up to isomorphism if $t1 : G_0 \Rightarrow^* G_1$ has an isomorphic sequence $t1' : G_0 \Rightarrow^* G_2$ (using the same sequence of rules) with $i : G_1 \xrightarrow{\sim} G_2$, written $trace(t1') = i \circ trace(t1)$, such that $t1' \approx t2$. This means especially that the rule sequence in $t2$ is a permutation of that in $t1$.

The third main result of this paper shows that strong functional behaviour of model transformations based on forward translation rules with filter NACs can be completely characterized by the absence of significant critical pairs.

Theorem 3 (Strong Functional Behaviour). A model transformation based on terminating forward translation rules TR_{FN} with filter NACs has strong functional behaviour and does not require backtracking leading to polynomial time complexity if and only if TR_{FN} has no significant critical pair.

Proof. **Direction "$\Leftarrow$":** Assume that TR_{FN} has no significant critical pair. Similar to the proof of Thm. 2 we obtain for each $G^S \in \mathcal{L}_S$ a $G^T \in VL_T$ and a complete TGT-sequence $G'_0 \xRightarrow{tr^*_{FT}} G'$ and a model transformation $(G^S, G'_0 \xRightarrow{tr^*_{FT}} G', G^T)$ based on TR_{FT} underlying TR_{FN}. By Fact. 3.9 we also have a complete TGT-sequence $G'_0 \xRightarrow{tr^*_{FN}} G'$ and hence, also a model transformation $(G^S, G'_0 \xRightarrow{tr^*_{FT}} G', G^T)$ based on TR_{FT} underlying TR_{FN}. In order to show strong functional behaviour let $G'_0 \xRightarrow{tr^*_{FN}} G'_n$ and $G'_0 \xRightarrow{\overline{tr}^*_{FN}} \overline{G}'_m$ be two terminating TGT-sequences with $m, n \geq 1$. We have to show that they are switch-equivalent up to isomorphism. We show by induction on the combined length $n + m$ that both sequences can be extended to switch-equivalent sequences.

For $n + m = 2$ we have $n = m = 1$ with $t1 : G'_0 \xRightarrow{tr_{FN}, m} G'_1$ and $\overline{t1} : G'_0 \xRightarrow{\overline{tr}_{FN}, \overline{m}} \overline{G}'_1$. If $tr_{FN} = \overline{tr}_{FN}$ and $m = \overline{m}$, then both are isomorphic with isomorphism $i : \overline{G}'_1 \xrightarrow{\sim} G'_1$, such that $t1 \approx i \circ \overline{t1}$. If not, then $t1$ and $\overline{t1}$ are parallel independent, because otherwise we would have a significant critical pair by completeness of critical pairs in (Lambers2009). By the local Church-Rosser theorem (Lambers2009) we have $t2 : G'_1 \xRightarrow{\overline{tr}_{FN}} G'_2$ and $\overline{t2} : \overline{G}'_1 \xRightarrow{tr_{FN}} G'_2$, such that $t2 \circ t1 \approx \overline{t2} \circ \overline{t1} : G'_0 \Rightarrow^* G'_2$.

Now assume that for $t1 : G'_0 \Rightarrow^* G'_{n-1}$ and $\overline{t1} : G'_0 \Rightarrow^* \overline{G}'_m$ we have extensions $t2 : G'_{n-1} \Rightarrow^* H$, $\overline{t2} : \overline{G}'_m \Rightarrow^* H$, such that $t2 \circ t1 \approx \overline{t2} \circ \overline{t1}$.

$$\begin{tikzcd}
G'_0 \arrow[r, Rightarrow, "t1"', "*"] \arrow[d, Rightarrow, "\overline{t1}"', "*"] & G'_{n-1} \arrow[r, Rightarrow, "t"] \arrow[d, Rightarrow, "t2", "*"'] & G'_n \arrow[d, Rightarrow, "t3", "*"'] \\
\overline{G}'_m \arrow[r, Rightarrow, "\overline{t2}"', "*"] & H \arrow[r, Rightarrow, "\overline{t3}"', "*"] & K
\end{tikzcd}$$

For a step $t : G'_{n-1} \Rightarrow G'_n$, then we have to show that $t \circ t1$ and $\overline{t1}$ can be extended to switch-equivalent sequences. By induction hypothesis and definition of significant critical pairs also t and $t2$ can be extended by $t3 : G'_n \Rightarrow^* K$, $\overline{t3} : H \Rightarrow^* K$, such that $t3 \circ t \approx \overline{t3} \circ t2$. Now, composition closure of switch equivalence implies $t3 \circ t \circ t1 \approx \overline{t3} \circ \overline{t2} \circ \overline{t1} : G'_0 \Rightarrow^* K$. This completes the induction proof.

Now, we use that G'_n and $\overline{G}'_m$ are both terminal which implies that $t3$ and $\overline{t3} \circ \overline{t2}$ must be isomorphisms. This shows that $G'_0 \xRightarrow{tr^*_{FN}} G'_n$ and $G'_0 \xRightarrow{\overline{tr}^*_{FN}} \overline{G}'_m$ are switch-equivalent up to isomorphism.

Direction "$\Rightarrow$": Assume now that TR_{FN} has strong functional behaviour and that TR_{FN} has a significant critical pair. We have to show a contradiction in this case.

Let $P_1 \xLeftarrow{tr_{1,FN}} K \xRightarrow{tr_{2,FN}} P_2$ be the significant critical pair which can be embedded into a parallel dependent pair $G_1 \xLeftarrow{tr_{1,FN}} G' \xRightarrow{tr_{2,FN}} G_2$, such that there is $G^S \in \mathcal{L}_S$

with $G'_0 \xRightarrow{tr^*_{FN}} G'$ and $G'_0 = (Att^{\mathbf{F}}(G^S) \leftarrow \varnothing \rightarrow \varnothing)$. Since TR_{FN} is terminating we have terminating sequences $G_1 \Rightarrow^* G_{1n}$ and $G_2 \Rightarrow^* G_{2m}$ via TR_{FN}. By composition we have the following terminating TGT-sequences

1 $G'_0 \xRightarrow{tr_{FN}} G' \xRightarrow{tr_{1,FN}} G_1 \Rightarrow^* G_{1n}$
2 $G'_0 \xRightarrow{tr_{FN}} G' \xRightarrow{tr_{2,FN}} G_2 \Rightarrow^* G_{2m}$

Since TR_{FN} has strong functional behaviour both are switch-equivalent up to isomorphism. For simplicity assume $G_{1n} = G_{2m}$ instead of $G_{1n} \cong G_{2m}$. This implies $n = m$ and $G' \xRightarrow{tr_{1,FN}} G_1 \Rightarrow^* G_{1n}$ switch-equivalent to $G' \xRightarrow{tr_{2,FN}} G_2 \Rightarrow^* G_{1n}$. This means $tr_{2,FN}$ occurs in $G_1 \Rightarrow^* G_{1n}$ and can be shifted in $G' \xRightarrow{tr_{1,FN}} G_1 \Rightarrow^* G_{1n}$, such that we obtain $G' \xRightarrow{tr_{2,FN}} G_2 \Rightarrow^* G_{1n}$.

But this implies that in an intermediate step we can apply the parallel rule $tr_{1,FN} + tr_{2,FN}$ leading to parallel independence of $G' \xRightarrow{tr_{1,FN}} G_1$ and $G' \xRightarrow{tr_{2,FN}} G_2$, which is a contradiction. Hence, TR_{FN} has no significant critical pair.

It remains to show that strong functional behaviour implies that backtracking is not required. This is a direct consequence of Thm. 2, since we have no significant critical pair and therefore, all of them are strictly confluent. □

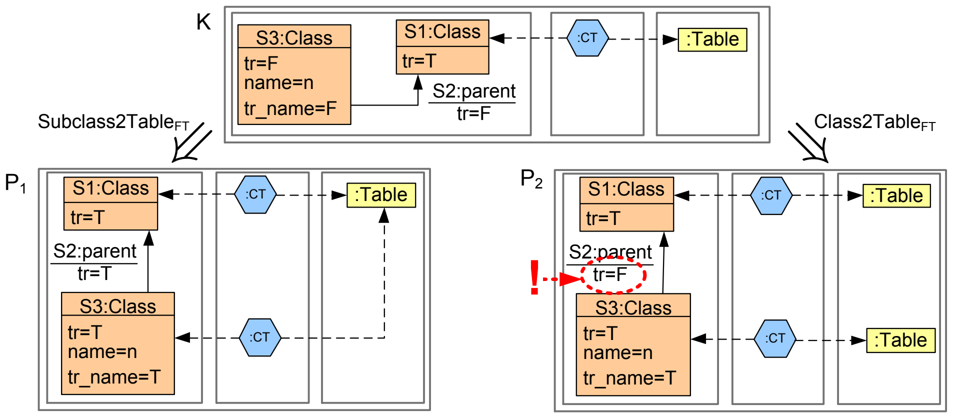

Fig. 13. Critical pair for the rules *Subclass2Table$_{FT}$* and *Class2Table$_{FT}$*

Example 3.15 (Functional and Strong Functional Behaviour). We analyse functional behaviour of the model transformation *CD2RDBM*. By Fact 3.11, *CD2RDBM* is terminating, because all *TGG*-triple rules are creating in the source component. For analysing local confluence we use the tool AGG (AGG2011) for the generation of critical pairs. The set of derived forward translation rules from the rules *TR* in Figs. 2 and 4 is given by $TR_{FT} = \{Class2Table_{FT}, Subclass2Table_{FT}, Attr2Column_{FT}, PrimaryAttr2Column_{FT}, Association2ForeignKey_{FT}\}$. We exchange the forward translation rule *Class2Table$_{FT}$* by the extended rule with filter NACs *Class2Table$_{FN}$* as shown in Fig. 12, and additionally extend it by a further filter NAC obtained by the automated generation according to Fact 3.7. We use AGG (version 2.0) for generating the critical pairs. AGG detects three critical pairs for conflicts of the rule "*PrimaryAttr2Column*" with itself. The corresponding overlapping graphs K of the critical pairs contain two primary attribute nodes which belong to classes that are connected to the same table. This implies that the resulting graphs P_1 and P_2 of each critical pair ($P_1 \Leftarrow K \Rightarrow P_2$) are misleading, because the remaining untranslated primary attribute of the initially two cannot be translated in any bigger context due to the source NAC of the rule and because no other rule translates a primary attribute. Therefore, all critical pairs lead to additional filter NACs by the interactive generation of filter NACs in Fact 3.8. For the resulting system of forward translation rules with filter NACs, AGG does not generate any critical pair. Thus, we can apply Thm. 3 and show that the model transformation based on the forward translation rules with filter NACs TR_{FN} has *strong functional behaviour* and does not require backtracking. Furthermore, by Thm. 2 we can conclude that the model transformation based on the forward translation rules TR_{FT} without filter NACs has *functional behaviour*. As an example, Fig. 9 shows the resulting triple graph of a model transformation starting with the class diagram G^S.

3.2. *Information Preservation*

Model transformations are information preserving if their corresponding backward transformations can be used to derive parts of the given source model from a target model that was derived via a forward transformation. In fact, several TGG tools do not support backtracking and use optimizations, such that they cannot ensure completeness. This implies that the execution of backward transformations may stop without creating a valid source model for some target models (Giese et al.2010; Schürr and Klar2008; Klar et al.2010). This section provides results for analysing and ensuring information preservation for TGG model transformations according to Sec. 2. In particular, we analyse whether and how a source model can be reconstructed from the computed target model.

For this purpose, we distinguish forward and backward model transformations. Interestingly, it turns out that complete information preservation, i.e. the complete reconstruction of the source model, is ensured by functional behaviour of the backward model transformation. We present the techniques for model transformations based on forward rules. According to the equivalence result in Fact 2.20, we also know that these techniques provide the same results for model transformations based on forward translation rules. Moreover, due to the symmetric definition of TGGs, the results can be applied dually for backward model transformations.

Definition 3.16 (Information Preserving Model Transformation). A forward model transformation based on forward rules is *information preserving*, if for each forward model transformation sequence $(G^S, G_0 \xRightarrow{tr_F^*} G_n, G^T)$ there is a backward model transformation sequence $(G^T, G'_0 \xRightarrow{tr'^*_B} G'_m, G'^S)$ with $G^S = G'^S$, i.e. the source model G^S can be reconstructed from the resulting target model G^T via a target consistent backward transformation sequence.

By Thm. 4 we show that model transformations based on forward rules are information preserving.

Theorem 4 (Information Preserving Model Transformation). Each forward model transformation based on forward rules is information preserving.

Proof. Given a set of triple rules TR with derived forward rule TR_F and backward rules TR_B. By Fact 2.9 and Rem. 2.10 applied to the source consistent forward sequence $G_0 \xRightarrow{tr_F^*} G_n$ via TR_F we derive the target consistent backward transformation $G'_0 = (G^T \leftarrow \varnothing \rightarrow \varnothing) \xRightarrow{tr_B^*} G_n$ via TR_B with $G_n^S = G^S$. This means that we have a backward model transformation sequence $(G^T, G'_0 \xRightarrow{tr_B^*} G_n, G'^S)$ with $G^S = G'^S$. $\square$

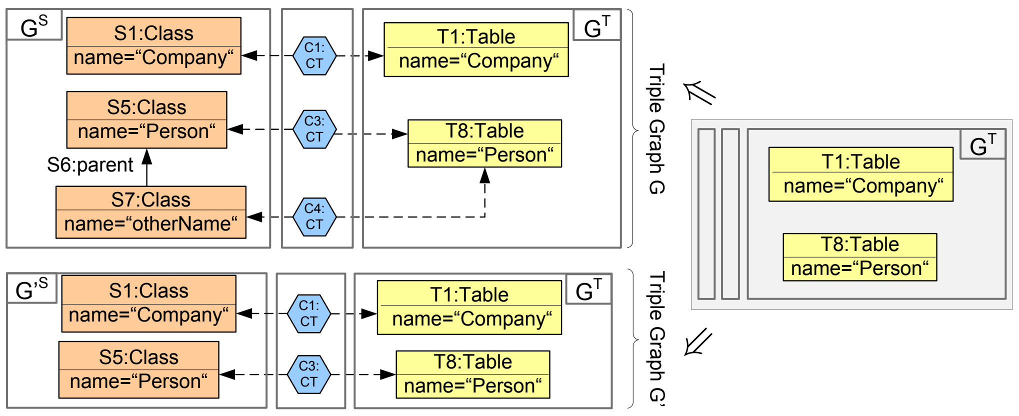

Fig. 14. Two possible target consistent backward transformations

Example 3.17 (Information Preserving Model Transformation *CD2RDBM*). The model transformation *CD2RDBM* is information preserving, because it consists of model transformation sequences based on forward rules, which ensure source consistency of the forward sequences by definition. Therefore, the presented source model G^S of the triple graph in Fig. 9 can be reconstructed by a target consistent backward transformation sequence starting at the model $G'_0 = (\varnothing \leftarrow \varnothing \rightarrow G^T)$. But there are several possible target consistent backward transformation sequences starting at G'_0. The reason is that the rule $Subclass2Table_B$ can be applied arbitrarily often without having an influence concerning the target consistency, because the rule is identical on the target component. This means that the inheritance information within a class diagram has no explicit counterpart within a relational data base model.

There are many possible target consistent backward transformation sequences for the same derived target model G^T where two of them are presented in Fig. 14. The source model G^S can be transformed into $G = (G^S \leftarrow G^C \rightarrow G^T)$. But starting with G^T, both depicted backward transformation sequences are possible and target consistent. The resulting source graphs G^S and G'^S, however, differ with respect to the class node

$S7$ and the edge $S6$ in G^S. Hence, some information of G^S cannot be reconstructed uniquely and therefore, are partially lost in the target model G^T.

According to Thm. 4 each model transformation based on forward rules is information preserving. But the reconstruction of a corresponding source model from a derived target model is in general not unique. In order to ensure uniqueness of the reconstruction we now present the notion of complete information preservation. This stronger notion ensures that all information contained in a source model of a source domain specific language (DSL) can be reconstructed from the derived target model itself. More precisely, starting with the target model, each backward model transformation sequence will produce the original source model. This ensures that only one backward model transformation sequence has to be constructed. Intuitively, this means that the model transformation is invertible.

Definition 3.18 (Complete Information Preservation). A forward model transformation with source DSL $\mathcal{L}_S$ is *completely information preserving* if it is information preserving and furthermore, given a source model $G^S \in \mathcal{L}_S$ and the resulting target model G^T of a forward model transformation sequence, then each partial backward transformation sequence starting with G^T terminates and produces the given source model G^S as result.

We can verify complete information preservation by showing functional behaviour of the corresponding backward model transformation with respect to the derived target models $\mathcal{L}'_T \subseteq MT(\mathcal{L}_S) \subseteq VL_T$.

Theorem 5 (Completely Information Preserving Model Transformation). Given a forward model transformation MT. Then, MT is completely information preserving if the corresponding backward model transformation according to Rem. 2.10 has functional behaviour with respect to the target language $\mathcal{L}'_T = MT(\mathcal{L}_S)$.

Proof. By Thm. 4 we know that MT is information preserving. For a model transformation sequence $(G^S, G_0 \xRightarrow{tr_F^*} G_n, G^T)$, we additionally know that $G^T \in VL_T$ by Thm. 1, and furthermore, that $G^T \in \mathcal{L}'_T = MT(\mathcal{L}_S)$. Using the functional behaviour of the corresponding backward model transformation according to Def. 3.1 for the language $\mathcal{L}'_T$ we know that for each model H^T the backward model transformation yields a unique $H^S \in VL_S$. Therefore, each backward model transformation sequence $(G^T, G'_0 \xRightarrow{tr_B^*} G'_n, G'^S)$ leads to a unique $G'^S \in VL_S$. Furthermore, there is a backward model transformation sequence $(G^T, G''_0 \xRightarrow{tr_B^*} G''_n, G^S)$ by Thm. 4 implying $G^S \cong G'^S$, i.e. the model transformation is completely information preserving. □

Example 3.19 (Complete Information Preservation). The model transformation MT_1 = *CD2RDBM* is not completely information preserving. Consider e.g. the source model G^S in Fig. 14 of Ex. 3.17, where two backward model transformation sequences are possible starting with the same derived target model G^T. This means that the backward model transformation has no functional behaviour with respect to $MT_1(\mathcal{L}_S) = MT(VL_S) = VL_T = \mathcal{L}_T$.

However, we can also consider the inverse model transformation, i.e. swapping the forward and backward direction leading to the model transformation $MT_2 = \mathit{RDBM2CD}$ from relational data base models to class diagrams. In this case, the model transformation is completely information preserving meaning that each relational data base model M_{DB} can be transformed into a class diagram M_{CD}, and each data based model model M_{DB} can be completely and uniquely reconstructed from its derived class diagram M_{CD}. In other words, each class diagram resulting from a model transformation sequence of *RDBM2CD* contains all information that were present in the given data base model. According to Ex. 3.15 we know that the model transformation *CD2RDBM* has functional behaviour and hence, the backward model transformation of *RDBM2CD* has functional behaviour with respect to VL_T being equal to the source language VL_S of *CD2RDBM*. For this reason, we can apply Thm. 5 and have that *RDBM2CD* is completely information preserving. In particular, foreign keys are completely represented by associations, and primary keys by primary attributes. There is no structure within the data base model which is not explicitly represented within the class diagram.

4. Related Work

TGGs have been successfully applied for model transformations with different purposes in a variety of domains (Guerra and de Lara2006a; Guerra and de Lara2006b; Kindler and Wagner2007; Königs and Schürr2006; Taentzer et al.2005). The formal construction and analysis of model transformations based on TGGs has been started in (Ehrig et al.2007) by analysing information preservation of bidirectional model transformations and continued in (Ehrig et al.2008; Ehrig and Prange2008; Ehrig et al.2009a; Ehrig et al.2009b; Hermann et al.2010c), where model transformations based on TGGs are compared with those on plain graph grammars in (Ehrig et al.2008), TGGs with specification NACs are analysed in (Ehrig et al.2009b) and an efficient on-the-fly construction is introduced in (Ehrig et al.2009a). Pattern-based model-to-model transformations have been introduced in (de Lara and Guerra2008) and corresponding correctness, completeness and termination results have been presented in (Orejas et al.2009), which are, however, limited in comparison with the results in this article.

A first approach on analysing functional behaviour was presented for restricted TGGs with distinguished kernels in (Ehrig and Prange2008) and a more general approach based on forward translation rules in (Hermann et al.2010a; Hermann et al.2010c). The concept of forward translation rules is inspired by the translation algorithm in (Schürr and Klar2008), which uses a set for storing the elements that have been translated during a transformation. The results in this paper for model transformations based on forward translation rules with specification and filter NACs are based on results in most of these papers. In particular, we extended these formal results by providing a less restrictive condition for functional behaviour and a sufficient condition for complete information preservation.

In (Ehrig et al.2007), a similar case study based on forward rules is presented, but without using NACs. The grammar with NACs in this paper handles primary keys and foreign keys in a more appropriate way and allows us to show strong functional behaviour.

A more restrictive condition for ensuring functional behaviour is presented in (Giese et al.2010), which requires the complete absence of all critical pairs, while the presented condition in this article only requires strict confluence of the significant critical pairs after optimizing the rules by the automatic and interactive generation of filter NACs. In order to reduce backtracking, Klar et. al. (Klar et al.2010) propose a concept similar to the automatic generation of filter NACs in Sec. 3.1. The effect of the filter NACs is specified directly within the transformation algorithm, however, complete elimination of backtracking cannot be ensured.

There are several other approaches to model transformations and in particular bidirectional ones, where the general idea is to define one direction of the model transformation and get the backward direction for free. This is different to the TGG case, where we define the triple rules that build up the language of consistently integrated models and from these triple rules we derive both forward and backward rules. In (Hidaka et al.2010), a bidirectional language is defined using structural recursion on graphs. In (Bohannon et al.2006), lenses are introduced, which are basically a pair of functions - get for forward transformation and put for backward transformation - obeying certain behavioral laws. Foster uses these lenses to propose a bidirectional language for model transformations for updating views (Foster2009), which ensures that changes are propagated back to the underlying model. Stevens discusses different important properties for model transformations in (Stevens2008). Among specification, composition, and maintenance of model transformations, also verification and correctness properties are advised and some corresponding laws for lenses are formulated. With their main focus on updates, lenses seem to fit especially to views, but their usefulness for general model transformations with very different source and target models and the application to graphs and other high-level structures requires further analysis.

5. Conclusion

5.1. *Summary of Main Results*

In this paper we have studied model transformations based on triple graph grammars (TGGs) with negative application conditions (NACs) in order to improve analysis and execution compared with previous approaches in the literature.

The first key idea is that model transformations can be constructed by applying forward translation rules with NACs, which can be derived automatically from the given TGG-rules with NACs. The first main result shows correctness and completeness of model transformations for forward transformations and also for forward translations by combining Thm. 1 and Fact 2.20. The second main result provides a sufficient condition for functional behaviour (Thm. 2) based on the analysis of critical pairs for forward translation rules with filter NACs. The generation of filter NACs improves the analysis of functional behaviour for model transformations based on critical pair analysis (using the tool AGG (AGG2011)) by filtering out backtracking paths and this way, some critical pairs. If we are able to construct filter NACs such that the corresponding rules have no more "significant" critical pairs, then the third main result shows that we have strong functional behaviour (Thm. 3). Moreover, Thms. 2 and 3 also show that strict confluence of significant critical pairs ensures that backtracking is not required for the execution of the model transformation which implies polynomial time complexity. Finally, we show by Thm. 4 that TGG-model transformations are information preserving and by Thm. 5 that forward transformations are completely information preserving, if the corresponding backward transformation has functional behaviour. For our case study *CD2RDBM* we show that backtracking can be eliminated and strong functional behaviour can be obtained by an automatic optimization based on filter NACs. Moreover, this leads to complete information preservation for the derived backward transformation.

The main challenge in applying our main results on (strong) functional behaviour and complete information preservation is to find suitable filter NACs, such that we have a minimal number of critical pairs. For this purpose, we provide automated and interactive techniques for the generation of filter NACs (see Facts 3.7 and 3.8).

5.2. *Practical Relevance*

In the following we discuss how the results in this paper can be used to meet the "Grand Research Challenge of the TGG Community" formulated by Schürr et.al. in (Schürr and Klar2008). The main aims are "Consistency", "Completeness", "Expressiveness" and "Efficiency" of model transformations.

1 *Consistency:* Model transformations are consistent towards the given TGG, if whenever the algorithm translates a source model G_S into a target model G_T then there is a triple graph $G = (G_S \leftarrow G_C \rightarrow G_T) \in VL$ generated by the TGG. This property is shown in Thm. 1.

2 *Completeness and Termination:* Completeness means that the execution of the model transformation translates each source model $G_S \in VL_S$. This property subsumes termination. Both properties are ensured for our construction by Thm. 1 and Fact 3.11 if triple rules are creating on the source part.

3 *Efficiency:* Model transformations shall have polynomial space and time complexity with exponent k the maximal number of elements of a rule. This property can be ensured, if we can show that a model transformation does not require backtracking and the TGG has a finite set of triple rules, which are creating on the source component, have finitely many NACs and whose rule components are finite. In this case, each execution of a model transformation has at most n steps with n being the amount of structural elements of the source model. As discussed in (Schürr and Klar2008), the bound k then ensures polynomial time complexity. Moreover, we provided sufficient criteria and techniques for reducing and eliminating backtracking in Sec. 3.1, where they are used for the the analysis of functional behaviour. Large TGGs with more than 50 rules and big input models may still slow down the execution. However, in a current project, we are using a TGG with 50 triple rules in the tool Henshin (Arendt et al.2010) where we experience that the execution time for transforming models with several hundred model elements is below 2 seconds on a standard PC. Moreover, the tool AGG (AGG2011) provides automated analysis components, which we used for

analysing functional behaviour and information preservation of the case study in this paper as described in Sec. 3.

4 *Expressiveness:* Finally, features that are urgently needed for solving practical problems like NACs and attribute conditions shall be captured. Both, NACs and attributes are handled by our approach. Moreover, we partially extended the results to the case of more general application conditions in (Golas et al.2011) in the sense of (Habel and Pennemann2009).

Summing up, the presented approach to model transformations based on triple graph grammars provides an intuitive, expressive, formally well-founded and efficient framework for bidirectional model transformations including powerful results for analysis and optimization via filter NACs. According to the listed achievements above there are several important advantages in comparison to other existing approaches, like (Schürr and Klar2008; Königs and Schürr2006; Kindler and Wagner2007; Giese and Wagner2009; Giese and Hildebrandt2009), which are mainly software engineering focused, and therefore, do not offer similar formal results. However, these approaches are very similar and stimulated the development of some constructions. For this reason, the presented results can be potentially transferred to the related approaches with some modification efforts.

5.3. *Future Work*

While we considered functional behaviour with respect to unique target models in this paper, the more general notion in (Schürr and Klar2008) regarding some semantic equivalence of target models will be part of further extensions of our techniques. Moreover, we will study further static conditions for eliminating misleading execution paths and we will develop extensions to layered model transformations and amalgamated rules. Finally, we already applied some of the presented results for model transformation to the model synchronization based on TGGs in order to ensure correctness (Hermann et al.2011b). But there are several further problems in model synchronization which will require new results, e.g. concerning a notion of information preservation for partially related domain languages.

References

AGG (2011). *AGG.* TFS-Group, TU Berlin. http://tfs.cs.tu-berlin.de/agg.

Arendt, T., Biermann, E., Jurack, S., Krause, C., and Taentzer, G. (2010). Henshin: Advanced concepts and tools for in-place EMF model transformations. In *Proc. of the ACM/IEEE 13th Intern. Conf. on Model Driven Engineering Languages and Systems (MoDELS'10)*, volume 6394 of *LNCS*, pages 121–135.

Bisztray, D., Heckel, R., and Ehrig, H. (2009). Verification of architectural refactorings: Rule extraction and tool support. *Electronic Communications of the EASST*, 16.

Bohannon, A., Vaughan, J. A., and Pierce, B. C. (2006). Relational Lenses: A Language for Updateable Views. In *Principles of Database Systems (PODS).*

de Lara, J. and Guerra, E. (2008). Pattern-Based Model-to-Model Transformation. In Ehrig, H., Heckel, R., Rozenberg, G., and Taentzer, G., editors, *Proc. 4th Int. Conf. on Graph Transformations (ICGT 2008)*, volume 5214 of *LNCS*, pages 426–441. Springer.

Ehrig, H., Ehrig, K., Ermel, C., Hermann, F., and Taentzer, G. (2007). Information Preserving Bidirectional Model Transformations. In *Proc. Fundamental Approaches to Software Engineering (FASE'07)*, volume 4422 of *LNCS*, pages 72–86. Springer.

Ehrig, H., Ehrig, K., Prange, U., and Taentzer, G. (2006). *Fundamentals of Algebraic Graph Transformation.* EATCS Monographs in Theor. Comp. Science. Springer.

Ehrig, H., Ermel, C., and Hermann, F. (2008). On the Relationship of Model Transformations Based on Triple and Plain Graph Grammars. In *Proc. Graph and Model Transformation (GraMoT'08)*, pages 9–16. ACM.

Ehrig, H., Ermel, C., Hermann, F., and Prange, U. (2009a). On-the-Fly Construction, Correctness and Completeness of Model Transformations based on Triple Graph Grammars. In *Proc. Model Driven Engineering Languages and Systems (MODELS'09)*, volume 5795 of *LNCS*, pages 241–255. Springer.

Ehrig, H., Golas, U., and Hermann, F. (2010). Categorical Frameworks for Graph Transformation and HLR Systems based on the DPO Approach. *Bulletin of the EATCS*, 102:111–121.

Ehrig, H., Hermann, F., and Sartorius, C. (2009b). Completeness and Correctness of Model Transformations based on Triple Graph Grammars with Negative Application Conditions. *ECEASST*, 18.

Ehrig, H., Pfender, M., and Schneider, H. (1973). Graph grammars: an algebraic approach. In *14th Annual IEEE Symposium on Switching and Automata Theory*, pages 167–180. IEEE.

Ehrig, H. and Prange, U. (2008). Formal Analysis of Model Transformations Based on Triple Graph Rules with Kernels. In *Proc. Intern. Conf. on Graph Transformation (ICGT'08)*, volume 5214 of *LNCS*, pages 178–193. Springer.

Foster, J. (2009). *Bidirectional Programming Languages.* Dissertation, University of Pennsylvania.

Giese, H. and Hildebrandt, S. (2009). Efficient Model Synchronization of Large-Scale Models . Technical Report 28, Hasso Plattner Institute at the University of Potsdam.

Giese, H., Hildebrandt, S., and Lambers, L. (2010). Toward bridging the gap between formal semantics and implementation of triple graph grammars. Technical Report 37, Hasso Plattner Institute at the University of Potsdam.

Giese, H. and Wagner, R. (2009). From model transformation to incremental bidirectional model synchronization. *Software and Systems Modeling*, 8(1):21–43.

Golas, U., Ehrig, H., and Hermann, F. (2011). Formal Specification of Model Transformations by Triple Graph Grammars with Application Conditions. *ECEASST.* To appear.

Guerra, E. and de Lara, J. (2006a). Attributed typed triple graph transformation with inheritance in the double pushout approach. Technical Report UC3M-TR-CS-2006-00, Universidad Carlos III, Madrid, Spain.

Guerra, E. and de Lara, J. (2006b). Model View Management with Triple Graph Grammars. In *Proc. Intern. Conf. on Graph Transformation (ICGT'06)*, volume 4178 of *LNCS*, pages 351–366. Springer.

Habel, A. and Pennemann, K.-H. (2009). Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, 19(2):245–296.

Hermann, F., Corradini, A., and Ehrig, H. (2011a). Analysis of Permutation Equivalence in M-adhesive Transformation Systems with Negative Application Conditions. *MSCS.* submitted, online available at http://tfs.cs.tu-berlin.de/publikationen/Papers11/HCE11.pdf.

Hermann, F., Ehrig, H., Golas, U., and Orejas, F. (2010a). Efficient Analysis and Execution of Correct and Complete Model Transformations Based on Triple Graph Grammars. In *Proc. Model Driven Interoperability (MDI'10)*, MDI '10, pages 22–31. ACM.

Hermann, F., Ehrig, H., Golas, U., and Orejas, F. (2010b). Efficient Analysis and Execution of Correct and Complete Model Transformations Based on Triple Graph Grammars - Extended Version. Technical Report 2010/13, FAk. IV, TU Berlin.

Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., and Xiong, Y. (2011b). Correctness of Model Synchronization Based on Triple Graph Grammars. In *Proc. of the ACM/IEEE 13th Intern. Conf. on Model Driven Engineering Languages and Systems (MoDELS'11)*, LNCS. Springer Verlag. to appear.

Hermann, F., Ehrig, H., Orejas, F., and Golas, U. (2010c). Formal Analysis of Functional Behaviour of Model Transformations Based on Triple Graph Grammars. In *Proc. Intern. Conf. on Graph Transformation (ICGT' 10)*, volume 6372 of *LNCS*, pages 155–170. Springer.

Hermann, F., Hülsbusch, M., and König, B. (2010d). Specification and verification of model transformations. *ECEASST*, 30:1–21.

Hidaka, S., Hu, Z., Inaba, K., Kato, H., Matsuda, K., and Nakano, K. (2010). Bidirectionalizing graph transformations. In *Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, ICFP '10, pages 205–216. ACM.

Kindler, E. and Wagner, R. (2007). Triple Graph Grammars: Concepts, Extensions, Implementations, and Application Scenarios. Technical Report TR-ri-07-284, Department of Computer Science, University of Paderborn, Germany.

Klar, F., Lauder, M., Königs, A., and Schürr, A. (2010). Extended Triple Graph Grammars with Efficient and Compatible Graph Translators. In *Graph Transformations and Model Driven Enginering - Essays Dedicated to Manfred Nagl on the Occasion of his 65th Birthday*, volume 5765 of *LNCS*, pages 144–177. Springer.

Königs, A. and Schürr, A. (2006). Tool Integration with Triple Graph Grammars - A Survey. In *Proc. SegraVis School on Foundations of Visual Modelling Techniques*, volume 148, pages 113–150. Electronic Notes in Theoretical Computer Science, Elsevier Science.

Lack, S. and Sobociński, P. (2005). Adhesive and quasiadhesive categories. *Theoretical Informatics and Applications*, 39(2):511–546.

Lambers, L. (2009). *Certifying Rule-Based Models using Graph Transformation.* PhD thesis, Technische Universität Berlin.

Newman, M. H. A. (1942). On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243.

Orejas, F., Guerra, E., de Lara, J., and Ehrig, H. (2009). Correctness, Completeness and Termination of Pattern-Based Model-to-Model Transformation. In Kurz, A., Lenisa, M., and Tarlecki, A., editors, *Int. Conf. on Algebra and Coalgebra in Computer Science (CALCO'09)*, volume 5728 of *LNCS*, pages 383–397. Springer.

Plump, D. (1993). Hypergraph Rewriting: Critical Pairs and Undecidability of Confluence. In *Term Graph Rewriting: Theory and Practice*, pages 201–213. John Wiley.

Plump, D. (2005). Confluence of Graph Transformation Revisited. In *Processes, Terms and Cycles: Steps on the Road to Infinity*, volume 3838 of *LNCS*, pages 280–308. Springer.

Rozenberg, G. (1997). *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations.* World Scientific.

Schürr, A. (1994). Specification of Graph Translators with Triple Graph Grammars. In *Proc. Workshop on Graph-Theoretic Concepts in Computer Science (WG'94)*, volume 903 of *LNCS*, pages 151–163. Springer Verlag.

Schürr, A. and Klar, F. (2008). 15 years of triple graph grammars. In *Proc. Int. Conf. on Graph Transformation (ICGT 2008)*, pages 411–425.

Stevens, P. (2008). A Landscape of Bidirectional Model Transformations. In *Proceedings of GTTSE 2008*, volume 5235 of *LNCS*, pages 408–424. Springer.

Taentzer, G., Ehrig, K., Guerra, E., de Lara, J., Lengyel, L., Levendovsky, T., Prange, U., Varro, D., and Varro-Gyapay, S. (2005). Model Transformation by Graph Transformation: A Comparative Study. In *Proc. Workshop Model Transformation in Practice.*

Appendix A. Category of Typed Attributed Garaphs

Typed attributed triple graphs are based on underlying category of of typed attributed graphs ($\mathbf{AGraphs}_{ATG}, \mathcal{M}$) which is given by the slice category ($\mathbf{AGraph}{\downarrow} ATG, \mathcal{M}$) of directed attributed graphs over a type graph ATG. In this App. A, we review the main constructions for the category of typed attributed graphs according to (Ehrig et al.2006).

An attributed graph consists of an extended directed graph for the structural part – called E-graph – together with an algebra for the specification of the carrier sets of the value nodes. An E-graph extends a directed graph by additional attribute value nodes and edges for the attribution of structural nodes and edges.

Definition A.1 (E-graph and E-graph morphism). An *E-graph* G with $G = (V_G, V_D, E_G, E_{NA}, E_{EA}, (source_j, target_j)_{j\in\{G,NA,EA\}})$ consists of the sets

— V_G and V_D , called the graph and data nodes (or vertices), respectively;
— E_G, E_{NA} , and E_{EA} called the graph, node attribute, and edge attribute edges, respectively; and the source and target functions
— $source_G : E_G \to V_G, target_G : E_G \to V_G$ for graph edges;
— $source_{NA} : E_{NA} \to V_G, target_{NA} : E_{NA} \to V_D$ for node attribute edges; and
— $source_{EA} : E_{EA} \to E_G, target_{EA} : E_{EA} \to V_D$ for edge attribute edges:

$$\begin{tikzcd}
 & E_G \arrow[rr, shift left, "source_G"] \arrow[rr, shift right, "target_G"'] & & V_G & \\
E_{EA} \arrow[ur, "source_{EA}"] \arrow[rr, "target_{EA}"'] & & V_D & & E_{NA} \arrow[ul, "source_{NA}"'] \arrow[ll, "target_{NA}"]
\end{tikzcd}$$

Consider the E-graphs G^1 and G^2 with $G^k = (V_G^k, V_D^k, E_G^k, E_{NA}^k, E_{EA}^k, (source_j^k, target_j^k)_{j\in\{G,NA,EA\}})$ for $k = 1, 2$. An E-graph morphism $f : G^1 \to G^2$ is a tuple $(f_{V_G}, f_{V_D}, f_{E_G}, f_{E_{NA}}, f_{E_{EA}})$ with $f_{V_i} : V_i^1 \to V_i^2$ and $f_{E_j} : E_j^1 \to E_j^2$ for $i \in \{G, D\}$, $j \in \{G, NA, EA\}$ such that f commutes with all source and target functions, for example $f_{V_G} \circ source_G^1 = source_G^2 \circ f_{E_G}$.

The carrier sets of attribute values that form the single set V_D of an E-graph are defined by an additional data algebra D, which also specifies the operations for generating and manipulating data values. The carrier sets D_s of D contain the data elements for each sort $s \in S$ according to a data signature $DSIG = (S_D, OP_D)$. These carrier sets are combined by disjoint union and form the set V_D of data elements.

Definition A.2 (Attributed Graph and Attributed Graph Morphism). Let $DSIG = (S_D, OP_D)$ be a data signature with attribute value sorts $S'_D \subseteq S_D$. An attributed graph $AG = (G, D)$ consists of an E-graph G together with a $DSIG$-algebra D such that $\uplus_{s \in S'_D} D_S = V_D$. For two attributed graphs $AG^1 = (G^1, D^1)$ and $AG^2 = (G^2, D^2)$, an attributed graph morphism $f : AG^1 \to AG^2$ is a pair $f = (f_G, f_D)$ with an E-graph morphism $f_G : G^1 \to G^2$ and an algebra homomorphism $f_D : D^1 \to D^2$ such that (1) commutes for all $s \in S'_D$, where the vertical arrows are inclusions.

$$\begin{tikzcd}
D^1_s \arrow[r, "f_{D,s}"] \arrow[d, hook] \arrow[dr, phantom, "(1)"] & D^2_s \arrow[d, hook] \\
V^1_D \arrow[r, "f_{G,V_D}"] & V^2_D
\end{tikzcd}$$

The category of typed attributed graphs $\mathbf{AGraphs}_{ATG}$ has as objects all attributed graphs with a *typing morphism* to the attributed graph ATG (type graph), and as arrows all attributed graph morphisms preserving the typing. The category $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ is shown in (Ehrig et al.2006) to be an adhesive HLR category, where the distinguished class of monomorphisms $\mathcal{M}$ contains all monomorphisms that are isomorphisms on the data part. For this reason, all results for adhesive HLR transformation systems presented in (Ehrig et al.2006) are valid. Since $\mathcal{M}$-adhesive categories (Ehrig et al.2010) are a slight generalisation of weak adhesive and adhesive HLR categories the category $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ is an $\mathcal{M}$-adhesive category.

Appendix B. Remaining Proofs of Technical Results

In this section we provide proofs for Facts 2.20, 3.7, 3.8 and 3.9. In order to prove Fact 2.20 above we use Def. B.1 and Lem. B.2 below concerning the equivalence of single transformation steps using the on-the-fly construction of model transformations based on forward rules presented in (Ehrig et al.2009a). In this context, forward sequences are constructed with an on-the-fly check for partial source consistency. Partial source consistency requires that the constructed forward sequence $G_0 \xRightarrow{tr^*_F} G_k$ is partially match consistent, meaning that for each intermediate forward step $G_{k-1} \xRightarrow{tr_{k,F}} G_k$ the compatibility with the corresponding source step $G_{k-1,0} \xRightarrow{tr_{k,S}} G_{k,0}$ of the simultaneously created source sequence $G_{00} \xRightarrow{tr^*_S} G_{k,0}$ is checked. Compatibility requires that the forward match $m_{k,F}$ is forward consistent, which means that the comatch $n_{k,S}$ of the source step and the match $m_{k,F}$ of the forward step coincide on the source component with respect to the inclusion $G_{k-1,0} \hookrightarrow G_0 \hookrightarrow G_{k-1}$. The formal condition of a forward consistent match is given in Def. B.1 by a pullback diagram where both matches satisfy the corresponding NACs, and intuitively, it specifies that the effective elements of the forward rule are matched for the first time in the forward sequence.

Definition B.1 (Forward Consistent Match). Given a partially match consistent sequence $\varnothing = G_{00} \xRightarrow{tr^*_S} G_{n-1,0} \xhookrightarrow{g_n} G_0 \xRightarrow{tr^*_F} G_{n-1}$ then a match $m_{n,F} : L_{n,F} \to G_{n-1}$ for $tr_{n,F} : L_{n,F} \to R_{n,F}$ is called *forward consistent* if there is a source match $m_{n,S}$ such that diagram (1) is a pullback and the matches $m_{n,F}$ and $m_{n,S}$ satisfy the corresponding target and source *NACs*, respectively.

$$\begin{tikzcd}
L_{n,S} \arrow[r, hook] \arrow[d, "m_{n,S}"'] & R_{n,S} \arrow[r, hook] \arrow[d, phantom, "(1)"] & L_{n,F} \arrow[d, "m_{n,F}"] \\
G_{n-1,0} \arrow[r, hook, "g_{n-1}"'] & G_0 \arrow[r, hook] & G_{n-1}
\end{tikzcd}$$

Lemma B.2 (Forward translation step). Let TR be a set of triple rules with $tr_i \in TR$ and let TR_F be the derived set of forward rules. Given a partially match consistent forward sequence $\varnothing = G_{00} \xRightarrow{tr_S^*} G_{i-1,0} \xhookrightarrow{g_{i-1}} G_0 \xRightarrow{tr_F^*} G_{i-1}$ and a corresponding forward translation sequence $G'_0 \xRightarrow{tr_{FT}^*} G'_{i-1}$, both with almost injective matches, such that $G'_{i-1} = G_{i-1} \oplus Att^{\mathbf{F}}_{G_0 \setminus G_{i-1,0}} \oplus Att^{\mathbf{T}}_{G_{i-1,0}}$. Then the following are equivalent:

1 There is a TGT-step $G_{i-1} \xRightarrow{tr_{i,F}, m_{i,F}} G_i$ with forward consistent match $m_{i,F}$
2 There is a forward translation TGT-step $G'_{i-1} \xRightarrow{tr_{i,FT}, m_{i,FT}} G'_i$

and we have $G'_i = G_i \oplus Att^{\mathbf{F}}_{G_0 \setminus G_{i,0}} \oplus Att^{\mathbf{T}}_{G_{i,0}}$.

The proof of Lem. B.2 is given by the proof of Fact 1 in (Hermann et al.2010b).

Fact 2.20 (Equivalence of Forward Transformations and Forward Translation Sequences (see Sec. 2.2)**).**

Proof. We first show the equivalence of the sequences disregarding the NACs. Item 1 is equivalent to the existence of the sequence $G_0 \xRightarrow{tr_{1,F}, m_{1,F}} G_1 \xRightarrow{tr_{2,F}, m_{2,F}} G_2 \ldots \xRightarrow{tr_{n,F}, m_{n,F}} G_n$ with $G^S_n = G^S$, where each match is forward consistent according to Def. B.1. Item 2 is equivalent to the existence of the complete forward translation sequence $G'_0 \xRightarrow{tr_{1,FT}, m_{1,FT}} G'_1 \xRightarrow{tr_{2,FT}, m_{2,FT}} G'_2 \ldots \xRightarrow{tr_{n,FT}, m_{n,FT}} G'_n$ via TR_{FT}.

Disregarding the NACs, it remains to show that $G'^S_0 = Att^{\mathbf{F}}(G^S)$ and $G'^S_n = Att^{\mathbf{T}}(G^S)$. We apply Lemma B.2 for $i = 0$ with $G_{0,0} = \varnothing$ up to $i = n$ with $G_{n,0} = G_0$ and using $G^S_0 = G^S$ we derive:
$G'^S_0 = G^S_0 \oplus Att^{\mathbf{T}}_{G_{0,0}} \oplus Att^{\mathbf{F}}_{G^S_0 \setminus G^S_{0,0}} = G^S_0 \oplus Att^{\mathbf{F}}_{G^S_0} = G^S \oplus Att^{\mathbf{F}}_{G^S} = Att^{\mathbf{F}}(G^S)$.
$G'^S_n = G^S_n \oplus Att^{\mathbf{T}}_{G^S_{n,0}} \oplus Att^{\mathbf{F}}_{G^S_0 \setminus G^S_{n,0}} = G^S_n \oplus Att^{\mathbf{T}}_{G^S_{n,0}} = G^S \oplus Att^{\mathbf{T}}_{G^S} = Att^{\mathbf{T}}(G^S)$.

Now, we show that the single steps are also NAC consistent. For each step, we have transformations $G_{i-1,0} \xRightarrow{tr_{i,S}, m_{i,S}} G_{i,0}$, $G_{i-1} \xRightarrow{tr_{i,F}, m_{i,F}} G_i$, $G'_{i-1} \xRightarrow{tr_{i,FT}, m_{i,FT}} G'_i$ with $G'_{i-1} = G_{i-1} \oplus Att^{\mathbf{F}}_{G_0 \setminus G_{i-1,0}} \oplus Att^{\mathbf{T}}_{G_{i-1,0}}$, $G'_i = G_i \oplus Att^{\mathbf{F}}_{G_0 \setminus G_{i,0}} \oplus Att^{\mathbf{T}}_{G_{i,0}}$, and $m_{i,FT}|_{L_{i,F}} = m_{i,F}$.

For a target NAC $n : L_i \to N$, we have to show that $m_{i,F} \models n$ iff $m_{i,FT} \models n_{FT}$, where n_{FT} is the corresponding forward translation NAC of n. If $m_{i,FT} \not\models n_{FT}$, we find a monomorphism q' with $q' \circ n_{FT} = m_{i,FT}$. Since $n = n_{FT}|_N$, define $q = q'|_N$ and it follows that $q \circ n = m_{i,F}$, i.e. $m_{i,F} \not\models n$. Vice versa, if $m_{i,F} \not\models n$, we find a monomorphism q with $q \circ n = m_{i,F}$. Since $N^S = L^S_i$, we do not have any additional translation attributes in N_{FT}. Thus $m_{i,FT}$ can be extended by q to $q' : N_{FT} \to G'_{i-1}$ such that $m_{i,FT} \not\models n_{FT}$.

Similarly, we have to show that for a source NAC $n : L \to N$, $m_{i,S} \models n$ iff $m_{i,FT} \models n_{FT}$. As for target NACs, if $m_{i,FT} \not\models n_{FT}$, we find a monomorphism q' with $q' \circ n_{FT} = m_{i,FT}$ and for the restriction to L^S_i and N^S it follows that $q^S \circ n^S = m^S_{i,FT}$, i.e. $m_{i,S} \not\models n$. Vice versa, if $m_{i,S} \not\models n$, we find a monomorphism q with $q \circ n = m_{i,S}$. Now define q' with $q'(x) = m_{i,FT}(x)$ for $x \in L_{FT}$, $q'(x) = q(x)$ for $x \in N \setminus L_i$, and for each $x \in N^S \setminus L^S_i$ we have that $q(x) \in G_{i-1,0}$. From the above characterization of G'_{i-1} it follows that the corresponding translation attributes $\mathtt{tr\_}x$ and $\mathtt{tr\_}x\mathtt{\_}a$ are set to $\mathbf{T}$ in G'_{i-1}. Thus, q' is well-defined and $q' \circ n_{FT} = m_{i,FT}$, i.e. $m_{i,FT} \not\models n_{FT}$.

The equality of the model transformation relations follows by the equality of the pairs (G^S, G^T) in the model transformation sequences in both cases. $\square$

Fact 3.7 (Automated Generation of Filter NACs (see Sec. 3.1)**).**

Proof. Consider a generated NAC $(n : L_{FT} \to N)$ for a node x in tr with an outgoing edge e in $N \setminus L$. A transformation step $N \xRightarrow{tr_{FT},n} M$ exists, because the gluing condition is always satisfied for forward translation rules as explained in Sec. 2.2, and the edge e in M is still labelled with a translation attribute set to "**F**", but x is labelled with "**T**", because it is matched by the rule. Now, consider a graph $H' \supseteq M$, such that H' is a graph with translation attributes over a graph without translation attributes H, i.e. $H' = H \oplus Att_{H_0}$ for $H_0 \subseteq H'$ meaning that H' has at most one translation attributes for each element in H without translation attributes.

We show that H' is not translatable, which implies that M is misleading (Def. 3.3). Forward translation rules only modify translation attributes from "**F**" to "**T**", they do not increase the amount of translation attributes of a graph and no structural element is deleted. Thus, each graph H_i in a TGT sequence $H' \xRightarrow{tr^*_{FT}} \overline{H}_n$ will contain the edge e labelled with "**F**", because the rules, which modify the translation attribute of e are not applicable due to x being labelled with "**T**" in each graph $\overline{H}_i$ in the sequence and there is only one translation attribute for x in H'. Thus, each $\overline{H}_n$ is not completely translated and therefore, M is misleading. This means that $(n : L_{FT} \to N)$ is a filter NAC of tr_{FT}. By duality, the result also holds for a generated NAC with respect to an incoming edge. □

Fact 3.8 (Interactive Generation of Filter NACs (see Sec. 3.1)**).**

Proof. The constructed NACs are filter NACs, because the transformation step $K \xRightarrow{tr_{1,FT},m_1} P_1$ contains the misleading graph P_1. The procedure terminates, because the number of critical pairs is bounded by the amount of possible pairwise overlappings of the left hand sides of the rules. The amount of overlappings can be bounded by considering only constants and variables as possible attribute values. □

Fact 3.9 (Equivalence of Transformations with Filter NACs (see Sec. 3.1)**).**

Proof. Sequence 1 consists of the same transformation diagrams as Sequence 2. NAC-consistency of sequence 2 implies NAC-consistency of sequence 1, because each step in Sequence 2 involves a superset of the NACs for the corresponding step in Sequence 1. For the inverse direction, consider a step $G_{i-1} \xRightarrow{tr_{(i,FT)},\mathrm{m}_{(i,FT)}} G_i$, which leads to the step $G_{i-1} \xRightarrow{tr_{(i,FN)},\mathrm{m}_{(i,FT)}} G_i$ if NACs are not considered. Assume that m_{FT} does not satisfy some NAC of tr_{FN}. This implies that a filter NAC $(n : L_{i,FT} \to N)$ is not fulfilled, because all other NACs are fulfilled by NAC-consistency of Sequence 1. Thus, there is a triple morphism $q : N \to G_{i-1}$ with $q \circ n = m_{i,FT}$. By Thm. 6.18 (Restriction Thm.) in (Ehrig et al.2006) we have that the transformation step $G_{i-1} \xRightarrow{tr_{(i,FN)},\mathrm{m}_{(i,FT)}} G_i$ can be restricted to $N \xRightarrow{tr_{(i,FT)},n} H$ with embedding $H \to G_i$. By Def. 3.5 of filter NACs we know that $N \xRightarrow{tr_{(i,FT)},n} H$ and H is misleading, which implies by Def. 3.3 that G_i is not translatable. This is a contradiction to the completely translated graph G_n in sequence 1 and therefore, the filter NAC is fulfilled leading to NAC-consistency of sequence 2. □