Towards Local Confluence Analysis for Amalgamated Graph Transformation *

Gabriele Taentzer[1] and Ulrike Golas[2]

[1] Philipps-Universität Marburg, Germany
taentzer@informatik.uni-marburg.de
[2] Humboldt-Universität zu Berlin & Zuse Institut Berlin, Germany
ulrike.golas@hu-berlin.de / golas@zib.de

Abstract. Amalgamated graph transformation allows to define schemes of rules coinciding in common core activities and differing over additional parallel independent activities. Consequently, a rule scheme is specified by a kernel rule and a set of extending multi-rules forming an interaction scheme. Amalgamated transformations have been increasingly used in various modeling contexts.
Critical Pair Analysis (CPA) can be used to show local confluence of graph transformation systems. It is an open challenge to lift the CPA to amalgamated graph transformation systems, especially since infinite many pairs of amalgamated rules occur in general. As a first step towards an efficient local confluence analysis of amalgamated graph transformation systems, we show that the analysis of a finite set of critical pairs suffices to prove local confluence.

Keywords: Amalgamated graph transformation, parallel independence, critical pair analysis

1 Introduction

In model-based software development, models play a primary role w.r.t. requirements elicitation, software design and software validation. Model changes can be well specified as model transformations. Algebraic graph transformation has been shown to be a suitable underlying formal framework of model transformations, especially of in-place transformations [4]. If several developers work on the same model concurrently, they may run into conflicts that have to be resolved. To analyze such conflicts as early as possible, critical pair analysis has been used to check transformation rules at specification time, i.e., before run time.

While simple model changes can be well specified by the application of simple rules, this is usually not sufficient for more complex model changes. Amalgamated graph transformation has been used to specify core activities equipped with a number of optional or context-dependent activities (e.g., [2, 3, 7, 12]). A typical example of such complex model changes are model refactorings where, e.g., equal attributes in subclasses are pulled up to one attribute in their super class. Concurrently working developers aim to understand when model changes can be applied in parallel and when they

*This work is partly supported by a Humboldt Post-Doc Fellowship as part of the Excellence Initiative by the German federal and state governments.

are a potential source for conflicts. Being in conflict, it would be interesting to analyze if and how these conflicts can be resolved. Hence, the notions of parallel independence, conflict and conflict resolution have to be lifted to amalgamated graph transformation.

An amalgamated graph transformation is specified by a so-called interaction scheme containing a kernel rule and a set of extending multi-rules. While the kernel rule is intended to be matched exactly once, each multi-rule may be matched arbitrarily often. An amalgamated rule over an interaction scheme contains at least the kernel rule and arbitrary many copies of multi-rules overlapping at the kernel rule. Hence, an interaction scheme specifies infinitely many amalgamated rules in general.

While the check for parallel independence of rules and transformations is well-known and used to support parallel model changes, parallel independence of amalgamated rules and transformations has hardly been investigated. In [8, 9], the parallel independence of amalgamated graph transformations has been characterized as transformations that can be executed sequentially in either order. This semantic characterization cannot be checked at specification time, i.e., on the level of rule schemes. An easy-to-check criterion for the parallel independence on the basis of interaction schemes is the first contribution of this paper: If two interaction schemes are parallel and sequentially independent, all their induced amalgamated transformations are parallel independent and can be sequentialized in any order. We assume that the occurring matches are maximal, i.e, that always the largest possible amalgamated rules are applied.

The second contribution of this paper is concerned with the analysis and resolution of conflicts between rule schemes. It is based on the well-known critical pair analysis [13, 6]: If a critical pair can be restricted to a smaller one showing the same kind of conflict and resolving it in the same way, then this pair does not have to be considered during conflict analysis. We show that only finitely many critical pairs cannot be further restricted. Thus, the usually infinite set of critical pairs for rule schemes can be reduced to a finite set being enough to show local confluence of the transformation system.

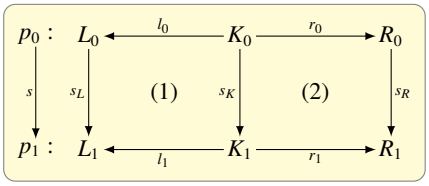
The paper is organized as follows: Section 2 presents the necessary basic notions on amalgamated graph transformation. Most of them are recalled from or similar to [8, 9]. Parallel independence is considered in Section 3 while the conflict analysis of rule schemes is presented in Section 4. The paper is concluded in Section 5.

2 Amalgamated Graph Transformation

In this section, we review the formal foundations of amalgamated graph transformation based on the well-known double-pushout approach. We assume the reader to be familiar with this approach (see, e.g., [6] for an introduction and a large number of theoretical results). We concentrate on the presentation of only those concepts and results needed for the confluence analysis. For simplicity, here we present the theory without application conditions and attributes. In fact, the theory in [8, 9] is presented in the categorical framework of $\mathcal{M}$-adhesive transformation systems for rules with nested application conditions in the sense of [11]. In particular, this means that in the following the graphs and morphisms can be any objects from $\mathcal{M}$-adhesive categories, e.g., any kinds of (labeled, typed, attributed) graphs.

Formally, a kernel morphism describes how the kernel rule is embedded into a multi-rule (recall the definition of [9]).

Definition 2.1 (Rule and kernel morphism). *Given rules* $p_0 = (L_0 \xleftarrow{l_0} K_0 \xrightarrow{r_0} R_0)$ *and* $p_1 = (L_1 \xleftarrow{l_1} K_1 \xrightarrow{r_1} R_1)$ *with injective morphisms* l_i, r_i *for* $i \in \{0,1\}$*, a* rule morphism $s : p_0 \to p_1$*,* $s = (s_L, s_K, s_R)$ *consists of injective morphisms* $s_L : L_0 \to L_1$*,* $s_K : K_0 \to K_1$*, and* $s_R : R_0 \to R_1$ *such that in the following diagram* (1) *and* (2) *commute. s is an* isomorphism*, if* s_L*,* s_K*, and* s_R *are isomorphisms. A rule is* finite*, if all occurring objects are finite.*



If (1) *and* (2) *are pullbacks and* (1) *has a pushout (PO) complement for* $s_L \circ l_0$*, s is called* kernel morphism*. Then* p_0 *is called* kernel rule *and* p_1 multi-rule*.*

The technical preconditions ensure that the multi-rule is consistent w.r.t. the kernel rule: The requirement of (1) and (2) being pullbacks ensures that the multi-rule deletes and creates the elements matched by the kernel rule in the same way. The existence of the PO complement of (1) makes sure that p_0 can be applied to L_1. This condition is needed to construct the complement rule later.

Example 2.2 (Specification of refactoring "Push Down Attribute"). As running example, we consider a graph representation of simple class models and some refactorings to improve their structure. Our class models contain classes (typed by "C"), attributes (typed by "A"), a generalization relation between classes (typed by "G"), and references between classes (typed by "R").

In Figure 1, we show the kernel and a multi-rule for the refactoring "Push Down Attribute". The kernel rule takes an attribute in a super class (being target of a generalization) and pushes it down to one of its subclasses (being source of the connecting generalization). The multi-rule specifies that the attribute in the super class is also pushed down to any other subclass. This refactoring is useful if a common attribute shall be individually changed in the subclasses. Note that the intermediate graph K of each depicted rule can be deduced from the graphical notation by considering the overlapping graph of the left- and right-hand sides. The overlapping graph is exactly that subgraph which is enhanced by numbers occurring in both sides. These numbers specify the morphisms going to the left- and right-hand sides as well as the kernel morphisms. Note that we only number those elements that are actually mapped.

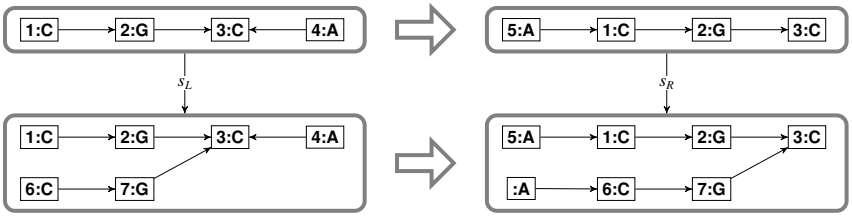


Fig. 1. Kernel and multi-rules for refactoring "Push Down Attribute"

This rule morphism satisfies the additional conditions for a kernel morphism: The relation between kernel and multi-rules is characterized by two pullbacks which means that all kernel actions are reflected in the multi-rule. Moreover, the required PO complement exists being the left-hand side of the multi-rule without the A-node and its adjacent edge. Although it is the intermediate graph of the multi-rule here, this is not generally the case.

Inverting the kernel and multi-rules in Figure 1, we get a specification of the refactoring "Pull Up Attribute" assuming that all the attributes in the subclasses have the same name and type (which is not specified here). In this simple example, we just check if each subclass has an attribute. In that case, one attribute of each subclass is deleted and a new one is created in their superclass. This refactoring is usually applied to lift common attributes to superclasses and hence, to reduce redundancy.

To obtain a kernel morphism also for these rules, we have to check that the right-hand side has a PO complement as well. Actually, this is the case using the right-hand side of the multi-rule without the upper attribute. Note that this graph is not the intermediate one of the multi-rule.

For a given kernel morphism, the complement rule is the remainder of the multi-rule after the application of the kernel rule, i.e. it describes what the multi-rule does in addition to the kernel rule. Intuitively, the complement rule is the smallest rule that extends K_0 such that it creates and deletes all those elements handled by the multi- but not by the kernel rule. It is important to decompose amalgamated transformations into kernel and complement rule applications (see Corollary 2.13). There is a canonical way to construct the complement rule for a given kernel morphism; due to its complex construction, we only give an example here and refer to [8, 9].

Example 2.3 (Complement rule). Figure 2 shows the complement rule of the kernel and multi-rules in Figure 1. Note that the general attribute is deleted by the kernel rule, which also inserts an attribute into one subclass. All other subclasses can be equipped by a new attribute applying the complement rule thereafter.

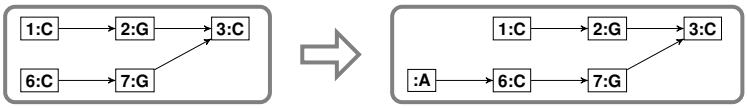


Fig. 2. Complement rule for refactoring "Push Down Attribute"

A bundle of kernel morphisms over a common kernel rule forms an interaction scheme. An interaction scheme instance contains copies of kernel morphisms for different matches of multi-rules of a chosen interaction scheme.

Definition 2.4 (Interaction scheme (instance)). *Given a rule set Basic* $= \{p_i = (L_i \xleftarrow{l_i} K_i \xrightarrow{r_i} R_i), i = 0, \ldots, n\}$*, an* interaction scheme *s over Basic is a bundle of kernel morphisms* $s = (s_i : p_0 \to p_i)_{i=0,\ldots,n}$ *with* $s_0 = id_{p_0}$*. For* $n = 0$*, s consists of the rule* p_0 *only. An* interaction scheme instance s_{inst} *over s is an interaction scheme where each kernel morphism of* s_{inst} *is isomorphic to some kernel morphism of s. An interaction scheme* $s' = (s'_i : p'_0 \to p'_i)_{i=0,\ldots,n}$ *is* more parallel *than s if* p'_0 *is a subrule of* p_0 *with inclusion* $i_0 : p'_0 \to p_0$*,* $p'_i = p_i$ *for* $i > 0$*, and* $s'_i = s_i \circ i_0$ *for all* $i = 0, \ldots, n$*.*

Example 2.5 (Interaction schemes for refactorings). In Figure 3, an interaction scheme for replacing an inheritance relation with a delegation is shown. This classical refactoring is defined for all attributes of a super class being copied to its subclass as soon as the generalization relation between these classes is replaced by a reference. This is necessary since after the refactoring the class is not a subclass anymore. Note that the conditions for kernel morphisms are also satisfied here.

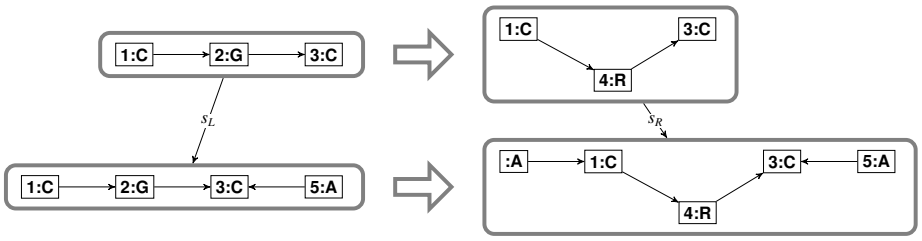


Fig. 3. Interaction scheme for refactoring "Replace Inheritance With Delegation"

Figure 4 shows the specification of refactoring "Remove All Inheritances" which detaches all subclasses from their super class. The refactoring shall only be applied if the superclass is empty, i.e., does not have any references or attributes.

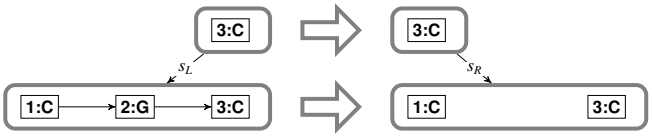


Fig. 4. Interaction scheme for refactoring "Remove All Inheritances"

Figure 5 shows a simple rule which deletes a class. It can be considered as an interaction scheme over $n = 0$. Combining this rule with the interaction scheme "Remove all inheritances" would be an approach to specify the refactoring "Delete Super Class". Imagine similar rules as in Figure 4 but without Class 1:C on the right-hand side. Unfortunately, they would not form an interaction scheme since there is no PO complement on the left. The problem is that, once the super class is deleted, we do not find a complement rule deleting all inheritance relations. It follows that these have to be deleted first. Hence, two steps are required to delete a super class with all incoming inheritance relations.

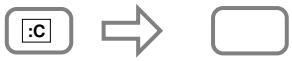


Fig. 5. Interaction scheme for refactoring "Delete Class"

Given an interaction scheme s which describes the basic actions in its kernel rule and a set of multi-rules, we need to construct an *interaction scheme instance* over s for a given graph and kernel rule match. This interaction scheme instance contains a certain number of multi-rule copies for each multi-rule of the basic scheme. To do so, we search for all different multi-rule matches which overlap in the kernel match *only*. The number of different multi-rule matches determines how many copies are included in the graph-specific interaction scheme instance which is the starting point for the amalgamated rule construction defined in Def. 2.7.

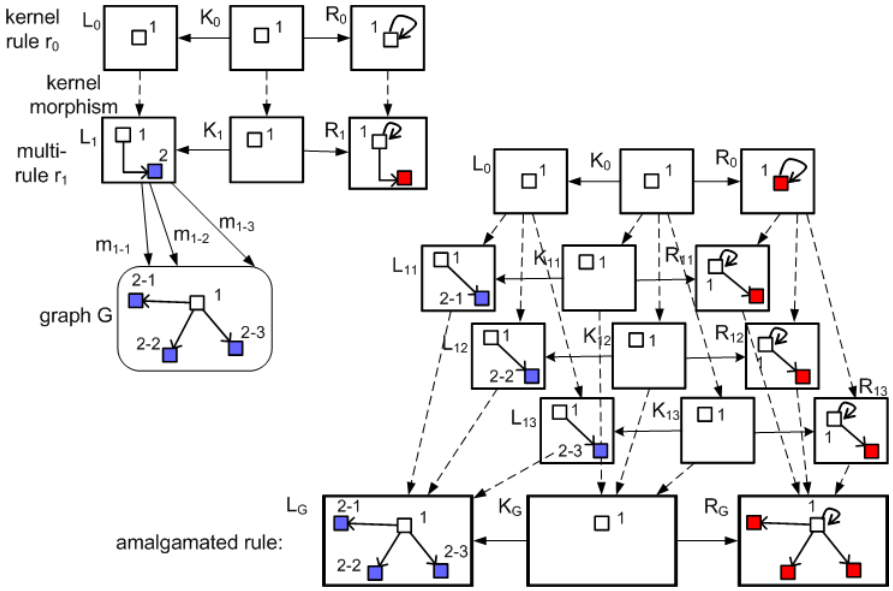
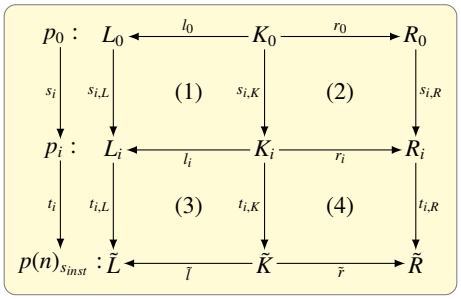


Fig. 6. Construction of an amalgamated rule

Example 2.6 (Construction of amalgamated rule). To illustrate the construction of an amalgamated rule consider Fig. 6 as an example. The basic interaction scheme is given on the left. It consists of a kernel rule r_0 which adds a loop. Moreover, it contains one multi-rule r_1 modeling that object 2 being connected to object 1 is deleted and a new object is created and connected to object 1 which has a loop now. Note that the left-hand part of this kernel morphism has a PO complement (not being depicted). Given graph G, there are obviously three different matches of the multi-rule r_1 to G which overlap in the match of the kernel rule to G. Hence, the multi-rule can be applied three times. Thus, we need three copies of the multi-rule in the interaction scheme instance, all with kernel morphisms from kernel rule r_0. In our example, the interaction scheme instance is shown on the right. Gluing all its multi-rules at their common kernel rule, we get the amalgamated rule with respect to G, shown at the bottom of Fig. 6.

In the following definition, we clarify how to construct an amalgamated rule from a given interaction scheme.

Definition 2.7 (Amalgamated rule). *Given an interaction scheme s and an interaction scheme instance $s_{inst} = (s_i : p_0 \to p_i)_{i=0,\dots,n}$ over s with rules $p_i = (L_i \xleftarrow{l_i} K_i \xrightarrow{r_i} R_i)$ for $i = 0, \dots, n$, then the* amalgamated rule $p(n)_{s_{inst}} = (\tilde{L} \xleftarrow{\tilde{l}} \tilde{K} \xrightarrow{\tilde{r}} \tilde{R})$ *is the colimit over the kernel morphisms of s_{inst} being constructed as stepwise pushouts over $i \geq 1$.*

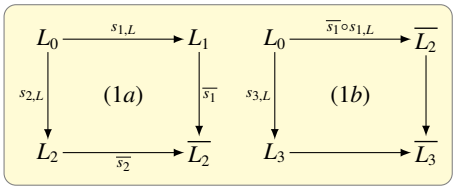


For an interaction scheme s, $Amalg(s)$

denotes the set of all amalgamated rules over all interaction scheme instances over s. Given two amalgamated rules $p, p' \in Amalg(s)$, p is smaller than p', written $p <_h p'$, if there is a non-isomorphic rule morphism $h : p \to p'$. We write $p \leq_h p'$ if there is a rule morphism $h : p \to p'$.

We sketch the idea how to construct the stepwise pushout for $n = 3$ for the L-component given morphisms $s_{1,L}$–$s_{3,L}$:

1. Construct the pushout $(1a)$ of $s_{1,L}$ and $s_{2,L}$.
2. Construct the pushout $(1b)$ of $\overline{s_1} \circ s_{1,L}$ and $s_{3,L}$.
3. $\overline{L_3}$ is the resulting left-hand side $\tilde{L}$ for the amalgamated rule.



This construction is unique, independent of the order of i, and can be done similarly for the K- and R-components. By pushout properties (see [6]), we obtain unique morphisms $\tilde{l}$ and $\tilde{r}$.

Example 2.8 (Amalgamated rule for pushing down an attribute to 3 subclasses). Figure 7 shows an amalgamated rule built up over the interaction scheme consisting of the kernel morphism in Figure 1 and two copies of the multi-rule. It specifies the push down of an attribute to three subclasses.

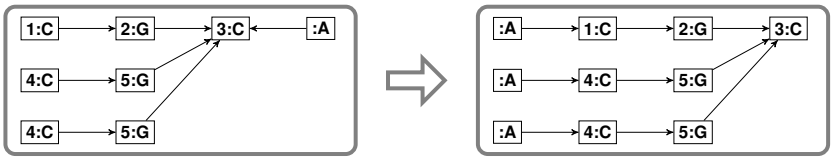
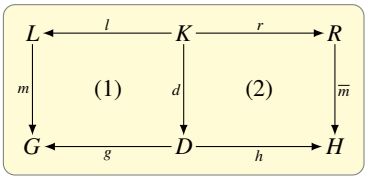


Fig. 7. Amalgamated rule for refactoring "Push Down Attribute"

Definition 2.9 (Transformation). *Given a rule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ and a match $m : L \to G$ of p to a graph G, a* transformation step $t : G \overset{p,m}{\Longrightarrow} H$ *consists of the following diagram where (1) and (2) are pushouts, and $\overline{m}$ is called* co-match*. If p is an amalgamated rule, t is also called* amalgamated transformation step*. Let $der(t) = (G \xleftarrow{g} D \xrightarrow{h} H)$ be the* derivation *of t. An amalgamated transformation $t : G = G_0 \overset{p_1(k_1),m_1}{\Longrightarrow} \dots \overset{p_n(k_n),m_n}{\Longrightarrow} G_n = H$, short $t : G \overset{[p_n]}{\Longrightarrow}_* H$, consists of $n \geq 0$ transformation steps each of which may apply an amalgamated rule $p_i(k_i)$. This means that $[p_n]$ is defined by a list of applied amalgamated rules $(p_1(k_1), \dots, p_n(k_n))$ with $n \geq 0$. Note that $[p_0]$ is the empty list.*



When given an interaction scheme, we want to apply as many multi-rules as often as possible over a certain kernel rule match. This is ensured by maximal matches.

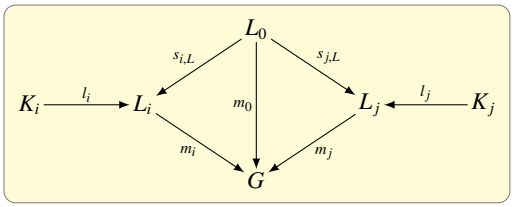
Definition 2.10 (Maximal match). *Given an interaction scheme s, an amalgamated rule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ over s and a graph G, a morphism $m : L \to G$ of p to G is called* match *if there is a PO complement of p and m. A match m is called* maximal *if there is no amalgamated rule $p' = (L' \xleftarrow{l'} K' \xrightarrow{r'} R')$ over s with a match $m' : L' \to G$ such that $p <_t p'$ with $m' \circ t_L = m$ for $t : p \to p'$.*

The derivation of a transformation sequence t is defined by the derived span as in, e.g., [6]. Note that in general, the match of an amalgamated rule does not have to be maximal. However, this match strategy is often intended.

Definition 2.11 (Maximized transformation). *Given a set of interaction schemes S and a transformation sequence $t : G \overset{[p_n]}{\Longrightarrow}_* H$ with $[p_n]$ being a list of applied amalgamated rules $(p_1(k_1), \ldots, p_n(k_n))$, $n \geq 0$. The* maximized *transformation $max(t) : G \overset{max([p_n])}{\Longrightarrow}_* H'$ applies $max([p_n])$ being the list $(p_1(k'_1), \ldots, p_n(k'_n))$ of amalgamated rules with maximal matches only. Hence, $p_x(k'_x) = p_x(k_x)$ if $p_x(k_x)$ has already a maximal match or $p_x(k'_x) > p_x(k_x)$ with $p_x(k'_x)$ having a maximal match, for all $1 \leq x \leq n$.*

If we have a bundle of direct transformations of a graph G, where for each transformation one of the multi-rules is applied, we want to analyze if the amalgamated rule is applicable to G combining all the single transformation steps. These transformations are compatible, i.e. *multi-amalgamable*, if the matches agree on the kernel match, and are independent outside.

Definition 2.12 (Multi-amalgamable). *Given an interaction scheme $s = (s_i : p_0 \to p_i)_{i=0,\ldots,n}$, a bundle of direct transformations steps $(G \overset{p_i,m_i}{\Longrightarrow} G_i)_{i=1,\ldots,n}$ is* multi-amalgamable *over s, if*



- *it has* consistent matches*, i.e., $m_i \circ s_{i,L} = m_j \circ s_{j,L} =: m_0$ for all $i, j = 1, \ldots, n$ and*
- *it has* weakly independent matches*, i.e., $m_i(L_i) \cap m_j(L_j) \subseteq m_0(L_0) \cup (m_i(l_i(K_i)) \cap m_j(l_j(K_j)))$ for all $1 \leq i \neq j \leq n$ which means that the elements in the intersection of the matches m_i and m_j are either preserved by both transformations, or are also matched by m_0.*

If a bundle of direct transformations of a graph G is multi-amalgamable then we can apply the amalgamated rule directly to G leading to a parallel execution of all the changes performed also by the single transformation steps. This is stated by the Multi-Amalgamation Theorem in [8, 9]. This theorem can also be used to decompose an amalgamated rule into a smaller amalgamated transformation and the complement transformation containing all complement rules not yet applied. The following corollary states this result; its proof is given in [16].

Corollary 2.13 (Multi-Amalgamation). *Given a bundle of multi-amalgamable transformations $(G \overset{p_i,m_i}{\Longrightarrow} G_i)_{i=1,\ldots,n}$ over an interaction scheme s and a sub-bundle s' for*

$i = 1, \ldots, k < n$, then there is a transformation $G \overset{p_{s'}, \tilde{m}'}{\Longrightarrow} \overline{H}$ amalgamating the subbundle and a transformation $\overline{H} \overset{\overline{q}}{\Rightarrow} H$ over some rule $\overline{q}$ such that $G \overset{p_{s'}, \tilde{m}'}{\Longrightarrow} \overline{H} \overset{\overline{q}}{\Rightarrow} H$ is a decomposition of $G \overset{p_s, \tilde{m}}{\Longrightarrow} H$.

Note that $\overline{q}$ is constructed as a gluing of the complement rules of $p_{k+1}, \ldots, p_n$.

3 Parallel Independence of Rule Schemes

Two graph transformation steps are parallel independent if one transformation step does not delete any graph item being used by the other one. In this case, both transformation steps can be executed in either order. This is stated by the well-known Church–Rosser-Property [6]. Even if both transformation steps intend to delete a common graph item, this is considered as a dependency since one transformation step cannot be executed anymore after the other has been executed and has deleted that item.

Parallel independent amalgamated graph transformations have already been considered in [8] in the context of bundles of amalgamable transformations, but without maximal matches. In the following, we characterize the parallel and sequential independence of amalgamated transformation steps on the level of interaction schemes.

Definition 3.1 (Parallel independence). *Two transformation steps $G \overset{p,m}{\Longrightarrow} H$ and $G \overset{p',m'}{\Longrightarrow} H'$ with derivations $(G \overset{g}{\longleftarrow} D \overset{h}{\longrightarrow} H)$ and $(G \overset{g'}{\longleftarrow} D' \overset{h'}{\longrightarrow} H')$ are* parallel independent *iff there exist morphisms $ld : L \to D'$ and $ld' : L' \to D$ such that $g' \circ ld = m$ and $g \circ ld' = m'$.*

Two rules p and p' are parallel independent *if all pairs of transformation steps over p and p' are parallel independent.*

Two interaction schemes $s = (s_i : p_0 \to p_i)_{i=0,\ldots,n}$ and $s' = (s'_j : p'_0 \to p'_j)_{j=0,\ldots,n'}$ are parallel independent *if p_i and p'_j are parallel independent for all pairs (i, j) with $0 \leq i \leq n$ and $0 \leq j \leq n'$.*

Example 3.2 (Parallel independent interaction schemes). Considering the interaction schemes in Section 2, they are all parallel independent from the interaction scheme "Delete Class" which just consists of the kernel rule. This rule can only be applied to classes being disconnected from others, hence they cannot be in the match of any other refactoring rule. Any other two interaction schemes, however, can be applied such that they are not parallel independent.

Definition 3.3 (Sequential independence). *Two transformation steps $G \overset{p,m}{\Longrightarrow} H$ and $H \overset{p',m'}{\Longrightarrow} X$ with derivations $(G \overset{g}{\longleftarrow} D \overset{h}{\longrightarrow} H)$ and $(H \overset{h'}{\longleftarrow} D' \overset{x}{\longrightarrow} X)$ are* sequentially independent *iff there exist morphisms $rd : R \to D'$ and $ld' : L' \to D$ such that $h' \circ rd = \overline{m}$ and $h \circ ld' = m'$ with $\overline{m}$ bein the co-match of m.*

Two rules p and p' are sequentially independent *if all pairs of transformation steps over p and p' are sequentially independent.*

Two interaction schemes $s = (s_i : p_0 \to p_i)_{i=0,\ldots,n}$ and $s' = (s'_j : p'_0 \to p'_j)_{j=0,\ldots,n'}$ are sequentially independent *if p_i and p'_j are sequentially independent for all pairs (i, j) with $0 \leq i \leq n$ and $0 \leq j \leq n'$.*

Theorem 3.4 (Independence of interaction schemes). *Two interaction schemes* $s = (s_i : p_0 \to p_i)_{i=0,\dots,n}$ *and* $s' = (s'_j : p'_0 \to p'_j)_{j=0,\dots,n'}$ *are parallel (sequentially) independent iff* p *and* p' *are parallel (sequentially) independent for all pairs of amalgamated rules* p *over* s *and* p' *over* s'.

This result allows us to formulate the Local Church–Rosser property not only for arbitrary, but also for maximal matches of amalgamated transformations. Intuitively, this means that in case of both parallel and sequential independence, the application of one transformation step does not lead to new matches of the other interaction scheme.

Theorem 3.5 (Church–Rosser property for interaction schemes). *Given two interaction schemes* $s = (s_i : p_0 \to p_i)_{i=0,\dots,n}$ *and* $s' = (s'_j : p'_0 \to p'_j)_{j=0,\dots,n'}$, *the following statements hold:*

1. *If* s *and* s' *are parallel independent, then any two amalgamated transformations* $G \overset{p,m}{\Longrightarrow} H$ *and* $G \overset{p',m'}{\Longrightarrow} H'$ *applying amalgamated rules* p *over* s *and* p' *over* s' *can be completed by amalgamated transformations* $H \overset{p',\bar{m}'}{\Longrightarrow} X$ *and* $H' \overset{p,\bar{m}}{\Longrightarrow} X$.
2. *If* s *and* s' *are parallel and sequentially independent, then any two amalgamated transformations* $G \overset{p,m}{\Longrightarrow} H$ *and* $G \overset{p',m'}{\Longrightarrow} H'$ *applying amalgamated rules* p *over* s *and* p' *over* s' *at maximal matches* m *and* m' *can be completed by amalgamated transformations* $H \overset{p',\bar{m}'}{\Longrightarrow} X$ *and* $H' \overset{p,\bar{m}}{\Longrightarrow} X$ *at maximal matches* $\bar{m}$ *and* $\bar{m}'$.

The proofs of both theorems can be found in [16].

4 Conflict Analysis for Rule Schemes

The critical pair analysis (CPA) is a well-known technique to analyze potential conflicts and dependencies of transformation systems. It has first been introduced for term rewriting and later generalized to graph transformation [13, 6]. A critical pair describes a minimal conflicting situation that may occur in the transformation system. It is well-known that if all critical pairs can be shown to be strictly confluent, the transformation system is locally confluent. A transformation system is *locally confluent* if each pair of direct transformation steps can be resolved by arbitrary many steps to a common graph. The notion of *strict confluence* means that the jointly preserved part of a critical pair is also preserved by its resolution [14]. Up to now, this theory has been shown for simple rules. In the following, we extend it to interaction schemes such that the CPA can also be used for amalgamated rules. The main problem we have to deal with is that, in general, there is an infinite set of critical pairs for all amalgamated rules over an interaction scheme.

Definition 4.1 (Critical pair). *A* critical pair*, short CP, consists of two transformation steps* $t_i : G \overset{p_i,m_i}{\Longrightarrow} H_i$ *applying rules* $p_i = (L_i \overset{l_i}{\longleftarrow} K_i \overset{r_i}{\longrightarrow} R_i)$ *at matches* m_i *for* $i \in \{1, 2\}$ *such that* G *is minimal, i.e.,* m_1 *and* m_2 *are jointly surjective. Given two interaction schemes* s_1 *and* s_2, $CP(s_1, s_2)$ *denotes the set of all critical pairs over transformation steps* t_1 *and* t_2 *as above, applying amalgamated rules* $p_1 \in Amalg(s_1)$ *and* $p_2 \in Amalg(s_2)$. *Given a set* S *of interaction schemes,* $CP(S) = \bigcup_{s_1,s_2 \in S} CP(s_1, s_2)$.

Example 4.2 (Critical pair). Figure 8 shows a critical pair applying the kernel rule PDA(0) of the refactoring "Push down Attribute" and the multi-rule RIWD(1) of the refactoring "Replace Inheritance With Delegation". Since PDA(0) deletes attribute 4:A while RIWD(1) is reading and preserving it, this critical pair reports a delete-use-conflict. It can be resolved by applying the refactoring "Pull Up Attribute" taking back the previous refactoring and then applying RIWD(1) as on the right. Hence, the common graph will be isomorphic to $H2$.

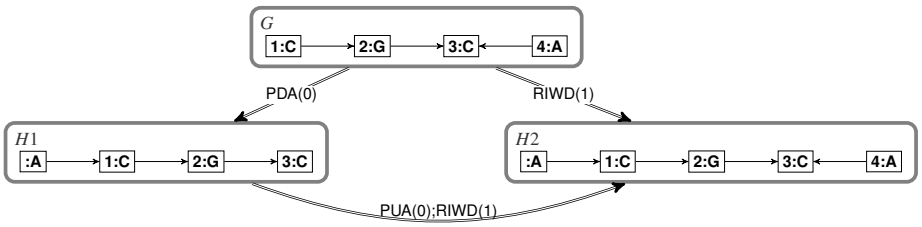


Fig. 8. Critical pair between "Push Down Attribute" and "Replace Inheritance With Delegation"

Corollary 4.3 (Confluence of interaction schemes). *A set S of interaction schemes is locally confluent if, for all $s, s' \in S$ and for all rule pairs (al, ar) with $al \in Amalg(s)$ and $ar \in Amalg(s')$, all critical pairs over (al, ar) are strictly confluent.*

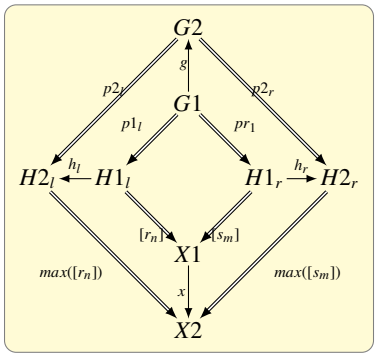
This corollary directly follows from the Local Confluence Theorem and Critical Pair Lemma (see, e.g., Theorem 3.34 in [6]) since we can consider amalgamated rules as normal rules.

Although this corollary yields a result on the confluence of interaction schemes, it can hardly be used to check confluence since the set of amalgamated rules over an interaction scheme is infinite in general. Hence, infinite many critical pairs have to be checked in general. The key idea for reducing the set of critical pairs is to take out those critical pairs that do not specify any new conflicting situation. We continue to develop a characterization for critical pairs being redundant in that sense.

Definition 4.4 (Extraction of critical pairs). *Given a set $CP(s_l, s_r)$ of critical pairs with $cp_1 = (G1 \overset{p1_l, m1_l}{\Longrightarrow} H1_l, G1 \overset{p1_r, m1_r}{\Longrightarrow} H1_r)$ and $cp_2 = (G2 \overset{p2_l, m2_l}{\Longrightarrow} H2_l, G2 \overset{p2_r, m2_r}{\Longrightarrow} H2_r) \in CP(s_l, s_r)$ being two critical pairs with $p1_l, p2_l \in Amalg(s_l)$, $p1_r, p2_r \in Amalg(s_r)$, $p2_l \geq p1_l$, and $p2_r \geq p1_r$. The critical pair cp_2 is* larger than *cp_1, short $cp_2 > cp_1$, if there are injective graph morphisms $g : G1 \to G2, d_l : D1_l \to D2_l, h_l : H1_l \to H2_l, d_r : D1_r \to D2_r$, and $h_r : H1_r \to H2_r$ such that corresponding diagrams commute and cp_2 is a proper extension of cp_1, i.e., at least one of morphisms g, h_l and h_r is not surjective. We can also say that cp_1 is* smaller than *cp_2. If $g(m1_l(L1_l - l1_l(K1_l)) \cup m1_r(L1_r - l1_r(K1_r))) \subseteq m2_l(L2_l - l2_l(K2_l)) \cup m2_r(L2_r - l2_r(K2_r))$ holds in addition, cp_1 is called an* extraction *of cp_2.*

In the following, we characterize under which conditions a critical pair cp_1 is considered to be restricted w.r.t. another critical pair cp_2. Note that the restriction of critical pairs is more than cutting away unnecessary context. In general, both critical pairs coincide w.r.t. the interaction schemes applied but differ in the size of the actual amalgamated rules. This applies to the resolving interaction schemes as well.

Definition 4.5 (Restricted critical pair). *Consider a set of interaction schemes S and critical pairs $cp_1 = (G1 \xRightarrow{p1_l, m1_l} H1_l, G1 \xRightarrow{p1_r, m1_r} H1_r)$ and $cp_2 = (G2 \xRightarrow{p2_l, m2_l} H2_l, G2 \xRightarrow{p2_r, m2_r} H2_r)$ applying rules of Amalg(S) such that cp_1 is an extraction of cp_2 and cp_1 is strictly confluent; this means (among others) that there are transformations $t1_l : H1_l \xRightarrow{[r_n]} X1$ and $t1_r : H1_r \xRightarrow{[s_m]} X1$. The critical pair cp_1 is* more restricted *than cp_2 if there are transformations $t_l : H2_l \xRightarrow{max([r_n])} X2$ and $t_r : H2_r \xRightarrow{max([s_m])} X2$ and an injective morphism $x : X1 \to X2$ compatible with the derivations of t_l and t_r. We also say that cp_2 is* redundant *wrt. cp_1. Given the set CP of critical pairs over S, $Res(CP) \subseteq CP$ contains all critical pairs not being redundant of another one of CP.*



Example 4.6 (Restricted critical pairs). Figures 9 and 10 show two critical pairs cp_2 and cp_3 both being strictly confluent. We will consider these critical pairs first and then argue why cp_2 is more restricted than cp_3.

In Figure 9, the multi-rules of the refactorings "Push Down Attribute" and "Replace Inheritance With Delegation" are applied such that they overlap in conflicting elements: The attribute 4:A is deleted by PDA(1) and preserved by RIWD(1), and the generalization 6:G is preserved by PDA(1) and deleted by RIWD(1). Both refactorings can be applied one after the other such that these conflicts can be resolved. This critical pair is called cp_2.

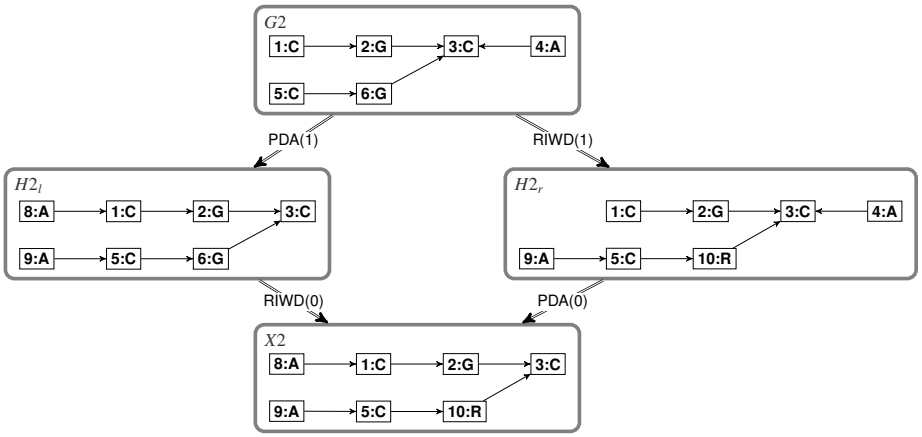


Fig. 9. Critical pair between "Push Down Attribute" and "Replace Inheritance With Delegation"

Figure 10 shows a similar critical pair cp_3 where the same multi-rule PDA(1) is applied on the left but a slightly larger rule on the right. It is an amalgamated rule applying the multi-rule of the refactoring "Replace Inheritance With Delegation" twice. The same kinds of conflicts are reported here but this time two attributes are in the

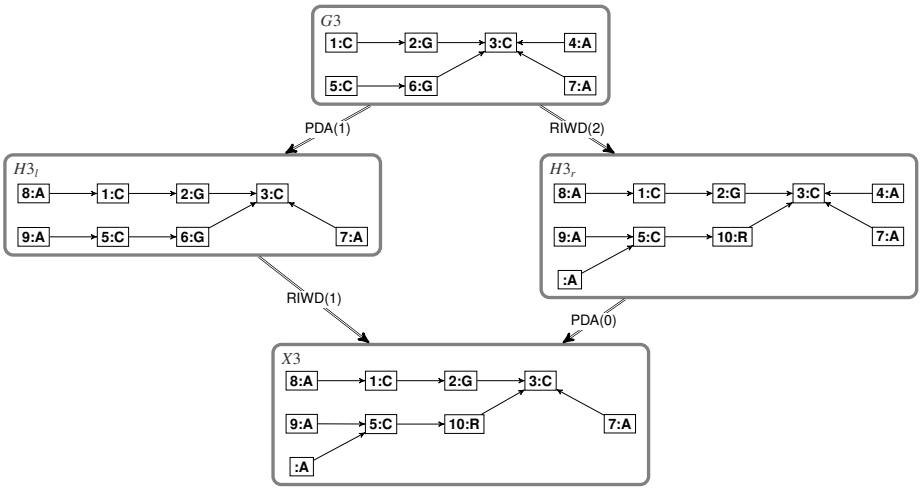


Fig. 10. Another critical pair between "Push Down Attribute" and "Replace Inheritance With Delegation"

super class 3:C. The resolution of this critical pairs resembles very much the one in cp_2. The only difference is that the multi-rule RIWD(1) is applied on the left, instead of the kernel rule RIWD(0).

The critical pair cp_2 is an extraction of cp_3 since cp_3 has larger graphs with compatible embeddings of corresponding graphs $G2$ and $G3$, $H2_l$ and $H2_r$, and $H3_l$ and $H3_r$ as well as $RIWD(1) < RIWD(2)$. Moreover, all elements being deleted from $G2$ have corresponding elements that are deleted from $G3$. Considering the conflict resolutions in both critical pairs, the one in cp_3 is the maximized version of the one in cp_2. Furthermore, there is an injective morphism from $X2$ to $X3$ being compatible with the corresponding derivations. Hence, cp_2 is more restricted than cp_3. It is straight forward to show for all critical pairs cp' distinguishing from cp_2 just by the number of attributes at super class 3:C that cp_2 is more restricted than cp'.

In the following, we show that the reduction of critical pair sets is sound, i.e., that strict confluence of the reduced set of critical pairs still induces strict confluence of the whole set.

Theorem 4.7 (Reduction of CP set). *Given a set S of interaction schemes and two critical pairs $cp_1, cp_2 \in CP(S)$ such that cp_1 is more restricted than cp_2, the following holds: If all critical pairs of $CP(S) - \{cp_2\}$ are strictly confluent then all critical pairs of $CP(S)$ are strictly confluent.*

Proof idea: Since cp_1 is confluent and more restricted than cp_2, it is straight forward to show that cp_2 is confluent. Let $H1_l$ be the embedded result graph after applying pl_1 (see diagram in Definition 4.5). Since the match of the complement rule $\overline{pl_2}$ is allowed to overlap with the embedding $h_l : H1_l \to H2_l$ in preserved items only, strict confluence of cp_2 can be shown based on the strict confluence of cp_1 as well as using pushout and pullback properties.

Proposition 4.8 (Transitive restriction of critical pairs). *Given a set $CP(S)$ of critical pairs it holds: If $cp_1 \in CP(S)$ is more restricted than $cp_2 \in CP(S)$ and cp_2 is more restricted than $cp_3 \in CP(S)$ then cp_1 is more restricted than cp_3 as well.*

The proof is straight forward along the definition of restricted critical pairs.

The following example shows that conflict resolutions for smaller rules over two selected interaction schemes cannot always be transfered to larger rules of the same schemes, even if the same conflicts are reported. The resolution is dependent on the available context. In some cases, the context is too large to apply a rule (violating the dangling condition) and in other cases, the context is not large enough to apply rules.

Example 4.9 (Non-redundant critical pairs). In Figure 11, a critical pair between the multi-rules of "Remove All Inheritances" and "Replace Inheritance With Delegation" is depicted. Both delete the only generalization relation. This conflict is resolved by deleting the isolated class on the left while on the right, the separate class is inlined into the referred class (taking back the previous refactoring). This resolution works here since class 1:C is the only subclass.

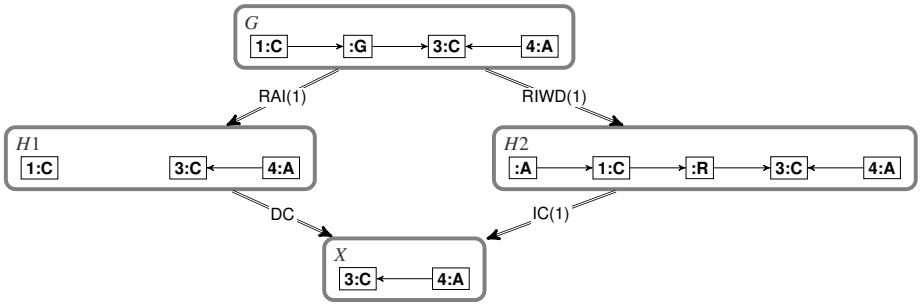


Fig. 11. Critical pair between "Remove All Inheritances" and "Replace Inheritance With Delegation"

Figure 12 shows a different critical pair between an amalgamated rule of "Remove All Inheritances" and the multi-rule of "Replace Inheritance With Delegation". Since graphs $H1$ and $H2$ of this critical pair have more context than in the previous example, the reported conflict cannot be resolved as before. While "Delete Class" has to be applied twice on the left, "Inline Class" is again applied on the right, together with "Delete Empty Subclass". This interaction scheme is not applicable in a too small context (as in the critical pair above).

This example points to a general problem that can occur in conflict resolution: The rule DC is applied dependent on how many subclasses are considered, i.e., the resolution is performed sequentially and larger critical pairs cannot become redundant. If DC were an interaction scheme with an empty kernel rule and the original rule as multi-rule, i.e., a more parallel interaction scheme, this problem can be solved.

The example shows that it is not enough to consider critical pairs over kernel and multi-rules, but smaller amalgamated rules have to be considered as well. Larger amalgamated rules, however, have recurring parts (multi-rule copies) that do not lead to new

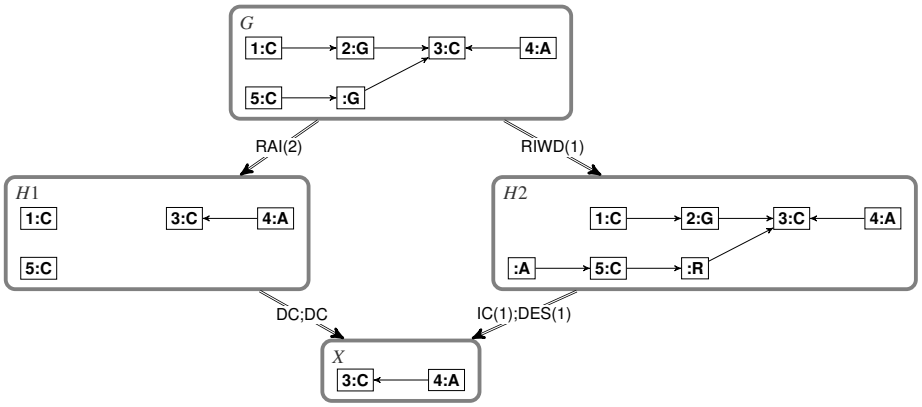


Fig. 12. Critical pair between "Remove All Inheritances" and "Replace Inheritance With Delegation"

kinds of resolutions, i.e., lead to redundant critical pairs. The following theorem states that the set of non-redundant critical pairs is in fact finite. It may happen that conflict resolutions result in applying interaction schemes in loops. In those cases, we consider more parallel interaction schemes where resolutions are performed in parallel.

Theorem 4.10 (Finite set of restricted critical pairs). *Given a finite set S of interaction schemes consisting of finite rules only. Then the set $Res(CP) \subseteq CP(S_{par})$ is finite for an interaction scheme S_{par} being more parallel than S (see Definition 2.4).*

Proof idea: We consider a pair of interaction schemes $s, s' \in S$ and the set $CP(s, s')$ of all their critical pairs. We show that the set $Res(CP(s, s'))$ of all critical pairs without redundant ones is finite. The main idea is that there are numbers c and d such that the following condition holds: Let R be the set of all kernel and multi-rules of all interaction schemes in S. Consider the set M of all partial matches of all rules in R w.r.t. all critical pairs (t_1, t_2) over amalgamated rules with at most c and d multi-rule instantiations for t_1 and t_2, respectively. Then there is no critical pair that leads to a match being non-equivalent to any match in M. c and d exist since copies of complement rules are applied parallel independently and hence, derive isomorphic graph parts in $H2_l$ and $H2_r$. We have to change to S_{par} if S contains interaction schemes that are applied in loops to resolve critical pairs, but still obtain the same result.

Full proofs of all the theorems above can be found in [16].

5 Related Work and Conclusion

Multi-objects and other variants of matching graph parts as often as possible have been considered in several graph transformation approaches, in tool environments such as PROGRES [15] and Fujaba [1] as well as in conceptual approaches by Grönmo [10] and Drewes et.al. [5]. Amalgamated graph transformation has been used to specify activities with some variabilities [2, 3, 7, 12]. Although being applied in different contexts and formalized in [8, 9], the critical pair analysis has not yet been extended to this kind of

graph transformation. It turns out that the CPA can be reused by considering only a finite set of critical pairs of smaller amalgamated rules to decide the local confluence of the whole transformation system. Future work is needed to develop an efficient algorithm for enumerating all non-redundant critical pairs and for evaluating the extended CPA in practice. Furthermore, the extension to a more sophisticated graph transformation approach with types, attributes, and application conditions is worthwhile to consider.

Acknowledgment: We thank Yngve Lamo and Kristopher Born for their valuable comments to this paper.

References

1. The Fujaba tool suite. `www.fujaba.de`
2. Biermann, E., Ehrig, H., Ermel, C., Golas, U., Taentzer, G.: Parallel Independence of Amalgamated Graph Transformations Applied to Model Transformation. In: Graph Transformations and Model-Driven Engineering. LNCS, vol. 5765, pp. 121–140. Springer (2010)
3. Biermann, E., Ermel, C., Taentzer, G.: Lifting parallel graph transformation concepts to model transformation based on the eclipse modeling framework. ECEASST 26 (2010)
4. Biermann, E., Ermel, C., Taentzer, G.: Formal foundation of consistent emf model transformations by algebraic graph transformation. Software & Systems Modeling 11(2), 227–250 (2012)
5. Drewes, F., Hoffmann, B., Janssens, D., Minas, M.: Adaptive star grammars and their languages. Theor. Comput. Sci. 411(34-36), 3090–3109 (2010)
6. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science. An EATCS Series, Springer (2006)
7. Golas, U., Biermann, E., Ehrig, H., Ermel, C.: A Visual Interpreter Semantics for Statecharts Based on Amalgamated Graph Transformation. ECEASST 39, 1–24 (2011)
8. Golas, U.: Analysis and correctness of algebraic graph and model transformations. Ph.D. thesis, Berlin Institute of Technology (2011)
9. Golas, U., Habel, A., Ehrig, H.: Multi-amalgamation of rules with application conditions in $\mathcal{M}$-adhesive categories. Mathematical Structures in Computer Science 24(4) (2014)
10. Grönmo, R., Krogdahl, S., Möller-Pedersen, B.: A collection operator for graph transformation. In: Proc. of ICMT 2009. LNCS, vol. 5563, pp. 67–82. Springer (2009)
11. Habel, A., Pennemann, K.H.: Correctness of high-level transformation systems relative to nested conditions. Mathematical Structures in Computer Science 19(2), 245–296 (2009)
12. Mantz, F., Taentzer, G., Lamo, Y., Wolter, U.: Co-evolving meta-models and their instance models: A formal approach based on graph transformation. Sci. Comput. Program. 104, 2–43 (2015)
13. Plump, D.: Critical Pairs in Term Graph Rewriting. In: Mathematical Foundations of Computer Science. LNCS, vol. 841, pp. 556–566. Springer (1994)
14. Plump, D.: On termination of graph rewriting. In: Graph-Theoretic Concepts in Computer Science, 21st Int. Workshop, WG '95. LNCS, vol. 1017, pp. 88–100. Springer (1995)
15. Schürr, A., Winter, A., Zündorf, A.: The PROGRES approach: Language and environment. In: Handbook of Graph Grammars and Computing by Graph Transformation, pp. 487–550. World Scientific (1999)
16. Taentzer, G., Golas, U.: Towards Local Confluence Analysis for Amalgamated Graph Transformation: Long Version. Tech. Rep. 15-29, Zuse Institute Berlin (2015), `https://opus4.kobv.de/opus4-zib/frontdoor/index/index/docId/5494`