Formal Analysis of Model Transformations Based on Triple Graph Rules with Kernels

Hartmut Ehrig and Ulrike Prange

Technische Universität Berlin, Germany
{ehrig,uprange}@cs.tu-berlin.de

Abstract. Triple graph transformation has become an important approach for model transformations. Triple graphs consist of a source, a target and a connection graph. The corresponding rules also contain these parts and describe the simultaneous construction of both the source and the target model. From these rules, forward rules can be derived which describe the model transformation from a given source model to a target model. The forward transformation must be source consistent in order to define a valid model transformation. Source consistency implies that the source and the target model correspond to each other according to a triple transformation.

In this paper, the relationship between the source consistency of forward transformations, and NAC consistency and termination used in other model transformation approaches is analysed from a formal point of view. We define the kernel of a forward rule and construct NACs based on this kernel. Then we give sufficient conditions such that source consistency implies NAC consistency and termination. Moreover, we analyse how to achieve local confluence independent of source consistency. Both results together provide sufficient conditions for functional behaviour of model transformations. Our results are illustrated by an example describing a model transformation from activity diagrams to CSP.

1 Introduction

Model transformations are most important for model-driven software development. In recent years, triple graph grammars (TGGs) introduced by A. Schürr [1] have been shown to be a suitable basis to define model transformations in various application areas [2, 3]. TGGs are based on triple graphs and triple rules which allow to consistently co-develop two related structures modeled by graphs. TGG rules are triples of non-deleting graph rules and generate the language of triple graphs, which can be projected to the first and third component, usually called source and target language, respectively.

In [1], it was shown that a triple rule tr can be decomposed into a source rule tr_S and a forward rule tr_F, and similarly for the transformations, where the forward rules can be used to define model transformations from source to target models. Dually, triple rules and transformations can be decomposed into target and backward rules and transformations leading to model transformations from

H. Ehrig et al. (Eds.): ICGT 2008, LNCS 5214, pp. 178–193, 2008.

target to source models, and hence to bidirectional model transformations between different domain-specific modeling languages [4, 5, 6]. The decomposition result in [1] has been extended by a corresponding composition result in [5] leading to a bijective correspondence between triple graph transformation sequences and consistent sequences of corresponding source and forward transformation sequences.

A forward transformation $G_S \overset{tr_F^*}{\Longrightarrow} G$ is called a forward model transformation from the source projection of G_S to the target projection of G if it is "source consistent". As defined in [5], "source consistency" means that there is a generating sequence $\varnothing \overset{tr_S^*}{\Longrightarrow} G_S$ for the source model via the corresponding source rules such that the matches of the source components in $G_S \overset{tr_F^*}{\Longrightarrow} G$ are defined by the items generated by $\varnothing \overset{tr_S^*}{\Longrightarrow} G_S$. Actually, source consistency can be considered as a control condition for forward transformation sequences in order to obtain a model transformation from the source to the target model.

In most practical approaches using TGGs for model transformations there is no formal control condition for how to apply the forward rules. But the intuitive idea is to apply each forward rule to a corresponding item of the source graph [7]. In other graph transformation approaches to model transformations (see [5]), the control condition is given by different layers of rules with negative application conditions (NACs), where the rules in each layer are applied as long as possible. Termination is checked by suitable termination criteria [8].

In this paper, the relationship between source consistency of forward transformation sequences on the one hand and NAC consistency and termination on the other hand is analysed from a formal point of view. For this purpose, we consider triple graph rules $tr : L \to R$ with kernels, where the kernel is a distinguished triple $k(tr) = (x, r, y) \in R$ of connected nodes in the source, connection, and target graphs created by the triple rule tr. This allows to define the corresponding kernels $k(tr_S) = x$ and $k(tr_F) = k(tr)$ for the source and forward rules, respectively. Moreover, for a forward rule $tr_F = L \to R$ we define so-called kernel NACs $NAC(tr_F) = L \cup k(tr_F)$. This means that a NAC consistent forward transformation via tr_F cannot be applied twice at the same match.

Our first main result shows that the source consistency of $G_S \overset{tr_F^*}{\Longrightarrow} G$ implies NAC consistency and termination. In fact, an even slightly weaker notion called "kernel source consistency" is sufficient for this result, where source consistency is restricted to the kernel elements. In our second main result we give sufficient conditions for local confluence of forward rules with kernel NACs. Both results together lead to confluence and termination of forward rules with kernel NACs and hence to functional behaviour of the corresponding model transformation.

This paper is organized as follows: In Section 2, we review the basic definitions for triple graph grammars. An example model transformation from activity diagrams to CSP is introduced in Section 3. In Section 4, kernels of triple rules and some basic properties are defined. In Section 5, the main results are stated and proven, and applied to our example model transformation in Section 6. Finally, the conclusion is given in Section 7.

2 Triple Graph Transformation

In this section, we give a short introduction of the basic notions of triple graphs and triple graph grammars. which have been introduced in [1] as an approach to consistently develop related structures.

Triple graphs consist of a source, a target and a connection graph which is embedded into both source and target graphs. For the underlying graphs, we use for simplicity the category **Graphs** of graphs and graph morphisms, but the main results in [5] have been formulated already in the framework of adhesive HLR systems [8].

Definition 1 (Triple Graph). *A* triple graph $G = (S_G \overset{s_G}{\leftarrow} C_G \overset{t_G}{\rightarrow} T_G)$ *consists of graphs* S_G, C_G, *and* T_G *called source graph, connection graph, and target graph, respectively, and graph morphisms* $s_G : C_G \to S_G$ *and* $t_G : C_G \to T_G$.

For triple graphs G *and* H, *a* triple graph morphism $f = (f_S, f_C, f_T) : G \to H$ *consists of graph morphisms* $f_S : S_G \to S_H$, $f_C : C_G \to C_H$ *and* $f_T : T_G \to T_H$ *such that* $s_H \circ f_C = f_S \circ s_G$ *and* $t_H \circ f_C = f_T \circ t_G$.

Triple graphs and triple graph morphisms form the category **TripleGraphs**.

Remark 1. The category **TripleGraphs** is isomorphic to the comma category $\mathbf{ComCat}(\mathbf{S}, \mathbf{C}, \mathbf{T}, \{\mathbf{s} : \mathbf{C} \to \mathbf{S}, \mathbf{t} : \mathbf{C} \to \mathbf{T}\})$ where $\mathbf{S}, \mathbf{C}, \mathbf{T} = \mathbf{Graphs}$ and all functors are the identities on **Graphs**.

For simplicity, for a morphism $f : G \to H$ and $x \in S_G$ we write $f(x)$ instead of $f_S(x)$, and similarly for $x \in C_G, T_G$.

The concept of typing plays an important role in modeling. Thus we introduce typed triple graphs.

Definition 2 (Typed Triple Graph). *Given a triple graph* TG *as type graph, a* typed triple graph $(G, type^G)$ *consists of a triple graph* G *and a triple graph morphism* $type^G : G \to TG$. G *is said to by typed over* TG *with typing* $type^G$.

For typed triple graphs $(G, type^G)$ *and* $(H, type^H)$, *a* typed triple graph morphism f *is a triple graph morphism* $f : G \to H$ *such that* $type^H \circ f = type^G$.

Triple graphs typed over TG *and typed triple graph morphisms form the category* $\mathbf{TripleGraphs_{TG}}$.

Remark 2. The category $\mathbf{TripleGraphs_{TG}}$ is isomorphic to the slice category $\mathbf{TripleGraphs}\backslash TG$.

For the access to the different graph parts, projections of a triple graph are defined.

Definition 3 (Projection). *For a triple graph* $G = (S_G \overset{s_G}{\leftarrow} C_G \overset{t_G}{\rightarrow} T_G)$ *the* projections *of* G *to the source, connection, and target graph are defined by* $proj_S(G) = S_G$, $proj_C(G) = C_G$, *and* $proj_T(G) = T_G$, *respectively. In case of typed triple graphs, also the typing is projected.*

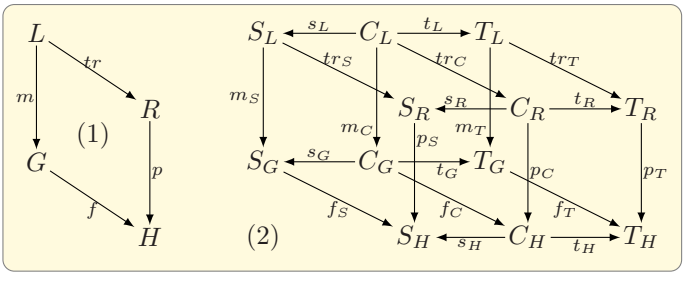
In the following, we give all the definitions based on triple graphs. They can be formulated analogously for typed triple graphs as used in Section 4.

A triple rule is a rule based on triple graphs as in standard DPO transformation. It constructs simultaneously source and target graphs as well as their connection graph. According to [1, 5], only non-deleting triple rules are allowed. This simplifies the definition since we do not need to consider an interface graph but only the left and right hand side of the rule.

Definition 4 (Triple Rule and Triple Transformation). *A* triple rule $tr = (L \overset{tr}{\to} R)$ *consists of triple graphs L and R, called left hand side and right hand side, respectively, and an injective triple graph morphism $tr : L \to R$.*

Given a triple rule tr, a triple graph G and a triple graph morphism $m : L \to G$, called match, a direct triple transformation $G \overset{tr,m}{\Longrightarrow} H$ *is given by the pushout* (1) *in* **TripleGraphs**, *which is the componentwise pushout* (2) *on the source, connection and target graphs in* **Graphs** *due to the comma category construction. The morphism p in pushout* (1) *is called* comatch.

A sequence of direct triple transformations is then called triple tranformation.



Since we consider only nondeleting injective rules $tr : L \to R$ we can assume w.l.o.g. that $L \subseteq R$ and all derived triple graphs are included in each other, i.e. for a transformation sequence $G_0 \overset{tr_1,m_1}{\Longrightarrow} G_1 \overset{tr_2,m_2}{\Longrightarrow} \ldots \overset{tr_n,m_n}{\Longrightarrow} G_n$ we have that $G_i \subseteq G_j$ for $i \leq j$.

To extend the expressiveness of triple graph transformations we define negative application conditions which restrict the applicability of a triple rule.

Definition 5 (Negative Application Condition). *Given a triple rule $tr = (L \overset{tr}{\to} R)$, a* negative application condition *(NAC) (N, n) consists of a triple graph N and a triple graph morphism $n : L \to N$.*

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

Up to now, the triple rules simultaneously create the source, connection and target graphs. But for a model transformation, some source model is given that has to be transformed into the corresponding target model. For this purpose, we can derive source and forward rules from a given triple rule. The source rule only creates a part of the source model, and the forward rule describes the transformation of this part to the target model.

Definition 6 (Source and Forward Rule). *Given a triple rule* $tr = (L \overset{tr}{\to} R)$, *the* source rule tr_S *and the* forward rule tr_F *are defined by* $tr_S = ((S_L \overset{\varnothing}{\leftarrow} \varnothing \overset{\varnothing}{\to} \varnothing) \overset{(tr_S,\varnothing,\varnothing)}{\longrightarrow} (S_R \overset{\varnothing}{\leftarrow} \varnothing \overset{\varnothing}{\to} \varnothing))$ *and* $tr_F = ((S_R \overset{tr_S \circ s_L}{\leftarrow} C_L \overset{t_L}{\to} T_L) \overset{(id_{S_R},tr_C,tr_T)}{\longrightarrow} R)$.

Now a triple graph grammar consists of a set of triple rules and a start graph.

Definition 7 (Triple Graph Grammar). *A* triple graph grammar $GG = (TR, S)$ *consists of a set* TR *of triple rules and a triple graph* S, *the start graph.*

For the rule set TR, *we get induced sets* $TR_S = \{tr_S \mid tr \in TR\}$ *and* $TR_F = \{tr_F \mid tr \in TR\}$ *of source and forward rules.*

For the relationship of triple rules with their source and forward rules, match and source consistency are introduced. Match consistency describes that in a transformation sequence, the forward rule is always applied via the comatch of the corresponding source rule for the source graph. Source consistency of a forward transformation requires a match consistent transformation. In [5], it is shown that a triple transformation can be split into match consistent source and forward transformations via the same rule sequence, and vice versa.

Definition 8 (Match Consistency). *Consider a triple transformation* $tt : \varnothing \overset{tr_S^*}{\Longrightarrow} G_S \overset{tr_F^*}{\Longrightarrow} G$ *where* $tr_S^* = (tr_{i,S})_{i=1,\ldots,n}$ *and* $tr_F^* = (tr_{i,F})_{i=1,\ldots,n}$ *are derived from the same triple rules* $tr^* = (tr_i)_{i=1,\ldots,n}$, *and we have matches* $m_{i,S}$ *and* $m_{i,F}$, *and comatches* $p_{i,S}$ *and* $p_{i,F}$, *respectively. Then* tt *is called* match consistent *if the source component of the match* $m_{i,F}$ *is completely determined by the comatch* $p_{i,S}$, *i.e.* $(m_{i,F})_S = (p_{i,S})_S$, *for* $i = 1, \ldots, n$.

Definition 9 (Source Consistency). *A forward triple transformation* $G_S \overset{tr_F^*}{\Longrightarrow} G$ *with* $tr_F^* = (tr_{i,F})_{i=1,\ldots,n}$ *is called* source consistent *if there exists a source triple transformation* $\varnothing \overset{tr_S^*}{\Longrightarrow} G_S$ *such that* $tr_S^* = (tr_{i,S})_{i=1,\ldots,n}$ *and* $\varnothing \overset{tr_S^*}{\Longrightarrow} G_S \overset{tr_F^*}{\Longrightarrow} G$ *is match consistent.*

According to [5], a source consistent transformation leads to a forward model transformation.

Definition 10 (Forward Model Transformation). *A forward triple transformation* $G_S \overset{tr_F^*}{\Longrightarrow} G$ *with* $G'_S = proj_S(G_S)$ *and* $G'_T = proj_T(G)$ *is called a* forward model transformation *from* G'_S *to* G'_T *if* $G_S \overset{tr_F^*}{\Longrightarrow} G$ *is source consistent.*

3 Example: From Activity Diagrams to CSP

In this section, we demonstrate the definitions from Section 2 on a model transformation from simple activity diagrams [9] with only actions, binary decisions and merges to communicating sequential processes (CSP) [10]. This transformation is a slightly smaller version of the case study proposed in [11]. Due to the

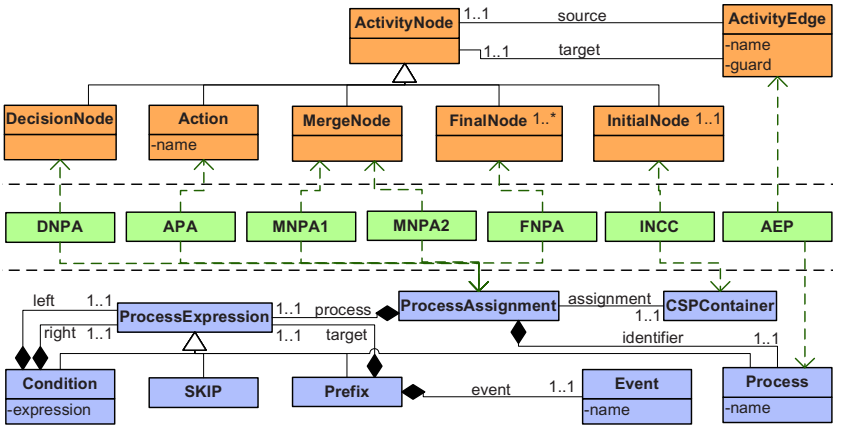


Fig. 1. The triple type graph

restrictions of the activity diagrams we can also restrict CSP to a SKIP process, prefix operations and conditions.

The triple type graph for the model transformation is shown in Fig. 1. In the upper part, the type graph for activity diagrams is shown. Basically, we have different kinds of activity nodes and activity edges, that are connected by source and target associations to the nodes. In the bottom of Fig. 1, the simplified type graph of CSP is depicted. Processes are defined via process assignments. A process expression can be a simple process, a prefix combining an event and a process, a condition, or a successful termination denoted by SKIP.

In Fig. 2, the triple rules for the consistent development of activity diagrams and the corresponding CSP models are depicted. We use a compact representation, where the stereotype ≪new≫ means that this element is created by the rule, and all other elements are already present in the left hand side. In the figure, on the left hand side of each rule the source graph is shown, followed by the connection graph and the target graph on the right hand side. The morphisms between these graphs are depicted by dashed arrows.

The triple rule $\mathsf{tr}^{\mathsf{InitialNode}}$ describes that an initial node corresponds to an CSP container where all other CSP elements are stored. With $\mathsf{tr}^{\mathsf{ActivityEdge}}$, an activity edge and its corresponding process are created. The other activity nodes correspond to different process assignments. With the triple rule $\mathsf{tr}^{\mathsf{Action}}$, an action and the corresponding prefix operation are created, while with $\mathsf{tr}^{\mathsf{FinalNode}}$ a final node and the corresponding SKIP process are defined. Finally, the rule $\mathsf{tr}^{\mathsf{DecisionNode}}$ handles the simultaneous creation of a binary decision and a condition, and $\mathsf{tr}^{\mathsf{MergeNode}}$ creates a binary merge and the corresponding identification of processes. Note that the rules $\mathsf{tr}^{\mathsf{ActivityEdge}}$ and $\mathsf{tr}^{\mathsf{Action}}$ have input parameters to define the attributes. To obtain a valid activity diagram, the rule $\mathsf{tr}^{\mathsf{InitialNode}}$ has to be applied exactly once, the rule $\mathsf{tr}^{\mathsf{FinalNode}}$ at least once, and each produced activity edge has to be connected by exactly one source and target association.

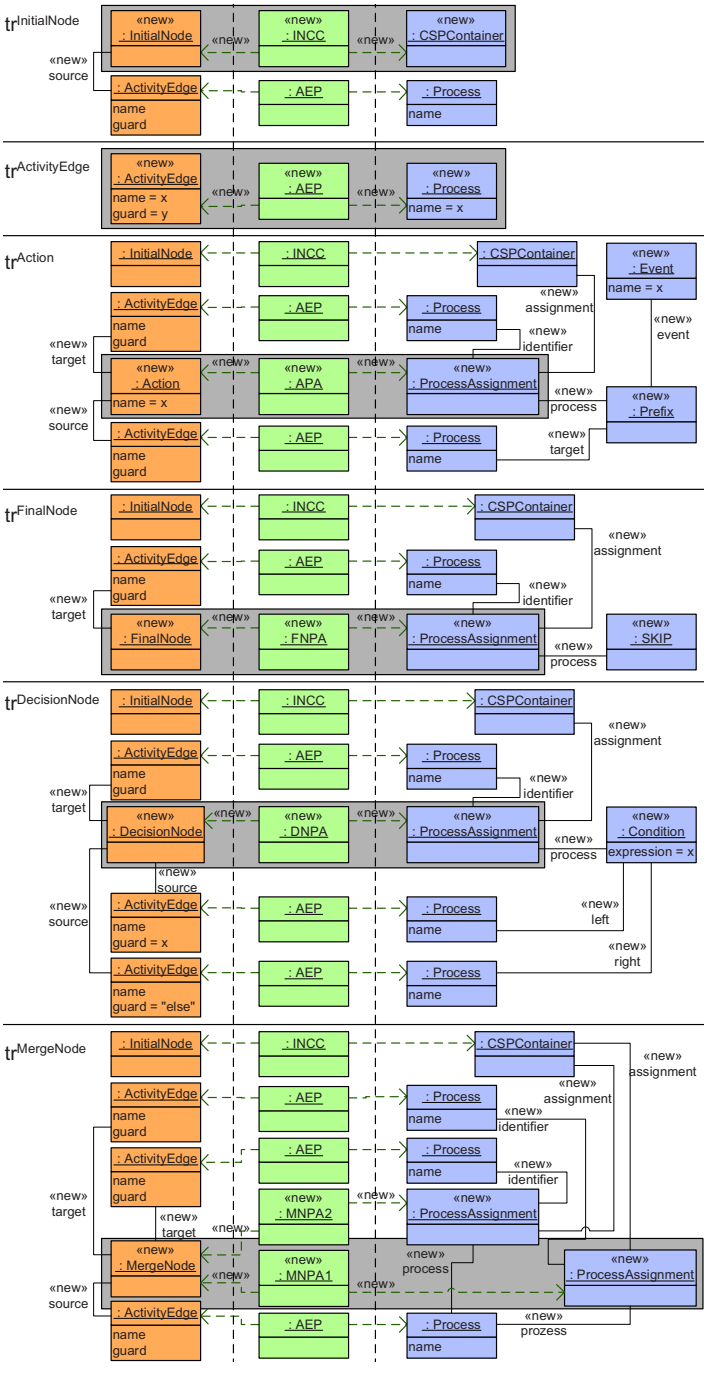


Fig. 2. The triple rules

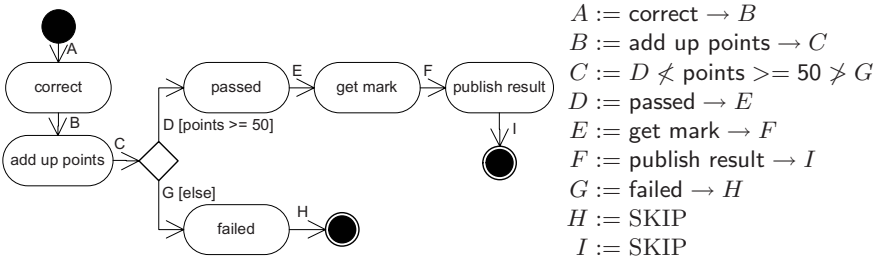


Fig. 3. Example model

In Fig. 3, an activity diagram and the corresponding process are depicted in concrete syntax, which are the results of the source and target projections of the transformation sequence $\varnothing \overset{tr^*}{\Longrightarrow} G$ with $tr^* = (\mathsf{tr}^{\mathsf{InitialNode}}, 9 \times \mathsf{tr}^{\mathsf{ActivityEdge}}, 6 \times \mathsf{tr}^{\mathsf{Action}}, \mathsf{tr}^{\mathsf{DecisionNode}}, 2 \times \mathsf{tr}^{\mathsf{FinalNode}})$ with suitable parameter values.

From the triple rules in Fig. 2, we can derive source and forward rules. For the source rules, we simply have to delete the connection and target graph parts of the rules. For the forward rules, the source graph of the left hand side of the forward rule is the source graph of the right hand side of the original rule, thus we only have to delete the ≪new≫-stereotypes of all elements in the source graph of a rule to obtain the corresponding forward rule. In Fig. 4, this is shown exemplarily for the rule $\mathsf{tr}^{\mathsf{FinalNode}}$ leading to the forward rule $\mathsf{tr}_{\mathsf{F}}^{\mathsf{FinalNode}}$.

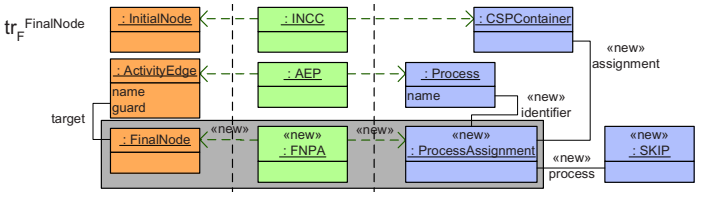


Fig. 4. A forward rule

Now the transformation $\varnothing \overset{tr^*}{\Longrightarrow} G$ from above can be decomposed into the transformations $\varnothing \overset{tr_S^*}{\Longrightarrow} G_S$ via the corresponding source rules and $G_S \overset{tr_F^*}{\Longrightarrow} G$ via the corresponding forward rules. In this case, $A = proj_S(G_S)$ is the activity diagram depicted in Fig. 3. The forward transformation $G_S \overset{tr_F^*}{\Longrightarrow} G$ is source consistent and leads to the forward model transformation from A to P, where $P = proj_T(G)$ is the CSP model in Fig. 3.

4 The Kernel Approach

In the kernel approach, we consider typed triple graphs and an empty start graph. For each rule, a distinguished kernel triple is selected. In this paper, we

only consider the source and forward rules, but the theory can be done similarly for the target and backward rules.

The kernel of each rule is a triple of nodes, one from each graph part of a triple graph, that is connected and generated by the rule.

Definition 11 (Kernel). *For a triple graph G, a node triple $(x,r,y) \in S_G \times C_G \times T_G$ is called* connected *if $s_G(r) = x$ and $t_G(r) = y$.*

Given a triple graph grammar $GG = (TR, \varnothing)$, we define for each rule $tr = (L \overset{tr}{\rightarrow} R) \in TR$ the kernel *$k(tr) = (x,r,y) \in R \backslash L = (S_R \backslash S_L) \times (C_R \backslash C_L) \times (T_R \backslash T_L)$ which is a connected node triple. Then $k(tr_S) = x$ and $k(tr_F) = (x,r,y)$ are the corresponding kernels of tr_S and tr_F, respectively.*

For the source and forward rules we want to have distinguished kernel typing, which means that elements of kernel types cannot be created as non-kernel types by any other rule. In our example, we have distinguished kernel typing (see Section 6).

Definition 12 (Distinguished Kernel Typing). *Define the* source kernel types *$KTYPE_S = \{type(x) \mid tr_S \in TR_S, k(tr_S) = x\}$ and the* forward kernel types *$KTYPE_F = \{type(r) \mid tr_F \in TR_F, k(tr_F) = (x,r,y)\}$.*

TR_S has distinguished kernel typing *if for all $tr_S \in TR_S$ and x created by tr_S, i.e. $x \in S_R \backslash S_L$, we have that $x \neq k(tr_S)$ implies $type(x) \notin KTYPE_S$.*

TR_F has distinguished kernel typing *if for all $tr_F \in TR_F$ and connected triples (x,r,y) created by tr_F we have that $(x,r,y) \neq k(tr_F)$ implies $type(r) \notin KTYPE_F$, where $(x,r,y) \in R$ created by $tr_F : L \rightarrow R$ means that $(x,r,y) \notin L$.*

Moreover, TR_F is called type functional *if for all tr_F, $tr'_F \in TR_F$ with kernels $k(tr_F) = (x,r,y)$ and $k(tr'_F) = (x',r',y')$ we have that $type(x) = type(x')$ implies $type(r) = type(r')$.*

Remark 3. If s_{TG} of the type graph TG is injective then TR_F is type functional in any case. Moreover, $type(r) = type(r')$ implies $type(y) = type(y')$.

In a triple transformation, the images of the kernels under the comatches are called the kernel elements of the resulting graph.

Definition 13 (Kernel Elements). *Consider a triple transformation $\varnothing \overset{tr_S^*}{\Longrightarrow} G_S$ with $tr_S^* = (tr_{i,S})_{i=1,\ldots,n}$ and comatches $p_{i,S}$. The* kernel elements *of G_S generated by tr_S^* are all elements $\overline{x_i} = p_{i,S}(x_i)$ for kernels $k(tr_{i,S}) = x_i$ and $i = 1,\ldots,n$.*

Consider a triple transformation $G_S \overset{tr_F^*}{\Longrightarrow} G$ with $tr_F^* = (tr_{i,F})_{i=1,\ldots,n}$ and comatches $p_{i,F}$. The kernel elements *of G generated by tr_F^* are all triples $(\overline{x_i}, \overline{r_i}, \overline{y_i}) = p_{i,F}(x_i, r_i, y_i)$ for kernels $k(tr_{i,F}) = (x_i, r_i, y_i)$ and $i = 1,\ldots,n$.*

In the following Facts 1 and 2, we show that for triple transformations with distinguished kernel typing, kernel elements are exactly the elements of kernel types.

Fact 1. *Consider a triple graph grammar* $(TR, \varnothing)$ *where* TR_S *has distinguished kernel typing and a transformation* $\varnothing \overset{tr_S^*}{\Longrightarrow} G_S$. *Then we have that* $\overline{x} \in G_S$ *is a kernel element if and only if* $type(\overline{x}) \in KTYPE_S$.

Proof. For $\overline{x} \in G_S$ with $type(\overline{x}) \in KTYPE_S$ there is a rule $tr_{i,S} = (L \overset{tr_{i,S}}{\rightarrow} R)$ such that $\overline{x}$ has been created by $tr_{i,S}$, and there is some $x \in S_R \backslash S_L$ with $p_{i,S}(x) = \overline{x}$. Suppose $\overline{x}$ is no kernel element, i.e. $x \neq k(tr_{i,S})$. Since TR_S has distinguished kernel typing it follows that $type(\overline{x}) = type(x) \notin KTYPE_S$, which is a contradiction. Thus, $\overline{x}$ is a kernel element of G_S. Vice versa, if $\overline{x}$ is a kernel element generated by $k(tr_{i,S}) = x$ then $type(\overline{x}) = type(x) \in KTYPE_S$.

Fact 2. *Consider a triple graph grammar* $(TR, \varnothing)$ *where* TR_F *has distinguished kernel typing and a triple transformation* $G_S \overset{tr_F^*}{\Longrightarrow} G$ *with* $proj_C(G_S) = \varnothing$. *Then we have that a connected triple* $(\overline{x}, \overline{r}, \overline{y}) \in G$ *is a kernel element if and only if* $type(\overline{r}) \in KTYPE_F$.

Proof. For $(\overline{x}, \overline{r}, \overline{y}) \in G$ with $type(\overline{r}) \in KTYPE_F$ there is a rule $tr_{i,F} = (L \overset{tr_{i,F}}{\rightarrow} R)$ such that $\overline{r}$ has been created by $tr_{i,F}$, because $proj_C(G_S) = \varnothing$. Then there is a triple $(x, r, y) \in R$ with $p_{i,F}(x, r, y) = (\overline{x}, \overline{r}, \overline{y})$. It follows that $(x, r, y) \notin L$, otherwise $\overline{r}$ was created earlier. Suppose $(x, r, y) \neq k(tr_{i,S})$. Since TR_F has distinguished kernel typing it follows that $type(\overline{r}) = type(r) \notin KTYPE_F$, which is a contradiction. Thus we have that $k(tr_{i,F}) = (x, r, y)$ and $(\overline{x}, \overline{r}, \overline{y})$ is a kernel element of G. Vice versa, if $(\overline{x}, \overline{r}, \overline{y})$ is a kernel element generated by $k(tr_{i,S}) = (x, r, y)$ then $type(\overline{r}) = type(r) \in KTYPE_F$.

Kernel match and source consistency is the restriction of source and match consistency to the kernel elements. Kernel consistency is easier to verify since only one element for each direct transformation has to be considered.

Definition 14 (Kernel Match Consistency). *Consider a triple transformation* $tt : \varnothing \overset{tr_S^*}{\Longrightarrow} G_S \overset{tr_F^*}{\Longrightarrow} G$ *where* $tr_S^* = (tr_{i,S})_{i=1,\ldots,n}$ *and* $tr_F^* = (tr_{i,F})_{i=1,\ldots,n}$ *are derived from the same triple rules* $tr^* = (tr_i)_{i=1,\ldots,n}$ *with kernels* $k(tr_i) = (x_i, r_i, y_i)$, *and we have matches* $m_{i,S}$ *and* $m_{i,F}$, *and comatches* $p_{i,S}$ *and* $p_{i,F}$, *respectively. The triple transformation* tt *is called* kernel match consistent *if* $m_{i,F}(x_i) = p_{i,S}(x_i)$ *for* $i = 1, \ldots, n$.

Definition 15 (Kernel Source Consistency). *A forward triple transformation* $G_S \overset{tr_F^*}{\Longrightarrow} G$ *with* $tr_F^* = (tr_{i,F})_{i=1,\ldots,n}$ *is called* kernel source consistent *if there exists a source triple transformation* $\varnothing \overset{tr_S^*}{\Longrightarrow} G_S$ *such that* $tr_S^* = (tr_{i,S})_{i=1,\ldots,n}$ *and* $\varnothing \overset{tr_S^*}{\Longrightarrow} G_S \overset{tr_F^*}{\Longrightarrow} G$ *is kernel match consistent.*

For each kernel element, from a triple transformation a unique rule can be identified which has created this element.

Fact 3. *Given a triple transformation* $tt : \varnothing \overset{tr_S^*}{\Longrightarrow} G_S \overset{tr_F^*}{\Longrightarrow} G$ *with* $tr_S^* = (tr_{i,S})_{i=1,\ldots,n}$ *and* $tr_F^* = (tr_{i,F})_{i=1,\ldots,n}$ *derived from the same triple rules* $tr^* = (tr_i)_{i=1,\ldots,n}$, *then we have that*

1. *For each kernel element* $\overline{x}$ *in* G_S *there is a unique* $i \in \{1,\ldots,n\}$ *such that* $\overline{x}$ *is generated by* $tr_{i,S}$, *i.e* $\overline{x} = \overline{x_i}$ *and* $\overline{x_i} \neq \overline{x_j}$ *for all* $j \neq i$.
2. *For each kernel element* $(\overline{x},\overline{r},\overline{y})$ *in* G *there is a unique* $i \in \{1,\ldots,n\}$ *such that* $(\overline{x},\overline{r},\overline{y})$ *is generated by* $tr_{i,F}$, *i.e.* $(\overline{x},\overline{r},\overline{y}) = (\overline{x_i},\overline{r_i},\overline{y_i})$ *and* $\overline{r_i} \neq \overline{r_j}$, $\overline{y_i} \neq \overline{y_j}$ *for all* $j \neq i$.
3. *If tt is kernel match consistent then in Item 2 also* $\overline{x_i} \neq \overline{x_j}$ *for all* $j \neq i$.

Proof. 1. For each $i = 1,\ldots,n$, when applying $tr_{i,S}$ a new kernel element $\overline{x_i} = p_{i,S}(x_i)$ is created in G_S such that $\overline{x_1},\ldots,\overline{x_n}$ are pairwise disjoint.
2. For each $i = 1,\ldots,n$, when applying $tr_{i,F}$ a kernel element $(\overline{x_i},\overline{r_i},\overline{y_i}) = p_{i,F}(x_i,r_i,y_i)$ is created in G. Since r_i and y_i are newly created by $tr_{i,F}$ it follows that $\overline{r_1},\ldots,\overline{r_n}$ and $\overline{y_1},\ldots,\overline{y_n}$ are pairwise disjoint.
3. In the case of kernel match consistency, $\overline{x_i}$ of a kernel triple $(\overline{x_i},\overline{r_i},\overline{y_i})$ is the kernel element $\overline{x_i}$ generated by the kernel of $tr_{i,S}$ such that $\overline{x_1},\ldots,\overline{x_n}$ are pairwise disjoint due to Item 1.

To allow the application of a forward rule to a source kernel element only once, kernel NACs for forward rules are defined.

Definition 16 (Kernel NAC). *For a forward rule* $tr_F = (L \overset{tr_F}{\rightarrow} R)$ *with kernel* $k(tr_F) = (x,r,y)$ *we define the* kernel NAC $NAC(tr_F) = (N,n)$ *with triple graph* $N = L \cup k(tr_F)$, $s_N(r) = x$, $t_N(r) = y$ *and inclusion* $n : L \rightarrow N$.

5 Results for Model Transformations in the Kernel Approach

In this section, we analyse how to achieve kernel source consistency and state the main results for forward transformations in the kernel approach. In Section 6, we apply these results to our example from Section 3.

A forward triple transformation $G_S \overset{tr_F^*}{\Longrightarrow} G$ with $tr_F^* = (tr_{i,F})_{i=1,\ldots,n}$ is kernel source consistent if G_S is generated by $\varnothing \overset{tr_S^*}{\Longrightarrow} G_S$ with corresponding source rule $tr_S = (tr_{i,S})_{i=1,\ldots,n}$ leading to kernel elements $\overline{x_i}$ in G_S that determine the kernel match for the forward rule $tr_{i,F}$. In other words, each forward rule $tr_{i,F}$ is applied exactly once at the kernel element $\overline{x_i}$ generated by the source rule $tr_{i,S}$.

For a forward transformation $G_S \overset{tr_F^*}{\Longrightarrow} G$ to become kernel source consistent we have to construct first a source generating sequence $\varnothing \overset{tr_S^*}{\Longrightarrow} G_S$ leading to kernel elements $\overline{x_1},\ldots,\overline{x_n} \in G_S$. This is a parsing problem for G_S, which may lead to nondeterministic results. Then we have to apply the corresponding forward rules $tr_{i,F}$ without kernel NACs at the kernel elements $\overline{x_i}$. If this is successful we obtain a kernel source consistent forward transformation $tt : G_S \overset{tr_F^*}{\Longrightarrow} G$ and under the conditions of Thm. 1 tt is NAC consistent and terminating.

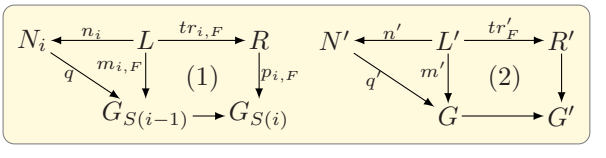
Obviously, source consistency of $G_S \overset{tr_F^*}{\Longrightarrow} G$ implies kernel source consistency. Vice versa, kernel source consistency implies source consistency if all matches are uniquely determined by the kernel matches (see Thm. 2).

Theorem 1 (NAC Consistency and Termination). *Consider a triple graph grammar $(TR, \varnothing)$ where TR_S, TR_F have distinguished kernel typing, a kernel source consistent forward triple transformation $tt : G_S \overset{tr_F^*}{\Longrightarrow} G$, and forward rules with kernel NACs. Then we have that:*

1. *tt is NAC consistent.*
2. *tt is terminating if TR_F is type functional.*

Proof. Let $tt : G_S = G_{S(0)} \overset{tr_{1,F}}{\Longrightarrow} G_{S(1)} \overset{tr_{2,F}}{\Longrightarrow} \ldots \overset{tr_{n,F}}{\Longrightarrow} G_{S(n)} = G$. Since tt is kernel source consistent there exists a triple transformation $\varnothing \overset{tr_S^*}{\Longrightarrow} G_S$ such that $\varnothing \overset{tr_S^*}{\Longrightarrow} G_S \overset{tr_F^*}{\Longrightarrow} G$ is kernel match consistent.

1. Suppose that tt is not NAC consistent. This means that there is a rule $tr_{i,F} = (L \overset{tr_{i,F}}{\rightarrow} R)$ with match $m_{i,F}$, comatch $p_{i,F}$, and kernel NAC (N_i, n_i) such that $G_{S(i-1)} \overset{tr_{i,F}}{\Longrightarrow} G_{S(i)}$ is not NAC consistent, i.e. there is a triple graph morphism $q : N_i \to G_{S(i-1)}$ such that $q \circ n_i = m_{i,F}$.
   For the kernel $k(tr_{i,F}) = (x_i, r_i, y_i)$ of $tr_{i,F}$ we have the kernel element $p_{i,F}(x_i, r_i, y_i) = (\overline{x_i}, \overline{r_i}, \overline{y_i}) \in G_{S(i)}$ and $q(x_i, r_i, y_i) = (\overline{x}, \overline{r}, \overline{y}) \in G_{S(i-1)}$ with $x_i \in S_L$ and $m_{i,F}(x_i) = \overline{x}$. The commutativity of (1) with horizontal inclusions and $x_i \in S_L$ imply that $\overline{x} = m_{i,F}(x_i) = p_{i,F}(x_i) = \overline{x_i}$.
   (x_i, r_i, y_i) is a connected triple and hence also $(\overline{x}, \overline{r}, \overline{y})$ is connected in $G_{S(i-1)}$. Since $type(\overline{r}) = type(r_i) \in KTYPE_F$, Fact 2 implies that $(\overline{x}, \overline{r}, \overline{y})$ is a kernel element of G, and by Fact 3 Item 2 there is a unique j such that $(\overline{x}, \overline{r}, \overline{y}) = (\overline{x_j}, \overline{r_j}, \overline{y_j})$ is generated by $tr_{j,F}$. Obviously, $j < i$ because $(\overline{x}, \overline{r}, \overline{y}) \in G_{S(i-1)}$. Now Fact 3 Item 3 implies that $\overline{x_i} \neq \overline{x_j} = \overline{x}$, which contradicts $\overline{x_i} = \overline{x}$. Hence tt is NAC consistent.



2. Suppose now that tt is not terminating, i.e. there is a direct triple transformation $G \overset{tr'_F, m'}{\Longrightarrow} G'$ for some triple rule $tr'_F = (L' \overset{tr'_F}{\rightarrow} R') \in TR_F$ with kernel $k(tr'_F) = (x', r', y')$ and kernel element $(\overline{x'}, \overline{r'}, \overline{y'}) \in G'$ with $\overline{x'} \in G_S$ and $type(\overline{x'}) = type(x') \in KTYPE_S$.

   By Fact 1, $\overline{x'}$ is a kernel element of G_S and by Fact 3 Item 1 there is a unique i such that $\overline{x'} = \overline{x_i}$ is generated by $tr_{i,S}$ with kernel $k(tr_{i,S}) = x_i$. Kernel match consistency of $\varnothing \overset{tr_S^*}{\Longrightarrow} G_S \overset{tr_F^*}{\Longrightarrow} G$ implies that the kernel $k(tr_{i,F}) = (x_i, r_i, y_i)$ implies a kernel element $(\overline{x_i}, \overline{r_i}, \overline{y_i}) \in G$ with $\overline{x_i} = \overline{x'}$. It follows that $type(x') = type(\overline{x'}) = type(\overline{x_i}) = type(x_i)$, and by type functionality of TR_F also $type(r') = type(r_i) = type(\overline{r_i})$ and $type(y') = type(y_i) = type(\overline{y_i})$. For the kernel NAC $NAC(tr'_F) = (N', n')$, we define a morphism $q' \colon N' \to G$ by $q'|_{L'} = m'$ with $m'(x') = \overline{x'} = \overline{x_i}$ and $q'(x', r', y') =$

$(\overline{x_i}, \overline{r_i}, \overline{y_i})$. q' is a valid morphism because it preserves the typing, and we have that $q' \circ n' = m'$. Thus, $G \overset{tr'_F}{\Longrightarrow} G'$ is not NAC consistent, hence tt is terminating.

Remark 4. Since source consistency implies kernel source consistency this theorem also holds for source consistent forward triple transformations $tt : G_S \overset{tr^*_F}{\Longrightarrow} G$.

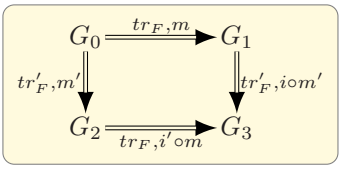
Thm. 1 and the following Thm. 2 concerning local confluence are applied to our example from Section 3 in Section 6.

Theorem 2 (Local Confluence). *Consider a triple graph grammar $(TR, \varnothing)$ where TR_F has distinguished kernel typing and kernel NACs. If we have that*

(1) The rules in TR_F are uniquely determined by the left hand sides, i.e. for rules $tr_F = (L \overset{tr_F}{\to} R)$ and $tr'_F = (L' \overset{tr'_F}{\to} R')$, $L \cong L'$ implies $tr_F = tr'_F$.

(2) The matches are uniquely determined by the kernel matches, i.e. given kernels $k(tr_F) = (x, r, y)$, $k(tr'_F) = (x', r', y')$ and matches $m : L \to G_0$, $m' : L' \to G_0$, $m(x) = m'(x')$ implies $m = m'$ with $L = L'$.

then the forward rules TR_F with kernel NACs are locally confluent. This means that given $G_0 \overset{tr_F, m}{\Longrightarrow} G_1$, $G_0 \overset{tr'_F, m'}{\Longrightarrow} G_2$ then we have either $G_1 \cong G_2$ or the direct transformations are parallel independent with NACs leading to the local Church-Rosser property, i.e. there are transformations $G_1 \overset{tr'_F, i \circ m'}{\Longrightarrow} G_3$, $G_2 \overset{tr_F, i' \circ m}{\Longrightarrow} G_3$.

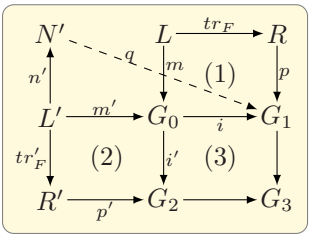


Proof. For given forward rules $tr_F, tr'_F \in TR_F$ and matches $m : L \to G_0$, $m' : L' \to G_0$ we have the following cases:

1. $m = m'$, which implies $L = L'$, and property (1) implies that also $tr_F = tr'_F$. Then the uniqueness of pushouts implies that $G_1 \cong G_2$.
2. $m \neq m'$. For the forward rules, we have kernels $k(tr_F) = (x, r, y)$ and $k(tr'_F) = (x', r', y')$ and $m(x) = \overline{x}$, $m'(x') = \overline{x}'$.

   Consider now the kernel NAC $NAC(tr'_F) = (N', n')$ with $N' = L' \cup k(tr'_F)$. Given the transformation $G_0 \overset{tr_F, m}{\Longrightarrow} G_1$ with pushout (1) we have to show that $i \circ m'$ is NAC consistent, i.e. there does not exist a morphism $q : N' \to G_1$ with $q \circ n' = i \circ m'$. Suppose that such a q exists, then using $k(tr'_F) = (x', r', y')$ there is a connected triple $q(x', r', y') = (\overline{x}', \overline{r}', \overline{y}')$ in G_1 with $type(x') = type(\overline{x}')$, $type(r') = type(\overline{r}')$ and $type(y') = type(\overline{y}')$. Since m' satisfies N', $(\overline{x}', \overline{r}', \overline{y}') \notin G_0$, but created by tr_F. Hence there is a connected triple $(x_2, r_2, y_2) \in R$, $(x_2, r_2, y_2) \notin L$ with $p(x_2, r_2, y_2) = (\overline{x}', \overline{r}', \overline{y}')$. Since $type(r_2) = type(\overline{r}') = type(r') \in KTYPE_F$ it follows that $k(tr_F) = (x, r, y) = (x_2, r_2, y_2)$ from distinguished kernel typing of TR_F. But this implies that $m(x) = m(x_2) = p(x_2) = \overline{x}' = m'(x')$ by commutativity of (1), and from property *(2)* it follows that $m = m'$, which is a contradiction.

Hence $i \circ m'$ is NAC consistent. Similarly, $i' \circ m$ is NAC consistent, and with pushout (3) the triple graph G_3 is the result of both transformations $G_1 \overset{tr'_F, i \circ m'}{\Longrightarrow} G_3$ and $G_2 \overset{tr_F, i' \circ m}{\Longrightarrow} G_3$. Thus we have parallel independence with NACs and the local Church-Rosser property leading to local confluence. □



Since local confluence and termination imply confluence, we get the following sufficient conditions for functional behaviour of forward model transformations.

Theorem 3 (Functional Behaviour). *Under the assumptions of Thms. 1 and 2, forward model transformations have functional behaviour, i.e. they are terminating and confluent.*

6 Analysis of the Example Model Transformation

Now we want to analyse the model transformation described in Section 3. The kernels for the triple rules in Fig. 2 are the triples in a box shaded in gray. Thus we have that $KTYPE_S = \{\mathsf{InitialNode}, \mathsf{ActivityEdge}, \mathsf{Action}, \mathsf{FinalNode}, \mathsf{DecisionNode}, \mathsf{MergeNode}\}$ and $KTYPE_F = \{\mathsf{INCC}, \mathsf{AEP}, \mathsf{APA}, \mathsf{FNPA}, \mathsf{DNPA}, \mathsf{MNPA1}\}$, and it is easy to see that both TR_S and TR_F have distinguished kernel typing and TR_F is type functional. Moreover, the forward rules are uniquely determined by the left hand sides.

Now consider the forward triple transformation $G_S \overset{tr^*_F}{\Longrightarrow} G$ leading to the forward model transformation from A to P from Section 3, with A and P depicted in Fig. 3. Since this forward triple transformation is source consistent, it is also kernel source consistent. From Thm. 1 it follows that it is then NAC consistent and terminating if we consider forward rules with kernel NACs.

In a valid activity diagram without merge nodes, the matches for the forward rules are uniquely determined by the kernel matches. To see this we have to take a closer look at the triple rules. First we know that there is only one initial node. This means, whenever an initial node is present in the left hand side its match is uniquely determined. Moreover, the kernel element and its match induce the complete match because of the graph structure, and in case of the triple rule $\mathsf{tr}_\mathsf{F}^\mathsf{DecisionNode}$ also the value of the attributes. This means that for the triple rules of our forward model transformation from A to P the conditions of Thm. 2 are fulfilled and this model transformation is confluent. Thus, the target model P is unique for the source model A.

On the other hand, for the triple rule $\mathsf{tr}_\mathsf{F}^\mathsf{MergeNode}$ the matches are not uniquely determined by the kernel matches. This is easy to see, since for a valid match we can swap the matches of the both activity edges which have the merge node as a target. Thus we cannot apply Thm. 2. When applying the forward rule via both matches, we get two different triple graphs which only differ in the mappings of the nodes MNPA1 and MNPA2. Note, that these two direct triple

transformations are not confluent, since no rule can be applied to this merge node due to the kernel NAC. Nevertheless, we have confluence concerning the target models. In fact, the resulting target models are already isomorphic, since the types of the connection nodes are not relevant for the target model.

7 Conclusion

In this paper, we have started a formal analysis of model transformations based on triple rules which have been introduced in [1] and applied to various application areas [2, 3, 7]. In [1], an important connection between the triple rules and the corresponding forward rules and transformations was given. This result was extended in [5] to a bijective correspondence between triple transformations $\varnothing \stackrel{tr^*}{\Longrightarrow} G$ based on triple rules tr^* and match consistent sequences $\varnothing \stackrel{tr_S^*}{\Longrightarrow} G_S \stackrel{tr_F^*}{\Longrightarrow} G$ based on corresponding source rules tr_S^* and target rules tr_F^*. This allows to define model transformations formally by source consistent forward transformation sequences $G_S \stackrel{tr_F^*}{\Longrightarrow} G$.

In order to analyse this kind of model transformations on a formal basis, we have defined the kernel of a forward rule and constructed a NAC based on this kernel. This allows to define kernel source consistency as source consistency restricted to kernel elements. Intuitively, this means that each forward rule is applied exactly once to the distinguished kernel element in the source graph generated by the corresponding source rule.

In our main results, we show that kernel source consistency implies NAC consistency and termination, and we give sufficient conditions for local confluence, which leads to functional behaviour of forward model transformations. Although the forward rules are non-deleting, this result is non-trivial because we have to ensure NAC consistency.

For a discussion of the relationship between model transformations based on triple and plain graph grammars we refer to [6].

At the moment, the conditions for distinguished kernel typing and type functionality are very restrictive, and only a subset of practical model transformations can be analyzed by our approach. In future work we want to extend our approach, in particular to forward rules that create either target or connection elements, but not both. As the discussion in Section 6 shows, the properties for local confluence in Thm. 2 are very restrictive. It would be interesting to analyse how these conditions and the concept for local confluence can be weakened, for example concerning confluence only on the target models. Moreover, we want to check under what conditions kernel source consistency is not only sufficient but also necessary for NAC consistency and termination. In addition to kernel NACs we want to consider also other NACs for forward rules. In this context, we want to apply the Critical Pair Lemma with NACs shown in [12] to forward transformations and verify confluence for other practical examples.

All our results dually hold for target and backward rules, which can be derived from triple rules similar to source and forward rules. This allows to

analyze bidirectional model transformations between source and target languages, especially the problem of how to obtain functional inverse model transformations.

References

[1] Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) WG 1994. LNCS, vol. 903, pp. 151–163. Springer, Heidelberg (1995)
[2] Aschenbrenner, N., Geiger, L.: Transforming Scene Graphs Using Triple Graph Grammars - A Practice Report. In: Proceedings of AGTIVE 2007 (2007)
[3] Guerra, E., de Lara, J.: Model View Management with Triple Graph Transformation Systems. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 351–366. Springer, Heidelberg (2006)
[4] Giese, H., Wagner, R.: Incremental Model Synchronization with Triple Graph Grammars. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 543–557. Springer, Heidelberg (2006)
[5] Ehrig, H., Ehrig, K., Ermel, C., Hermann, F., Taentzer, G.: Information Preserving Bidirectional Model Transformations. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 72–86. Springer, Heidelberg (2007)
[6] Ehrig, H., Ehrig, K., Hermann, F.: From Model Transformation to Model Integration Based on the Algebraic Approach to Triple Graph Grammars. ECEASST (to appear, 2008)
[7] Kindler, E., Wagner, R.: Triple Graph Grammars: Concepts, Extensions, Implementations, and Application Scenarios. Technical Report tr-ri-07-284, University of Paderborn (2007)
[8] Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs. Springer, Heidelberg (2006)
[9] OMG: Unified Modeling Language, version 2.1.1 (2006)
[10] Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Englewood Cliffs (1985)
[11] Bisztray, D., Ehrig, K., Heckel, R.: Case Study: UML to CSP Transformation. In: AGTIVE 2007 Graph Transformation Tool Contest (2007)
[12] Lambers, L.: Adhesive High-Level Replacement Systems with Negative Application Conditions. Technical Report 2007/14, TU Berlin (2007)