Efficient Analysis and Execution of Correct and Complete Model Transformations Based on Triple Graph Grammars

Frank Hermann
Department of Theoretical Computer Science and Software Technology
Technische Universität Berlin
Berlin, Germany
frank(at)cs.tu-berlin.de

Hartmut Ehrig
Department of Theoretical Computer Science and Software Technology
Technische Universität Berlin
Berlin, Germany
ehrig(at)cs.tu-berlin.de

Ulrike Golas
Department of Theoretical Computer Science and Software Technology
Technische Universität Berlin
Berlin, Germany
ugolas(at)cs.tu-berlin.de

Fernando Orejas
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Barcelona, Spain
orejas(at)lsi.upc.edu

ABSTRACT

Triple Graph Grammars are a well-established, formal and intuitive concept for the specification and analysis of bidirectional model transformations. In previous work we have formalized and analyzed already termination, correctness, completeness, local confluence and functional behaviour.

In this paper, we show how to improve the efficiency of the execution and analysis of model transformations in practical applications by using triple rules with negative application conditions (NACs). In addition to *specification NACs*, which improve the specification of model transformations, the generation of *filter NACs* improves the efficiency of the execution and the analysis of functional behaviour supported by critical pair analysis of the tool AGG. We illustrate the results for the well-known model transformation from class diagrams to relational database models.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications;
D.2.12 [**Software Engineering**]: Interoperability;
I.6.5 [**Simulation and Modeling**]: Model Development - Modeling methodologies

General Terms

Theory, Design, Verification

MDI 2010, October 5, 2010, Oslo, Norway.

Keywords

Model Transformation, Triple Graph Grammars, Functional Behaviour

1. INTRODUCTION

Model transformations based on triple graph grammars (TGGs) have been introduced by Schürr in [19]. Operational rules are automatically derived from the triple rules and used to define various bidirectional model transformation and integration tasks that are mainly focused on model-to-model transformations Since 1994, several extensions of the original TGG definitions have been published [20, 17, 10], and various kinds of applications have been presented [22, 11, 16]. Besides model transformation TGGs are also applied for model integration [1] and model synchronization [8] in order to support model driven interoperability.

For source-to-target model transformations, so-called *forward* transformations, forward rules are derived which take the source graph as input and produce a corresponding target graph. Similarly, backward rules are used for target-to-source transformations making the transformation approach bidirectional. Major properties expected to be fulfilled for model transformations are termination, correctness, completeness, efficient execution and — for several applications — functional behaviour. Termination, completeness and correctness of model transformations have been studied already in [6, 3, 7, 4]. Functional behaviour of model transformations based on triple graph grammars has been analyzed for triple rules without application conditions in [15] using forward translation rules that use additional translation attributes for keeping track of the elements that have been translated so far.

The main aim of this paper is to extend the analysis techniques for functional behaviour in [15] to the case of triple rules with negative application conditions (NACs) and to improve the efficiency of analysis and execution of

model transformations studied in [3, 4, 7, 15]. For this purpose, we distinguish between specification NACs and filter NACs. Specification NACs have been introduced already in [7, 4], where triple rules and corresponding derived source and forward rules have been extended by NACs in order to improve the modeling power. Exemplarily, we show that NACs improve the specification of the model transformation *CD2RDBM* from class diagrams to relational data base models presented in [6, 3]. Therefore, we extend the forward translation rules introduced in [15] by corresponding NACs and show that model transformations based on forward translation rules with NACs are equivalent to model transformations studied in [7, 4], such that main results concerning termination, correctness and completeness can be transferred to our new framework (see Thm. 1). In order to analyze functional behaviour we can use general results for local confluence of transformation systems with NACs in [18]. But in order to improve efficiency in the context of model transformations we introduce so-called filter NACs. They filter out several misleading branches considered in the standard analysis of local confluence using critical pairs. In our second main result (see Thm. 2) we show how to analyze functional behaviour of model transformations based on forward translation rules by analyzing critical pairs for forward translation rules with filter NACs. Moreover, we introduce a strong version of functional behaviour, including model transformation sequences. In our third main result (see Thm. 3) we characterize strong functional behaviour by the absence of "significant" critical pairs for the corresponding set of forward translation rules with filter NACs.

In Sec. 2 we introduce model transformations based on TGGs with specification NACs and show the first main result on termination, correctness, and completeness. In Sec. 3 we introduce forward translation rules with filter NACs and present our main results on functional and strong functional behaviour. Based on these main results we discuss in Sec. 4 efficiency aspects of analysis and execution. Related work and a conclusion are presented in Sections 5 and 6. The full proofs of the main results are given in [14].

2. MODEL TRANSFORMATIONS BASED ON TRIPLE GRAPH GRAMMARS WITH NACS

Triple graph grammars [19] are a well-known approach for bidirectional model transformations. Models are defined as pairs of source and target graphs, which are connected via a correspondence graph together with its embeddings into these graphs. In this section, we review main constructions and results of model transformations based on [20, 4, 15] and extend them to the case with NACs.

A triple graph $G = (G_S \xleftarrow{s_G} G_C \xrightarrow{t_G} G_T)$ consists of three graphs G_S, G_C, and G_T, called source, correspondence, and target graphs, together with two graph morphisms $s_G : G_C \to G_S$ and $t_G : G_C \to G_T$. A triple graph morphism $m = (m_S, m_C, m_T) : G \to H$ between triple graphs G and H consists of three graph morphisms $m_S : G_S \to H_S$, $m_C : G_C \to H_C$ and $m_T : G_T \to H_T$ such that $m_S \circ s_G = s_H \circ m_C$ and $m_T \circ t_G = t_H \circ m_C$. A typed triple graph G is typed over a triple graph TG by a triple graph morphism $type_G : G \to TG$.

Example 1. **Triple Type Graph:** Fig. 1 shows the type graph TG of the triple graph grammar TGG for our exam-

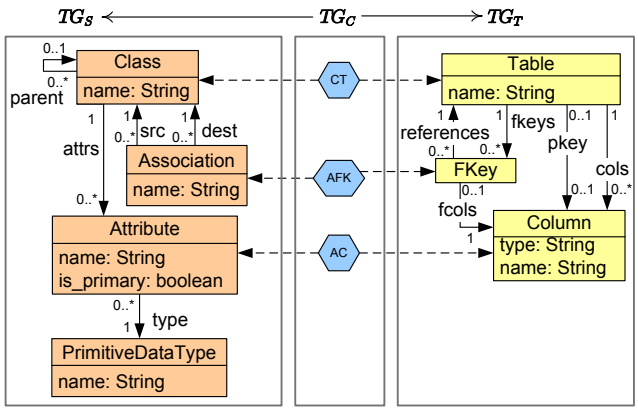

Figure 1: Triple type graph for *CD2RDBM*

ple model transformation from class diagrams to database models. The source component TG_S defines the structure of class diagrams while in the 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 [6], elements are arranged left, center, and right according to the component types source, correspondence and target. Morphisms starting at a correspondence part are specified by dashed arrows. The denoted multiplicity constraints are ensured by the triple rules in Figs. 3 and 5.

Note that the case study uses attributed triple graphs based on E-graphs as presented in [6] in the framework of weak adhesive HLR categories. We refer to [2] for more details on attributed graphs.

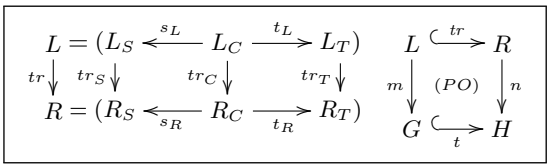

Figure 2: Triple rule and triple transformation step

Triple rules synchronously build up their source, target and correspondence graphs, i.e. they are non-deleting. A triple rule tr (left of Fig. 2) is an injective triple graph morphism $tr = (tr_S, tr_C, tr_T) : L \to R$ and w.l.o.g. we assume tr to be an inclusion. Given a triple graph morphism $m : L \to G$, a triple graph transformation (TGT) step $G \overset{tr,m}{\Longrightarrow} H$ (right of Fig. 2) from G to a triple graph H is given by a pushout of triple graphs with comatch $n : R \to H$ and transformation inclusion $t : G \hookrightarrow H$. A grammar $TGG = (TG, S, TR)$ consists of a triple type graph TG, a triple start graph $S = \varnothing$ and a set TR of triple rules.

Example 2. **Triple Rules:** The triple rules in Fig. 3 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 labeled with green "++" and marked by green line colouring. The rule "*Class2Table*" synchronously creates a class with name "n" together with the corresponding table in the

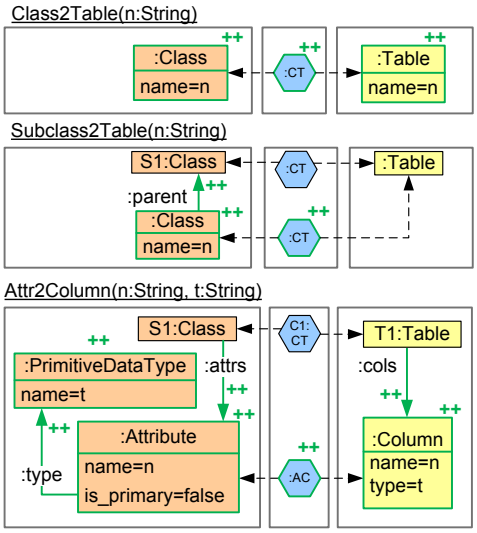

Figure 3: Rules for the model transformation *CD2RDBM*, Part 1

$$\begin{array}{ccc} (L_S \nleftarrow \varnothing \nrightarrow \varnothing) & (\varnothing \nleftarrow \varnothing \nrightarrow L_T) & (R_S \xleftarrow{tr_S \circ s_L} L_C \xrightarrow{t_L} L_T) \\ \downarrow tr_S \quad \downarrow \quad \downarrow & \downarrow \quad \downarrow \quad tr_T \downarrow & \downarrow id \quad tr_C \downarrow \quad tr_T \downarrow \\ (R_S \nleftarrow \varnothing \nrightarrow \varnothing) & (\varnothing \nleftarrow \varnothing \nrightarrow R_T) & (R_S \xleftarrow{s_R} R_C \xrightarrow{t_R} R_T) \\ \text{source rule } tr_S & \text{target rule } tr_T & \text{forward rule } tr_F \end{array}$$

Figure 4: Derived operational rules of a TGG

relational database. Accordingly, subclasses are connected to the tables of its super classes by rule "*Subclass2Table*". Attributes with type "t" are created together with their corresponding columns in the database component via the rule "*Attr2Column*".

From each triple rule tr we derive a source rule tr_S for the construction resp. parsing of a model of the source language and a forward rule tr_F for forward transformation sequences (see Fig. 4). By TR_S and TR_F we denote the sets of all source and forward rules derived from the set of triple rules 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.

A set of triple rules TR and the start graph $\varnothing$ generate a visual language VL of integrated models, i.e. models with elements in the source, target and correspondence component. The source language VL_S and target language VL_T are derived by projection to the triple components, i.e. $VL_S = proj_S(VL)$ and $VL_T = proj_T(VL)$. The set VL_{S0} of models that can be generated resp. parsed by the set of all source rules TR_S is possibly larger than VL_S and we have $VL_S \subseteq VL_{S0} = \{G_S \mid \varnothing \Rightarrow^* (G_S \leftarrow \varnothing \rightarrow \varnothing)$ *via* $TR_S\}$. Analogously, we have $VL_T \subseteq VL_{T0} = \{G_T \mid \varnothing \Rightarrow^* (\varnothing \leftarrow \varnothing \rightarrow G_T)$ *via* $TR_T\}$.

According to [7, 4] we present negative application conditions for triple rules. In 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. They prohibit the existence of certain structures either in the source or in the target part only, while general NACs may prohibit both at once.

Definition 1. **Triple Rules with Negative Application Conditions:** Given a triple rule $tr = (L \rightarrow R)$, a negative application condition (NAC) $(n : L \rightarrow N)$ 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 $m : L \rightarrow G$ is NAC consistent if there is no injective $q : N \rightarrow G$ 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.

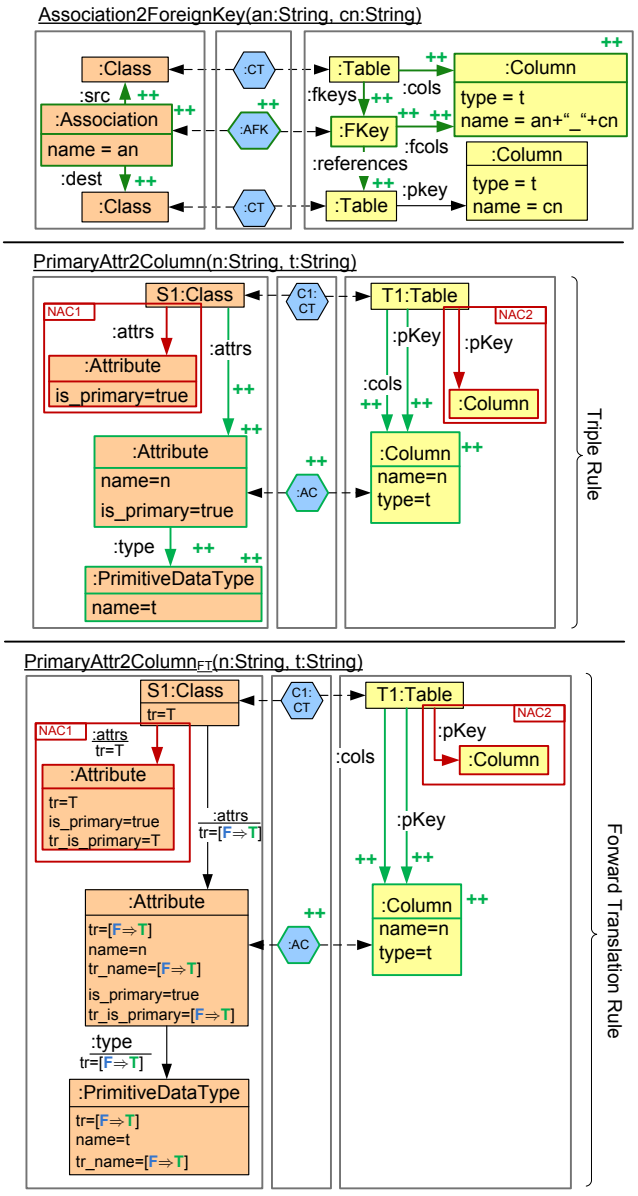

Figure 5: Rules for the model transformation *CD2RDBM*, Part 2

Example 3. **Triple Rules with NACs:** Figure 5 shows the remaining two triple rules for the model transformation "*CD2RDBM*" and additionally a derived forward translation rule as explained in Ex. 4. NACs are specified in short notation using the label "NAC" with a frame and red line colour

within the frame. A complete NAC is obtained by composing the left hand side of a rule with the red marked elements within the NAC-frame. The rule "*Association2ForeignKey*" creates an association between two classes and the corresponding foreign key and the NAC ensures that there is only one primary key at the destination table. The parameters "an" and "cn" are used to set the names of the association and column nodes. The rule "*PrimaryAttr2Column*" extends "*Attr2Column*" by creating additionally a link of type "*pkey*" for the column and by setting "is_primary=true". Furthermore, there is a source and a target NAC, which ensure that there is no primary attribute nor column currently present.

The extension of forward rules to forward translation rules is based on additional attributes, called translation attributes, that control the translation process by keeping track of the elements which have been translated so far. While in this paper the translation attributes are inserted in the source models they can be kept separate as an external pointer structure in order to keep the source model unchanged as shown in Sec. 5 of [13].

Definition 2. **Graph with Translation Attributes:** Given an attributed graph $AG = (G, D)$ and a subgraph $G_0 \subseteq G$ we call AG' a *graph with translation attributes over* AG if it extends AG with one boolean-valued attribute tr_x for each element x (node or edge) in G_0 and one boolean-valued attribute tr_x_a for each attribute associated to such an element x in G_0. This means that we have a partition of the items (nodes, edges, or attributes) of G_0 into I_1 and I_2 s.t. $AG' = AG \oplus Att^{\mathbf{T}}_{I_1} \oplus Att^{\mathbf{F}}_{I_2}$, where $Att^{\mathbf{T}}_{I_1}$ and $Att^{\mathbf{F}}_{I_2}$ denotes the translation attributes with value $\mathbf{T}$ for I_1 and value $\mathbf{F}$ for I_2. Moreover, we define $Att^v(AG) := AG \oplus Att^v_G$ for $v \in \{\mathbf{T}, \mathbf{F}\}$. In any case we require that there is at most one translation attribute tr_x or tr_x_a for each item.

The new concept of forward translation rules as introduced in [15] extends the construction of forward rules by additional translation attributes in the source component. The translation attributes keep track of the elements that have been translated so far, which ensures that each element in the source graph is not translated twice. The rules are deleting on the translation attributes and thus, the triple transformations are extended from a single (total) pushout to the classical double pushout (DPO) approach [2]. We call these rules forward translation rules, because pure forward rules need to be controlled by additional control conditions, such as the source consistency condition in [6, 4].

Definition 3. **Forward Translation Rules with NACs:** 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 1. Note that $(L_{FT} +_L N)$ is the union of L_{FT} and N with shared L and for a target NAC n the forward translation NAC n_{FT} does not contain any translation attributes because $N_S = L_S$.

Example 4. **Forward Translation Rule with NACs:** Fig 5 shows in its lower part the forward translation rule with NACs "*PrimaryAttr2Column*$_{FT}$". According to Def. 3 the source elements of the triple rule "*PrimaryAttr2Column*" 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 additional elements in the NAC are extended by translation attributes set to "**T**". Thus, the source NACs concern only elements that have been translated so far.

From the application point of view model transformation rules should be applied along matches that are injective on the structural part. 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 are mapped to the same data value. For this reason we use the notion of "*almost injective matches*" [15], 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. Next, we define model transformations based on forward translation rules based on complete forward translation sequences.

Definition 4. **Completely Translated Graphs and Complete Sequences:** A forward translation sequence $G_0 \xRightarrow{tr^*_{FT}} G_n$ with almost injective matches is called *complete* if G_n is *completely translated*, i.e. all translation attributes of G_n are set to true (" **T**").

Definition 5. **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 with NACs consists of a source graph G_S, a target graph G_T, and a complete TGT-sequence $G_0 \xRightarrow{tr^*_{FT}} G_n$ with almost injective matches, $G_0 = (Att^{\mathbf{F}}(G_S) \leftarrow \varnothing \to \varnothing)$ and $G_n = (Att^{\mathbf{T}}(G_S) \leftarrow G_C \to G_T)$.
A *model transformation* $MT : VL_{S0} \Rrightarrow VL_{T0}$ based on forward translation rules with NACs is defined by all model transformation sequences as above with $G_S \in VL_{S0}$ and $G_T \in VL_{T0}$. All these pairs (G_S, G_T) define the *model transformation relation* $MTR \subseteq VL_{S0} \times VL_{T0}$. The model transformation is *terminating* if there are no infinite TGT-sequences via forward translation rules and almost injective matches starting with $G_0 = (Att^{\mathbf{F}}(G_S) \leftarrow \varnothing \to \varnothing)$ for some source graph G_S.

Now, we are able to state our first main result concerning termination, correctness and completeness of model transformations.

THEOREM 1. **Termination, Correctness and Completeness:** *Each model transformation* $MT : VL_{S0} \Rrightarrow VL_{T0}$ *based on forward translation rules is*

- terminating, *if each forward translation rule changes at least one translation attribute from "**F**" to "**T**",*
- correct, *i.e. for each model transformation sequence* $(G_S,\ G_0 \overset{tr^*_{FT}}{\Longrightarrow} G_n, G_T)$ *there is* $G \in VL$ *with* $G = (G_S \leftarrow G_C \rightarrow G_T)$, *and it is*
- complete, *i.e. 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* $(G_S, G_0 \overset{tr^*_{FT}}{\Longrightarrow} G_n, G_T)$.

Proof Idea. The proof (see [14]) is based on a corresponding result in [15] for the case without NACs and a Fact showing the equivalence of (1) source and NAC-consistent TGT-sequences based on forward rules and (2) complete NAC-consistent TGT-sequences based on forward translation rules. □

Applying a rule according to the DPO approach involves the check of the gluing condition in general. However, in the case of forward translation rules and almost injective matches we have that the gluing condition is always satisfied. This means that the condition does not have to be checked, which simplifies the analysis of functional behaviour in Sec. 3.

Fact 1. **Gluing Condition for Forward Translation Rules:** Let tr_{FT} be a forward translation rule and $m_{FT} : L_{FT} \rightarrow G$ be an almost injective match, then the gluing condition is satisfied, i.e. there is the transformation step $G \overset{tr_{FT}, m_{FT}}{\Longrightarrow} H$.

Proof Idea. Since only attribution edges are deleted there are no dangling points and almost injective matching ensures that there are no identification points (see [14] for full proof). □

3. ANALYSIS OF FUNCTIONAL BEHAVIOUR

Functional behaviour of a model transformation means that each model of the source language $\mathcal{L}_S \subseteq VL_S$ is transformed into a unique model of the target language. This section presents new techniques especially developed to show functional behaviour of correct and complete model transformations based on TGGs.

Definition 6. **Functional Behaviour of Model Transformations:** A model transformation MT based on forward translation rules has *functional behaviour* if each execution of MT starting at a source model G_S of the source language $\mathcal{L}_S \subseteq VL_S$ leads to a unique target model $G_T \in VL_T$. The execution of MT requires backtracking, if there are terminating TGT-sequences $(Att^F(G_S) \leftarrow \varnothing \rightarrow \varnothing) \overset{tr^*_{FT}}{\Longrightarrow} G'_n$ with $G'^S_n \neq Att^T(G_S)$.

The standard way to analyze 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. In the context of model transformations, confluence only needs to be ensured for transformation paths which lead to completely translated models. For this reason, we introduce so-called *filter NACs* that extend the model transformation rules in order to avoid misleading paths that cause backtracking. The overall behaviour

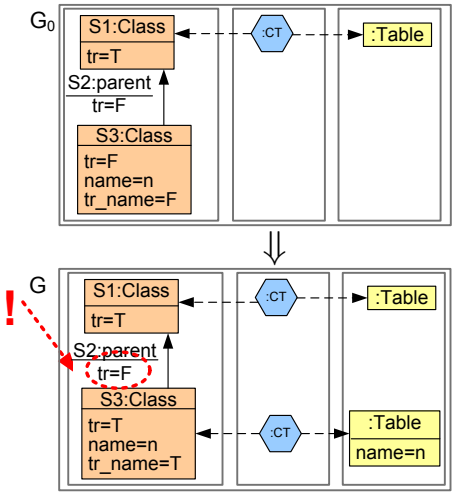

Figure 6: Step $G_0 \overset{Class2Table_{FT}}{\Longrightarrow} G$ **with misleading graph** G

w.r.t. the model transformation relation is preserved. 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 7. **Translatable and Misleading Graphs:** A triple graph with translation attributes G is *translatable* if there is a transformation $G \overset{*}{\Rightarrow} H$ such that H is completely translated. 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 5. **Misleading Graph:** Consider the transformation step shown in Fig. 6. The resulting graph G is misleading according to Def. 7, because the edge $S2$ is labeled with a translation attribute set to "**F**", but there is no rule which may change this attribute in any larger 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 labeled 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 8. **Filter NAC:** A filter NAC n for a forward translation rule $tr_{FT} : L_{FT} \rightarrow R_{FT}$ is given by a morphism $n : L_{FT} \rightarrow N$, such that there is a TGT step $N \overset{tr_{FT}, n}{\Longrightarrow} M$ 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.

Example 6. **Forward Translation Rule with Filter NACs:** The rule in Fig. 7 extends the rule $Class2Table_{FT}$ by a filter NAC obtained from graph G_0 of the transformation step $G_0 \overset{Class2Table_{FT}}{\Longrightarrow} G$ in Fig. 6, where G is misleading according to Ex. 5. In Ex. 7 we extend the rule by a further similar filter NAC with "tr = **T**" for node "$S2$".

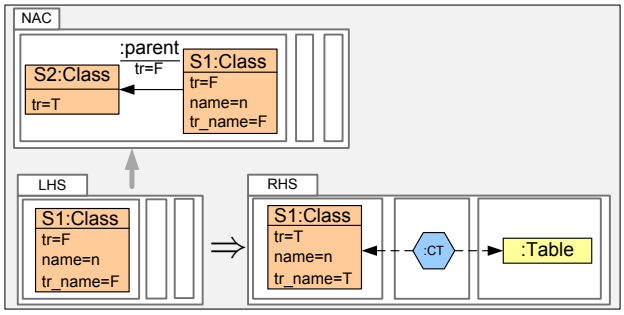

Figure 7: A forward translation rule with filter NAC: *Class2Table*$_{FN}$

A direct construction of filter NACs according to Def. 8 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 we can bound the size without losing generality. At first we present a static technique for a subset of filter NACs and thereafter, a dynamic generation technique leading to a much larger set of filter NACs. The first procedure in Fact 2 below is based on a sufficient criteria for checking the misleading property. Concerning our example this static generation leads to the filter NAC shown in Fig. 7 for the rule *Class2Table*$_{FT}$ for an incoming edge of type "*parent*".

Fact 2. **Static 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 the following conditions
  - 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 Idea. Each generated NAC $(n : L_{FT} \to N)$ for a node x in tr with an outgoing (incoming) for an edge e in $N \setminus L$ defines a transformation step $N \overset{tr_{FT},n}{\Longrightarrow} M$, where edge e is still labeled with "**F**", but x is labeled with "**T**".By the structure of forward translation rules it follows that edge e cannot be labeled with "**T**" at any later model transformation step for any given source model G_S. The full proof is given in [14]. □

The following dynamic technique for deriving relevant 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 [2]) we know that for each pair of two parallel dependent transformation steps there is a critical pair which can be embedded. For this reason, the generation of critical pairs can be used to derive filter NACs. A critical pair either directly specifies a filter NAC or a conflict that may lead to non-functional behaviour of the model transformation.

For the dynamic generation of filter NACs we use the tool AGG [23] for the generation of critical pairs for a plain graph transformation system. For this purpose, we first perform the flattening construction for triple graph grammars presented in [3, 15] extended to NACs using the flattening construction for morphisms. A critical pair $P_1 \overset{tr_{1,FT}}{\Longleftarrow} K \overset{tr_{2,FT}}{\Longrightarrow} P_2$ consists of a pair of parallel dependent transformation steps. If a critical pair contains a misleading graph P_1 we can use the overlapping graph K as a filter NAC of the rule $tr_{1,FT}$. However, checking the misleading property needs human assistance, such that the generated critical pairs can be seen as filter NAC candidates. But 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. **Dynamic Generation of Filter NACs:** Given a set of forward translation rules, then generate the set of critical pairs $P_1 \overset{tr_{1,FT},m_1}{\Longleftarrow} K \overset{tr_{2,FT},m_2}{\Longrightarrow} P_2$. 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 $K \overset{tr_{1,FN}}{\Longrightarrow} P_1$ violates the filter NAC. Hence, the critical pair for $tr_{1,FT}$ and $tr_{2,FT}$ is no longer a critical pair for 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. The constructed NACs are filter NACs, because the transformation step $K \overset{tr_{1,FT},m_1}{\Longrightarrow} P_1$ contains the misleading graph P_1. The procedure terminates, because the critical pairs are 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. □

For our case study the dynamic generation terminates already after the second round, which is typical for practical applications, because the amount of already translated elements in the new critical pairs usually decreases. Furthermore, the amount of NACs can be reduced by combining similar NACs differing only on some translation attributes. The remaining critical pairs that do not specify filter NACs show effective conflicts between transformation rules and they can be provided to the developer of the model transformation to support the design phase.

The filter NACs introduced in this paper on the one hand support the analysis of functional behaviour and on the

other hand, they also improve the efficiency of the execution. By definition, the occurrence of a filter NAC at an intermediate model means that the application of the owning rule would lead to a model that cannot be translated completely, i.e. the execution of the model transformation would perform backtracking at a later step. This way, a filter NAC *cuts off possible backtracking paths* of the model transformation. As presented in Fact 2 some filter NACs can be generated automatically and using Fact 3 a larger set of them can be obtained based on the generation of critical pairs. Finally, by Thms. 2 and 3 we can completely avoid backtracking if TR_{FN} has no significant critical pair or, alternatively, if all critical pairs are strictly confluent.

As shown by Fact 4 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. This equivalence is used on the one hand for the analysis of functional behaviour in Thms. 2 and 3 and furthermore, for improving the efficiency of the execution of model transformations as explained in Sec. 4.

Fact 4. **Equivalence of Transformations with Filter NACs:** Given a triple graph grammar $TGG = (TG, \varnothing, TR)$ and a triple graph $G_0 = (G_S \leftarrow \varnothing \rightarrow \varnothing)$ typed over TG. Let $G_0' = (Att^{\mathbf{F}}(G_S) \leftarrow \varnothing \rightarrow \varnothing)$. Then, the following are equivalent for almost injective matches:

1. $\exists$ a complete TGT-sequence $G_0' \overset{tr_{FT}^*, m_{FT}^*}{\Longrightarrow} G'$ via forward translation rules.
2. $\exists$ a complete TGT-sequence $G_0' \overset{tr_{FN}^*, m_{FT}^*}{\Longrightarrow} G'$ via forward translation rules with filter NACs.

Proof Idea. Sequence 1 consists of the same derivation diagrams as Sequence 2. The additional filter NACs in sequence 2 prevent a transformation rule to create a misleading graph. Both sequences lead to completely translated models, such that we know that the matches in sequence 1 also fulfill the filter NACs of the rules in sequence 2. The full proof is given in [14]. □

Theorem 2. **Functional Behaviour:** *Let MT be a model transformation based on forward translation rules TR_{FT} and let TR_{FN} extend TR_{FT} with filter NACs such that TR_{FN} is terminating and all critical pairs are strictly confluent. Then, MT has functional behaviour. Moreover, the model transformation MT' based on TR_{FN} does not require backtracking and defines the same model transformation relation, i.e. $MTR' = MTR$.*

Remark 2. TR_{FN} is terminating, if TR_{FT} is terminating and a sufficient condition is given in Thm. 1. Termination of TR_{FN} with strict confluence of critical pairs implies unique normal forms by the Local Confluence Theorem in [18].

Proof Idea. The proof (see [14]) is based on a decomposition theorem of triple rule sequences into match-consistent TGT-sequences based on source and forward rules with NACs in [7]. The latter are equivalent to complete TGT-sequences based on forward translation rules without NACs in [15] and with NACs in Fact 1 in [14]. Finally, by Fact 4 complete TGT-sequences via forward translation rules with and without filter NACs are equivalent. □

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 with forward translation rules TR_{FT} has functional behaviour. From an efficiency point of view, model transformations should be based on a compact set of rules, because large rule sets usually involve more attempts of matching until finding a valid match. In the optimal case, the rule set ensures that each transformation sequence of the model transformation is itself unique up to switch equivalence. For this reason, we introduce the notion of strong functional behaviour.

Definition 9. **Strong Functional Behaviour of Model Transformations:** A model transformation based on forward translation rules TR_{FN} with filter NACs has *strong functional behaviour* if for each $G_S \in \mathcal{L}_S \subseteq VL_S$ there is a $G_T \in VL_T$ and a model transformation sequence $(G_S, G_0 \overset{tr_{FN}^*}{\Longrightarrow} G_n, G_T)$ and each two terminating TGT-sequences $G_0' \overset{tr_{FN}^*}{\Longrightarrow} G_n'$ and $G_0' \overset{\overline{tr}_{FN}^*}{\Longrightarrow} \overline{G}_m'$ are switch-equivalent up to isomorphism.

Remark 3.

1. The sequences are terminating means that no rule in TR_{FN} is applicable any more, but 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 $G_0' \overset{tr_{FN}^*}{\Longrightarrow} G_n'$ and $G_0' \overset{\overline{tr}_{FN}^*}{\Longrightarrow} \overline{G}_m'$ 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 [18]. 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 $t1' = i \circ t1$, 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.

Definition 10. **Significant Critical Pair:** A critical pair $P_1 \overset{tr_{1,FN}}{\Longleftarrow} K \overset{tr_{2,FN}}{\Longrightarrow} P_2$ for TR_{FN} is called significant, if it can be embedded into a parallel dependent pair $G_1' \overset{tr_{1,FN}}{\Longleftarrow} G' \overset{tr_{2,FN}}{\Longrightarrow} G_2'$ such that there is $G_S \in VL_S$ and $G_0' \overset{tr_{FN}^*}{\Longrightarrow} G'$ with $G_0' = (Att^{\mathbf{F}}(G_S) \leftarrow \varnothing \rightarrow \varnothing)$.

$$G_0' \overset{*}{\Longrightarrow} G' \overset{tr_{1,FN}}{\rightrightarrows} \begin{matrix} G1' \\ G_2' \end{matrix} \quad (tr_{2,FN})$$

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 iff TR_{FN} has no significant critical pair.*

PROOF IDEA. The proof (see [14]) is based on that of Thm. 2 and the fact that in the absence of critical pairs two terminating sequences with the same source can be shown to be switch-equivalent up to isomorphism using the Local Church-Rosser and Parallelism Thm. with NACs in [18]. □

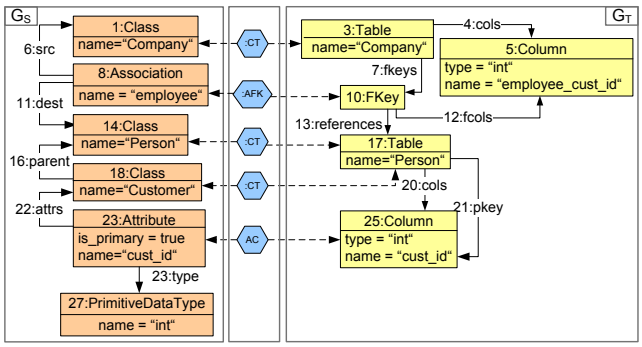

Figure 8: Triple graph instance

Example 7. **Functional Behaviour:** We analyze functional behaviour of the model transformation *CD2RDBM* with triple rules TR given in Figs. 3 and 5. First of all, *CD2RDBM* is terminating according to Thm. 1. For analyzing the local confluence we can use the tool AGG [23] for the generation of critical pairs. We use the extended rule $Class2Table_{FN}$ as shown in Fig. 7 and extend it by a further filter NAC obtained by the static generation acc. to Fact 2. AGG detects two critical pairs showing a conflict of the rule "*PrimaryAttr2Column*" with itself for an overlapping graph with two primary attributes. Both critical pairs lead to additional filter NACs by the dynamic generation of filter NACs in Fact 3 leading to a system of forward translation rules with filter NACs without 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* and does not require backtracking. As an example, Fig. 8 shows the resulting triple graph (translation attributes are omitted) of a model transformation starting with the class diagram G_S.

4. EFFICIENT ANALYSIS AND EXECUTION

Our approach to model transformations based on triple graph grammars (TGGs) with NACs will be discussed now with respect to the efficiency for both, analysis of properties and execution.

Correctness and Completeness: As shown by Thm. 1 based on [7, 4] model transformations based on TGGs with NACs are correct and complete with respect to the language of integrated models VL generated by the triple rules. Thus, correctness and completeness are ensured by construction.

Termination: As presented in [4] termination is essentially ensured, if all triple rules are creating on the source component. This property can be checked statically, automatically and efficiently by checking $(R_S \setminus L_S) \neq \emptyset$. In Thm. 1 we have given an explicit condition for the forward translation rules to be terminating.

Functional Behaviour: The new concept of filter NACs introduced in this paper provides a powerful basis for reducing the analysis efforts w.r.t. functional behaviour. Once termination is shown as explained above, functional behaviour of model transformations based on forward translation rules TR_{FT} can be checked by generating the critical pairs of the transformation system with AGG [23] and showing strict confluence. The static and dynamic generation of filter NACs (Facts 2 and 3) allows to eliminate critical pairs. In the best case, all critical pairs disappear showing the functional behaviour of the model transformation immediately. The new notion of strong functional behaviour of a system based on transformation rules TR_{FN} with filter NACs is completely characterized by the absence of "significant" critical pairs, such that we can ensure for each source model that the transformation sequence is unique up to switch equivalence. Furthermore, the critical pairs generated by AGG can be used to find the conflicts between the rules which may cause non-functional behaviour of the model transformation. The modeler can decide whether to change the rules or to keep the non-functional behaviour.

Model Size	Model Transformation Sequences of CD2RDBM				
	without Filter NACs		with Filter NACs		
	Time[1)]	Success Rate	Time[1)]	Overhead	Success Rate
[Elements[2)]]	[ms]	[%]	[ms]	[%]	[%]
11	143.75	42.86	158.33	10.14	100.00
25	302.75	16.84	335.45	10.80	100.00
53	672.68	3.94	742.62	10.40	100.00
109	1,481.43	0.17	1,584.86	6.98	100.00

1) Average time of 100 successful model transformation sequences
2) Nodes and Edges

Table 1: Benchmark, Tool: AGG [23]

Efficient Execution: Filter NACs do not only improve the analysis of functional behaviour of a TGG, but also the execution of the model transformation process by forbidding the application of misleading transformation steps that would lead to a dead-end eliminating the need of backtracking for these cases. Table 1 shows execution times using the transformation engine AGG [23]. The additional overhead caused by filter NACs is fairly small and lies in the area of 10% for the examples in the benchmark, which is based on the average execution times for 100 executions concerning models with 11, 25, 53 and 109 elements (nodes and edges), respectively. The first model with 11 elements is the presented class diagram in the source component of Fig. 8. We explicitly do not compare the execution times of the system with filter NACs with one particular system with backtracking, because these times can vary heavily depending on the used techniques for partial order reduction and the chosen examples. Instead we present the computed success rates for the system without NACs which show that backtracking will cause a substantial overhead in any case. Thus, the listed times concern successful execution paths only, i.e. those executions that lead to a completely translated model. The success rate for transformations without filter NACs decreases fast when considering larger models. Times for the

unsuccessful executions, which appear in the system without filter NACs, are not considered. However, in order to ensure completeness there is the need for backtracking for the system without filter NACs. This backtracking overhead is in general exponential and in our case study misleading graphs appear already at the beginning of many transformation sequences implying that backtracking is costly. Backtracking is reduced by filter NACs and avoided completely in the case that no "significant critical pair" remains present (see Thm. 3), which we have shown to be fulfilled for our example. The additional overhead of about 10% for filter NACs is in most cases much smaller than the efforts for backtracking.

Moreover, in order to perform model transformations using highly optimized transformation machines for plain graph transformation, such as Fujaba and GrGen.Net [21], we have presented how the transformation rules and models can be equivalently represented by plain graphs and rules. First of all, triple graphs and morphisms are flattened according to the construction presented in [3, 15], which can be extended to NACs using the flattening of morphisms. Furthermore, we presented in this paper how forward rules with NACs are extended to forward translation rules with NACs, such that the control condition "source consistency" [6] and also the gluing condition (Fact 1) are ensured automatically for complete sequences, i.e. they do not need to be checked during the transformation.

Summing up, the presented results allow us to combine the easy, intuitive and formally well founded specification of model transformations based on triple graph grammars with NACs with the best available tools for executing graph transformations while still ensuring correctness and completeness.

5. RELATED WORK

Since 1994, several extensions of the original TGG definitions [19] have been published [20, 17, 10] and various kinds of applications have been presented [22, 11, 16]. The formal construction and analysis of model transformations based on TGGs has been started in [6] by analyzing information preservation of bidirectional model transformations and continued in [3, 5, 4, 7, 15], where model transformations based on TGGs are compared with those on plain graph grammars in [3], TGGs with specification NACs are analyzed in [7] and an efficient on-the-fly construction is introduced in [4]. A first approach analyzing functional behaviour was presented for restricted TGGs with distinguished kernels in [5] and a more general approach, however without NACs, based on forward translation rules in [15]. The results in this paper for model transformations based on forward translation rules with specification and filter NACs are based on the results of all these papers except of [5].

In [6] a similar case study based on forward rules is presented, but without using NACs. This causes that more TGT-sequences are possible, in particular, an association can be transformed into a foreign key with one primary key, even if there is a second primary attribute that will be transformed into a second primary key at a later stage. This behaviour is not desired from the application point of view. Thus, the grammar with NACs in this paper handles primary keys and foreign keys in a more appropriate way. Furthermore, the system has strong functional behaviour as shown in Sec. 3.

In the following we discuss how the presented results can be used to meet the "Grand Research Challenge of the TGG Community" formulated by Schürr et.al. in [20]. The main aims are "Consistency", "Completeness", "Expressiveness" and "Efficiency" of model transformations. The first two effectively require correctness, completeness w.r.t. the triple language VL and additionally termination and functional behaviour. They are ensured as shown in Sec. 3. While we considered functional behaviour w.r.t. unique target models, the more general notion in [20] regarding some semantical equivalence of target models will be part of further extensions of our techniques. "Expressiveness" requires suitable control mechanisms like NACs, which are used extensively in this paper and we further extend the technique by additional control mechanisms. In [9] more general application conditions [12] are considered, but functional behaviour is not yet analyzed. In general, the overall usage of complex control structures should be kept low, because they may cause complex computations. Finally, we discussed in Sec. 4 that our approach can be executed efficiently based on efficient graph transformation engines. Especially model transformations fulfilling the conditions in Thm. 3 do not need to backtrack, which bounds the number of transformation steps to the elements in the source model as required in [20].

6. CONCLUSION

In this paper we have studied model transformations based on triple graph grammars (TGGs) with negative application conditions (NACs) in order to improve efficiency of 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 termination under weak assumptions, correctness and completeness of model transformations in this framework, which is equivalent to the approach in [7]. The second key idea is to introduce filter NACs in addition to the NACs in the given TGG-rules, which in contrast are called specification NACs in this paper. Filter NACs are useful to improve the analysis of functional behaviour for model transformations based on critical pair analysis (using the tool AGG [23]) by filtering out backtracking paths and this way, some critical pairs. The second main result provides a sufficient condition for functional behaviour based on the analysis of critical pairs for forward translation rules with filter NACs. 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, i.e. not only the results are unique up to isomorphism but also the corresponding model transformation sequences are switch-equivalent up to isomorphism. Surprisingly, we can show that the condition "no significant critical pairs" is not only sufficient, but also necessary for strong functional behaviour. Finally, we discuss efficiency aspects of analysis and execution of model transformations and show that our sample model transformation *CD2RDBM* based on TGG-rules with NACs has strong functional behaviour.

The main challenge in applying our main results on functional and strong functional behaviour is to find suitable filter NACs, such that we have a minimal number of critical pairs for the forward translation rules with filter NACs. For this purpose, we provide static and dynamic techniques

for the generation of filter NACs (see Facts 2 and 3). The dynamic technique includes a check that certain models are misleading. In any case, the designer of the model transformation can specify some filter NACs directly by himself, if he can ensure the filter NAC property. Furthermore, we can avoid backtracking completely by Thms. 2 and 3 if TR_{FN} has no significant critical pair or, alternatively, if all critical pairs are strictly confluent.

In future work, we will study further static conditions to check whether a model is "misleading", because this allows to filter out misleading execution paths. In addition to that, we currently develop extensions to layered model transformations and amalgamated rules, which allow to further reduce backtracking in general cases and to simplify the underlying rule sets. Moreover, we study applications to model transformations that partially relate two DSLs, were some node types are irrelevant for the model transformation.

7. REFERENCES

[1] Ehrig, H., Ehrig, K., Hermann, F.: From Model Transformation to Model Integration based on the Algebraic Approach to Triple Graph Grammars. In: Ermel, C., de Lara, J., Heckel, R. (eds.) Proc. GT-VMT'08. EC-EASST, vol. 10. EASST (2008)

[2] Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs, Springer (2006)

[3] Ehrig, H., Ermel, C., Hermann, F.: On the Relationship of Model Transformations Based on Triple and Plain Graph Grammars. In: Karsai, G., Taentzer, G. (eds.) Proc. GraMoT'08. ACM (2008)

[4] Ehrig, H., Ermel, C., Hermann, F., Prange, U.: On-the-Fly Construction, Correctness and Completeness of Model Transformations based on Triple Graph Grammars. In: Schürr, A., Selic, B. (eds.) Proc. ACM/IEEE MODELS'09. LNCS, vol. 5795, pp. 241–255. Springer (2009)

[5] Ehrig, H., Prange, U.: Formal Analysis of Model Transformations Based on Triple Graph Rules with Kernels. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) Proc. ICGT'08. LNCS, vol. 5214, pp. 178–193. Springer (2008)

[6] Ehrig, H., Ehrig, K., Ermel, C., Hermann, F., Taentzer, G.: Information preserving bidirectional model transformations. In: Dwyer, M.B., Lopes, A. (eds.) Proc. FASE'07. LNCS, vol. 4422, pp. 72–86. Springer (2007)

[7] Ehrig, H., Hermann, F., Sartorius, C.: Completeness and Correctness of Model Transformations based on Triple Graph Grammars with Negative Application Conditions. In: Heckel, R., Boronat, A. (eds.) Proc. GT-VMT'09. EC-EASST, vol. 18. EASST (2009)

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

[9] Golas, U., Ehrig, H., Hermann, F.: Enhancing the Expressiveness of Formal Specifications for Model Transformations by Triple Graph Grammars with Application Conditions. In: Proc. Int. Workshop on Graph Computation Models (GCM'10) (2010)

[10] Guerra, E., de Lara, J.: Attributed typed triple graph transformation with inheritance in the double pushout approach. Tech. Rep. UC3M-TR-CS-2006-00, Universidad Carlos III, Madrid, Spain (2006)

[11] Guerra, E., de Lara, J.: Model view management with triple graph grammars. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) Proc. ICGT'06. LNCS, vol. 4178, pp. 351–366. Springer (2006)

[12] Habel, A., Pennemann, K.H.: Correctness of high-level transformation systems relative to nested conditions. Mathematical Structures in Computer Science 19, 1–52 (2009)

[13] Hermann, F., Ehrig, H., Golas, U., Orejas, F.: Formal Analysis of Functional Behaviour for Model Transformations Based on Triple Graph Grammars - Extended Version. Tech. Rep. 2010-8, TU Berlin, Fak. IV (2010)

[14] Hermann, F., Ehrig, H., Golas, U., Orejas, F.: Efficient Analysis and Execution of Correct and Complete Model Transformations Based on Triple Graph Grammars - Extended Version. Tech. Rep. 2010-13, TU Berlin, Fak. IV (2010)

[15] Hermann, F., Ehrig, H., Orejas, F., Golas, U.: Formal Analysis of Functional Behaviour of Model Transformations Based on Triple Graph Grammars. In: Proc. Int. Conf. on Graph Transformation (ICGT'10). LNCS, vol. 6372, pp. 155–170. Springer (2010)

[16] Kindler, E., Wagner, R.: Triple graph grammars: Concepts, extensions, implementations, and application scenarios. Tech. Rep. TR-ri-07-284, Department of Computer Science, University of Paderborn, Germany (2007)

[17] Königs, A., Schürr, A.: Tool Integration with Triple Graph Grammars - A Survey. In: Proc. SegraVis School on Foundations of Visual Modelling Techniques. ENTCS, vol. 148, pp. 113–150. Elsevier Science (2006)

[18] Lambers, L.: Certifying Rule-Based Models using Graph Transformation. Ph.D. thesis, Technische Universität Berlin (November 2009)

[19] Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: Tinhofer, G. (ed.) Proc. WG'94. LNCS, vol. 903, pp. 151–163. Springer (1994)

[20] Schürr, A., Klar, F.: 15 years of triple graph grammars. In: Ehrig, H., Heckel, R., Rozenberg, G., Taentzer, G. (eds.) Proc. ICGT'08. pp. 411–425. LNCS, Springer (2008)

[21] Taentzer, G., Biermann, E., Bisztray, D., Bohnet, B., Boneva, I., Boronat, A., Geiger, L., Geiß, R., Horvath, A., Kniemeyer, O., Mens, T., Ness, B., Plump, D., Vajk, T.: Generation of Sierpinski Triangles: A Case Study for Graph Transformation Tools. In: Schürr, A., Nagl, M., Zündorf, A. (eds.) Proc. AGTIVE'07. LNCS, vol. 5088, pp. 514 – 539. Springer (2008)

[22] Taentzer, G., Ehrig, K., Guerra, E., de Lara, J., Lengyel, L., Levendovsky, T., Prange, U., Varro, D., Varro-Gyapay, S.: Model Transformation by Graph Transformation: A Comparative Study. In: Proc. MoDELS 2005 Workshop MTiP'05 (2005)

[23] TFS-Group, TU Berlin: AGG (2009), `http://tfs.cs.tu-berlin.de/agg`