Granularity of Conflicts and Dependencies in Graph Transformation Systems

Kristopher Born[1], Leen Lambers[2], Daniel Strüber[3], Gabriele Taentzer[1]

[1] Philipps-Universität Marburg, Germany,
{born, taentzer}@informatik.uni-marburg.de
[2] Hasso-Plattner-Institut, Potsdam, Germany,
leen.lambers@hpi.de
[3] Universität Koblenz-Landau, Germany,
strueber@uni-koblenz.de

Abstract. Conflict and dependency analysis (CDA) is a static analysis for the detection of conflicting and dependent rule applications in a graph transformation system. The state-of-the-art CDA technique, critical pair analysis, provides its users the benefits of completeness, i.e., its output contains a precise representation of each potential conflict and dependency in a minimal context, called *critical pair*. Yet, user feedback has shown that critical pairs can be hard to understand; users are interested in core information about conflicts and dependencies occurring in various combinations. In this paper, we investigate the granularity of conflicts and dependencies in graph transformation systems. We introduce a variety of new concepts on different granularity levels: We start with *conflict atoms*, representing individual graph elements as smallest building bricks that may cause a conflict. We show that each conflict atom can be extended to at least one *conflict reason* and, conversely, each conflict reason is covered by atoms. Moreover, we relate conflict atoms to *minimal conflict reasons*, representing smallest element sets to be overlapped in order to obtain a pair of conflicting transformations. We show how conflict reasons are related to critical pairs. Finally, we introduce dual concepts for dependency analysis. As we discuss in a running example, our concepts pave the way for an improved CDA technique.

1 Introduction

Graph transformation systems (GTSs) are a fundamental modeling concept with applications in a wide range of domains, including software engineering, mechanical engineering, and chemistry. A GTS comprises a set of transformation rules that are applied in coordination to achieve a higher-level goal. The order of rule applications can either be specified explicitly using a control flow mechanism, or it is given implicitly by causal dependencies of rule applications. In the latter case, conflicts and dependencies affect the control flow. For instance, a rule may delete an element whose existence is required by another rule to modify the graph.

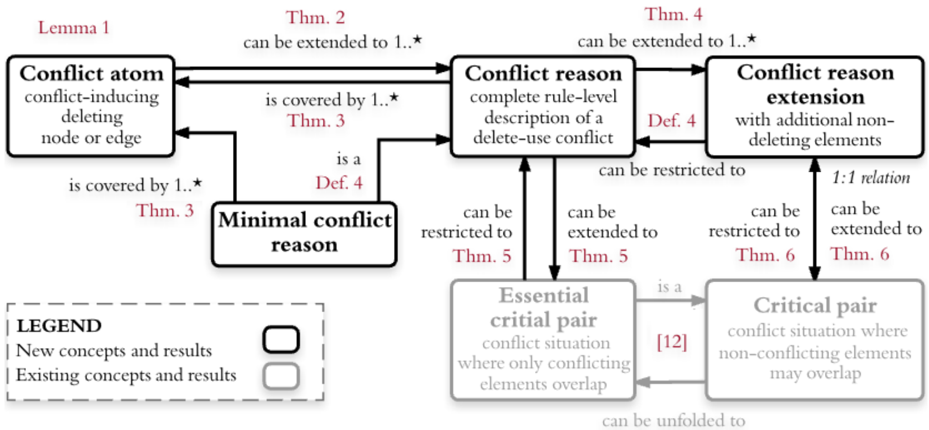


Fig. 1. Inter-relations of new and existing conflict notions

To better understand the implicit control flow of a GTS, one needs to analyze the potential conflicts and dependencies of its rule applications. Conflict and dependency analysis (CDA) is a static analysis for the detection of such conflicts and dependencies. An important CDA technique is *critical pair analysis* [1, 2], which has been used in the literature to detect conflicting functional requirements [3], feature interactions [4], conflicting and dependent change operations for process models [5], causal dependencies of aspects in aspect modeling [6], potential conflicts and dependencies between refactorings [7, 8], and to validate service-oriented architectures [9].

In these applications, there are generally two possible usage scenarios for CDA: First, the user may start with a list of expected conflicts and dependencies that are supposed to occur. CDA is then used to determine if the expected conflicts and dependencies in fact arise, and/or if there are any unexpected conflicts and dependencies. Violations of expectations signify potential errors in the rule specifications, and can be used for debugging [10]. Second, the user may want to improve their transformation system to reduce conflicts and dependencies, so that rules can be applied independently, e.g., to enable a collaborative modeling process based on edit operation rules [11]. In this case, conflicts and dependencies reported by CDA can be used to identify required modifications. In both cases, users need to inspect conflicts or dependencies to pinpoint their root causes.

To support users during this task, in this work, we lay the basis for a refined CDA technique, distinguishing a variety of new concepts to describe conflicts and dependencies between rules. Our investigation is guided by the notion of *granularity*, and, building on the existing theory for algebraic graph transformation, focuses on *delete-use*-conflicts. We introduce a variety of new concepts and their relations as summarized in Fig. 1. First, we introduce *conflict atoms*, i.e., single graph elements causing conflicts, to represent smallest entities of conflicts. Each conflict atom can be embedded into the *conflict reason* of a pair of conflicting rules, while each such conflict reason is fully covered by conflict atoms. A conflict reason comprises all elements being deleted by the first and required by the second rule of the considered rule pair. Conflict reasons correspond to essential critical pairs as introduced in previous work [12]. A special type of conflict reasons are *minimal conflict reasons*, representing conflicting graphs and embeddings that are minimal in the sense that they comprise smallest sets of elements required to yield a valid pair of conflicting transformations. Fourth and finally, conflict reasons can be augmented to conflict reason extensions, which have a one-to-one relationship with the notion of critical pairs [1]. Conflict atoms and minimal conflict reasons are more coarse-grained in the sense that they generally represent a larger number of potential conflicts while abstracting away many details of these conflicts, whereas conflict reasons and conflict reason extensions are more fine-grained since they describe conflicts more precisely.

With this contribution, we aim to improve on the state-of-the-art CDA technique, critical pair analysis (CPA) [1, 2], by offering improved support for cases where the CPA results did not match the user expectations. In CPA, all potential conflicts and dependencies that can occur when applying two rules are displayed in a minimal context. Confidence in CPA is established by positive fundamental results: via the Completeness Theorem, there exists a critical pair for each conflict, representing this conflict in a minimal context. However, experiences with the CPA indicate two drawbacks: (i) understanding the identified critical pairs can be a challenging task since they display too much information (i.e., they are too fine-grained), (ii) calculating the results can be computationally expensive. Our investigation provides the basis for a solution to compute and report potential conflicts on a level of detail being suitable for the task at hand.

In this paper, we investigate the granularity of conflict and dependencies in GTSs. Specifically, we make three contributions.

- We present a conceptual consideration of conflicts in GTSs, based on the notion of granularity, and focusing on delete-use-conflicts.
- We introduce a variety of formal results for interrelating the new concepts with each other and with the existing concepts. In particular, we relate the new concepts to the well-known conflict concepts of essential and regular critical pairs.
- We discuss how these concepts and results can be transferred to dependencies in a straight-forward manner. In particular, we introduce dependency atoms and reasons, the dual concepts to those introduced for conflict analysis.

The rest of this paper is structured as follows: In Sect. 2, we recall graph transformation concepts and conflict notions from the literature. In Sect. 3, we present the new concepts and formal results. Finally, we compare with related work and conclude in Section 4.

2 Preliminaries

As a prerequisite for our new analysis of conflicts and dependencies, we recall the double-pushout approach to graph transformation as presented in [2]. Furthermore, we reconsider two notions of conflicting transformation and their equivalence as shown in [12].

2.1 Graph Transformation: Double-Pushout Approach

Throughout this paper we consider graphs and graph morphisms as presented in [2]; since most of the definitions and results are given in a category-theoretical way, the extension to e.g. typed, attributed graphs [2] is prepared, but up to future work.

Graph transformation is the rule-based modification of graphs. A *rule* mainly consists of two graphs: L is the left-hand side (LHS) of the rule representing a pattern that has to be found to apply the rule. After the rule application, a pattern equal to R, the right-hand side (RHS), has been created. The intersection $K = L \cap R$ is the graph part that is not changed, the graph part that is to be deleted is defined by $L \setminus (L \cap R)$, while $R \setminus (L \cap R)$ defines the graph part to be created. Throughout this paper we consider a graph transformation system just as a set of rules.

A *graph transformation step* $G \overset{m,r}{\Longrightarrow} H$ between two graphs G and H is defined by first finding a graph morphism[1] m of the LHS L of rule r into G such that m is injective, and second by constructing H in two passes: (1) build $D := G \setminus m(L \setminus K))$, i.e., erase all graph elements that are to be deleted; (2) construct $H := D \cup m'(R \setminus K)$ such that a new copy of all graph elements that are to be created is added. It has been shown for graphs and graph transformations that r is applicable at m iff m fulfills the *gluing condition* [2]. In that case, m is called a *match*. For injective morphisms as we use them here, the gluing condition reduces to the *dangling condition*. It is satisfied if all adjacent graph edges of a graph node to be deleted are deleted as well, such that D becomes a graph. Injective matches are usually sufficient in applications and w.r.t. our work here, they allow to explain constructions much easier than for general matches.

Definition 1 (Rule and transformation). *A* rule r *is defined by* $r = (L \hookleftarrow K \hookrightarrow R)$ *with* $L, K,$ *and* R *being graphs connected by two graph inclusions. A* (direct) transformation $G \overset{m,r}{\Longrightarrow} H$ *which applies rule* r *to a graph* G *consists of two pushouts as depicted below. Morphism* $m : L \to G$ *is injective and is called* match*. Rule* r *is* applicable *at match* m *if there exists a graph* D *such that* $(PO1)$ *is a pushout.*

$$
\begin{tikzcd}
L \arrow[d, "m"'] \arrow[dr, phantom, "(PO1)"] & K \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "(PO2)"] & R \arrow[d, "m'"] \\
G & D \arrow[l] \arrow[r] & H
\end{tikzcd}
$$

Example 1. Refactoring is a generally acknowledged technique to improve the design of an object-oriented system [13]. To achieve a larger improvement there is typically a sequence of refactorings required. Due to implicit conflicts and

[1] A morphism between two graphs consists of two mappings between their nodes and edges being both structure-preserving w.r.t. source and target functions. Note that we denote inclusions by $\hookrightarrow$ and all other morphisms by $\to$.

dependencies that may occur between refactorings, it is not always easy for developers to determine which refactorings to use and in which order to apply them. To this aim, CDA can support the developer in finding out if there are conflicts or dependencies at all and, if this is the case, in understanding them.

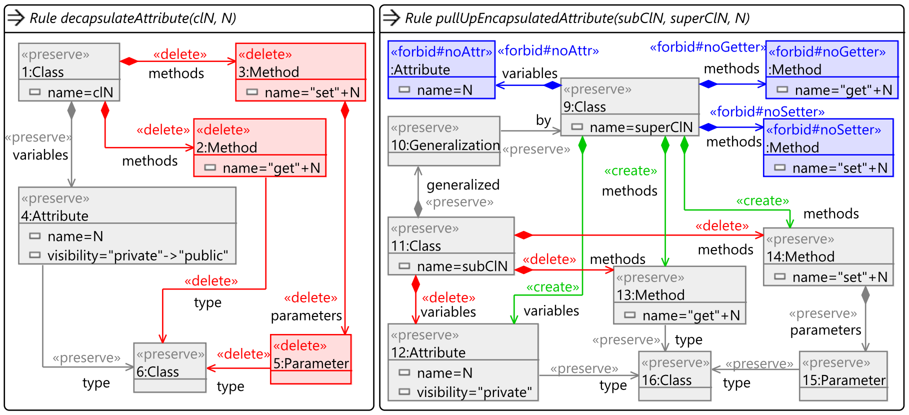


Fig. 2. Refactoring rules *decapsulateAttribute* and *pullUpEncapsulatedAttribute*.

Assuming graphs that model the class design of software systems, we consider Fig. 2 for two class model refactorings being specified as graph-based transformation rules. Rules are depicted in an integrated form where annotations specify which graph elements are deleted, preserved, and created. While the preserved and deleted elements form the LHS of a rule, the preserved and created elements form its RHS. Moreover, negative application conditions specify graph elements that are forbidden when applying a rule. Rule *decapsulateAttribute* removes getter and setter methods for a given attribute, thus inverting the well-known *encapsulate attribute* refactoring. Rule *pullUpEncapsulatedAttribute* takes an attribute with its getter and setter methods and moves them to a superclass if there are not already equally named elements.

2.2 Conflicting Transformations

In this subsection, we recall the essence of conflicting transformations. We concentrate on delete-use conflicts which means that the first rule application deletes graph items that are used by the second rule application. In the literature, there are two different definitions for delete-use conflicts. We recall these definitions and a theorem which shows the equivalence between these two.

The first definition [2] of a delete-use conflict states that the match for the second transformation cannot be found anymore after applying the first transformation. Note that we do not consider delete-use conflicts of the second transformation on the first one explicitly. To get also those ones, we simply consider the inverse pair of transformations.

Definition 2 (Delete-use conflict). *Given a pair of direct transformations* $(t_1, t_2) = (H_1 \overset{m_1,r_1}{\Longleftarrow} G \overset{m_2,r_2}{\Longrightarrow} H_2)$ *applying rules* $r_1 : L_1 \overset{le_1}{\hookleftarrow} K_1 \overset{ri_1}{\hookrightarrow} R_1$ *and*

$r_2 : L_2 \overset{le_2}{\hookleftarrow} K_2 \overset{ri_2}{\hookrightarrow} R_2$ *as depicted below. Transformation* t_1 *causes a* delete-use conflict *on transformation* t_2 *if there does not exist a morphism* $x : L_2 \to D_1$ *such that* $g_1 \circ x = m_2$*.*

$$
\begin{tikzcd}
R_1 & K_1 \arrow[l, "ri_1"'] \arrow[r, "le_1"] \arrow[d, "d_1"'] & L_1 \arrow[dr, "m_1"] & & L_2 \arrow[dl, "m_2"'] \arrow[dlll, dashed, "x"'] & K_2 \arrow[l, "le_2"'] \arrow[r, "ri_2"] \arrow[d, "d_2"'] & R_2 \arrow[d, "m'_2"] \\
H_1 \arrow[from=u, "m'_1"'] & D_1 \arrow[l, "h_1"] \arrow[rr, "g_1"'] & & G & & D_2 \arrow[ll, "g_2"] \arrow[r, "h_2"'] & H_2
\end{tikzcd}
$$

In the following, we consider an alternative characterization for a transformation to cause a delete-use conflict on another one (as introduced in [12]). It states that at least one deleted element of the first transformation is overlapped with some used element of the second transformation. This overlap is formally expressed by a span of graph morphisms between the minimal graph C_1, containing all elements to be deleted by the first rule, and the LHS of the second rule (Fig. 3). In particular, we use an initial pushout construction [2] over the left-hand side morphism of the rule to compute the *boundary graph* B_1 consisting of all nodes needed to make $L_1 \setminus K_1$ a graph and the *context graph* $C_1 := L_1 \setminus (K_1 \setminus B_1)$. We say that the nodes in B_1 are *boundary nodes.* The equivalence of these two conflict notions is recalled in the following theorem.

Theorem 1 (Delete-use conflict characterization). *Given a pair of transformations* $(t_1, t_2) = (H_1 \overset{m_1,r_1}{\Longleftarrow} G \overset{m_2,r_2}{\Longrightarrow} H_2)$ *via rules* $r_1 : L_1 \overset{le_1}{\hookleftarrow} K_1 \overset{ri_1}{\hookrightarrow} R_1$ *and* $r_2 : L_2 \overset{le_2}{\hookleftarrow} K_2 \overset{ri_2}{\hookrightarrow} R_2$*, the initial pushout (1) for* $K_1 \overset{le_1}{\hookrightarrow} L_1$*, and the pullback (2) of* $(m_1 \circ c_1, m_2)$ *in Fig. 2 yielding the span* $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\to} L_2$*, then the following equivalence holds:* t_1 *causes a delete-use conflict on* t_2 *according to Def. 2 iff* $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\to} L_2$ *satisfies the* conflict condition *i.e. there does not exist any morphism* $x : S_1 \to B_1$ *such that* $b_1 \circ x = o_1$*.*

$$
\begin{tikzcd}
 & & & S_1 \arrow[dl, "o_1"'] \arrow[ddr, "q_{12}"] \arrow[dll, dashed, "x"'] & & & \\
 & B_1 \arrow[r, "b_1"] \arrow[d] \arrow[dr, phantom, "(1)"] & C_1 \arrow[d, "c_1"] & & & & \\
R_1 & K_1 \arrow[l, "ri_1"'] \arrow[r, "le_1"] \arrow[d, "d_1"'] & L_1 \arrow[dr, "m_1"'] \arrow[rr, phantom, "(2)"] & & L_2 \arrow[dl, "m_2"] & K_2 \arrow[l, "le_2"'] \arrow[r, "ri_2"] \arrow[d, "d_2"'] & R_2 \arrow[d, "m'_2"] \\
H_1 \arrow[from=u, "m'_1"'] & D_1 \arrow[l, "h_1"] \arrow[rr, "g_1"'] & & G & & D_2 \arrow[ll, "g_2"] \arrow[r, "h_2"'] & H_2
\end{tikzcd}
$$

Fig. 3. Delete-use conflict characterization for transformations

In the rest of the paper we merely consider delete-use conflicts such that in the following we abbreviate *delete-use conflict* with *conflict*.

3 The Granularity of Conflicts and Dependencies

So far, a conflict between two transformations has always been considered as a whole. In the following, we investigate new notions of conflicting rules presenting them on different levels of granularity. Our intention is the possibility to gradually introduce users to conflicts. Starting with a coarse-grained conflict description in the form of conflict atoms, more information is gradually added until we arrive at the fine-grained representation of conflicts by critical pairs (as e.g. presented in [2]), representing each pair of conflicting transformations in a minimal context. Following this path we introduce several new concepts for conflicting rules and show their interrelations as well as their relations to (essential) critical pairs. Finally, we sketch dual concepts for dependencies.

3.1 Conflicting Rules: Considering Different Granularity Levels

Now, we lift our conflict considerations from transformations to the rule level, i.e., we consider conflicting rules. Two rules are in conflict if there is a pair of conflicting transformations applying these rules. According to Theorem 1 there is a span between these rules specifying the conflict reasons or at least parts of it. In the following, we will concentrate on these spans and distinguish several forms of spans showing conflict reasons in different granularity.

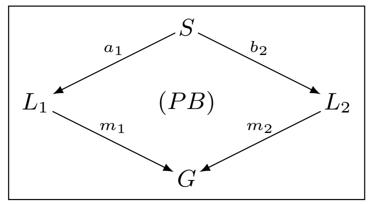
We start focusing on minimal building bricks, called conflict atoms. In particular, we consider a conflict atom to be a minimal sub-graph of C_1 which can be embedded into L_2 but not into B_1 (*conflict and minimality conditions*). Moreover, a pair of direct transformations needs to exist for which the match morphisms overlap on the conflict atom (*transformation condition*). Note that, in general, the matches of this pair of transformations may overlap also in graph elements not contained in the conflict atom. Hence, such a pair of transformations may be chosen flexibly, it need not show a conflict in a minimal context as critical pairs do. While conflict atoms describe the smallest conflict parts, a conflict reason is a complete conflict part in the sense that all in the reported conflict involved atoms are subsumed by it (*completeness condition*). While conflict reasons overlap in conflicting graph elements and boundary nodes only, conflict reason extensions may overlap in non-conflicting elements of the LHSs of participating rules as well (*extended completeness condition*).

Definition 3 (Basic conflict conditions). *Given rules $r_1 : L_1 \overset{le_1}{\hookleftarrow} K_1 \overset{ri_1}{\hookrightarrow} R_1$ and $r_2 : L_2 \overset{le_2}{\hookleftarrow} K_2 \overset{ri_2}{\hookrightarrow} R_2$ with the initial pushout (1) for $K_1 \overset{le_1}{\hookrightarrow} L_1$ as well as a span $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\hookrightarrow} L_2$ as depicted in Fig. 3,* basic conflict conditions *for the span s_1 of (r_1, r_2) are defined as follows:*

1. Conflict condition*: Span s_1 satisfies the* conflict condition *if there does not exist any injective morphism $x : S_1 \rightarrow B_1$ such that $b_1 \circ x = o_1$.*
2. Transformation condition*: Span s_1 satisfies the* transformation condition *if there is a pair of transformations $(t_1, t_2) = (H_1 \overset{m_1,r_1}{\Longleftarrow} G \overset{m_2,r_2}{\Longrightarrow} H_2)$ via (r_1, r_2) with $m_1(c_1(o_1(S_1))) = m_2(q_{12}(S_1))$ (i.e. (2) is commuting in Fig. 3).*

3. Completeness condition*: Span s_1 satisfies the* completeness condition *if there is a pair of transformations $(t_1, t_2) = (H_1 \overset{m_1,r_1}{\Longleftarrow} G \overset{m_2,r_2}{\Longrightarrow} H_2)$ via (r_1, r_2) such that (2) is the pullback of $(m_1 \circ c_1, m_2)$ in Fig. 3.*
4. Minimality condition*: A span $s'_1 : C_1 \overset{o'_1}{\hookleftarrow} S'_1 \overset{q'_{12}}{\rightarrow} L_2$ can be embedded into span s_1 if there is an injective morphism $e : S'_1 \rightarrow S_1$, called* embedding morphism*, such that $o_1 \circ e = o'_1$ and $q_{12} \circ e = q'_{12}$. If e is an isomorphism, then we say that the spans s_1 and s'_1 are* isomorphic*. (See (3) and (4) in Fig. 4.) Span s_1 satisfies the* minimality condition *w.r.t. a set SP of spans if any $s'_1 \in SP$ that can be embedded into s_1 is isomorphic to s_1.*

Finally, span $s : L_1 \overset{a_1}{\hookleftarrow} S \overset{b_2}{\rightarrow} L_2$ fulfills the *extended completeness condition* if there is a pair of transformations $(t_1, t_2) = (H_1 \overset{m_1,r_1}{\Longleftarrow} G \overset{m_2,r_2}{\Longrightarrow} H_2)$ via (r_1, r_2) such that s arises from the pullback of (m_1, m_2) in the figure on the right.



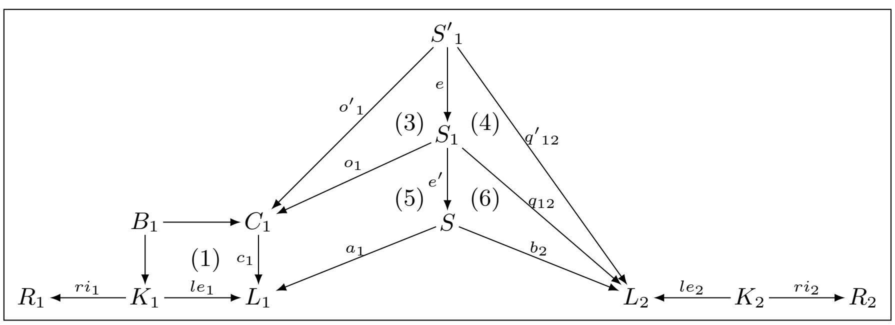


Fig. 4. Illustrating span embeddings

In the following, we define the building bricks of conflicts. The most basic notion to describe a conflict between two rules is that of a *conflict part*. Conflict parts may not describe the whole conflict between two rules. The smallest conflict parts are *conflict atoms*. If a conflict part describes a complete conflict, it is called *conflict reason*.

Definition 4 (Conflict notions for rules). *Let the rules $r_1 : L_1 \overset{le_1}{\hookleftarrow} K_1 \overset{ri_1}{\hookrightarrow} R_1$ and $r_2 : L_2 \overset{le_2}{\hookleftarrow} K_2 \overset{ri_2}{\hookrightarrow} R_2$ with initial pushout (1) for $K_1 \overset{le_1}{\hookrightarrow} L_1$ and a span $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\rightarrow} L_2$ as depicted in Fig. 3, be given.*

1. *Span s_1 is called* conflict part candidate *for the pair of rules (r_1, r_2) if it satisfies the conflict condition. Graph S_1 is called the* conflict graph *of s_1.*
2. *A conflict part candidate s_1 for (r_1, r_2) is a* conflict part *for (r_1, r_2) if s_1 fulfills the transformation condition.*

3. *A conflict part candidate* s_1 *for* (r_1, r_2) *is a* conflict atom candidate *for* (r_1, r_2) *if it fulfills the minimality condition w.r.t. the set of all conflict part candidates for* (r_1, r_2)*.*
4. *A conflict atom candidate* s_1 *for* (r_1, r_2) *is a* conflict atom *for* (r_1, r_2) *if* s_1 *fulfills the transformation condition.*
5. *A conflict part* s_1 *for* (r_1, r_2) *is a* conflict reason *for* (r_1, r_2) *if* s_1 *fulfills the completeness condition.*
6. *A conflict reason* s_1 *for* (r_1, r_2) *is* minimal *if it fulfills the minimality condition w.r.t. the set of all conflict reasons for* (r_1, r_2)*.*
7. *Span* $s : L_1 \overset{a_1}{\hookleftarrow} S \overset{b_2}{\to} L_2$ *is a* conflict reason extension *for* (r_1, r_2) *if it fulfills the extended completeness condition and if there exists a conflict reason* s_1 *for* (r_1, r_2) *with* $e' : S_1 \to S$ *a so-called embedding morphism being injective such that (5) and (6) in Fig. 4 commute. If the latter is the case, we say that* s_1 *can be embedded via* e' *into* s*.*

Note that a conflict part fulfilling the minimality condition is a conflict atom.

Example 2 (Conflict atoms and minimal conflict reasons). Our two example rules in Fig. 2 lead to four pairs of rule combinations to analyze regarding potential conflicts. To discuss the afore introduced building bricks of conflicts we focus on conflicts that may arise by the rule pair (*decapsulateAttribute*, *pullUpEncapsulatedAttribute*), that means by applying the rule *decapsulateAttribute* and making rule *pullUpEncapsulatedAttribute* inapplicable. Since we do not consider attributes and NACs explicitly in this paper, we neglect them within our conflict analysis. Since these features may restrict rule applications, this decision might lead to an over-approximation of potential conflicts.

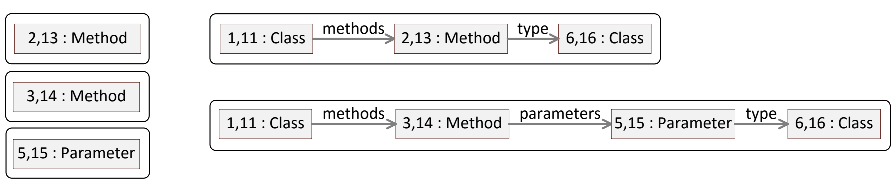


Fig. 5. Conflict atoms (left) and minimal conflict reasons (right) of rule pair (*decapsulateAttribute*, *pullUpEncapsulatedAttribute*)

The root cause of potential conflicts are the three nodes *2:Method*, *3:Method* and *5:Parameter* to be deleted by rule *decapsulateAttribute*. Nodes of the same type are to be used in rule *pullUpEncapsulatedAttribute*. *Method*-nodes are to be deleted twice by rule *decapsulateAttribute* as well as to be used twice in rule *pullUpEncapsulatedAttribute*. Building all combinations this leads to four different conflict atom candidates. Due to the transformation condition, only two of them are conflict atoms: *2,13:Method* and *3,14:Method*, as depicted in Fig. 5 on the left. A further conflict atom is *5,15:Parameter* which is deleted by *decapsulateAttribute* and used by *pullUpEncapsulatedAttribute*. Note that the span notation is rather compact here: Identifying node numbers of rules are used to indicate the mappings of the atom graph into rule graphs. The three conflict atoms are embedded into two *minimal conflict reasons*. Conflict atom *2,13:Method* and the nodes *1,11:Class* and *6,16:Class* are involved within the first *minimal conflict reason*. The remaining two conflict atoms, *3,14:Method* and *5,15:Parameter* can only be covered by a common *minimal conflict reason* due to the completeness condition. This second *minimal conflict reason* also involves nodes *1,11:Class* and *6,16:Class*. These results provide a concise overview on the root causes of the potential conflicts. The three conflict atoms outline the elements responsible for conflicts and the minimal conflict reasons put them into context to their adjacent nodes.

Remark 1 (conflict reasons for rules). In [12], a conflict reason is defined for a given pair of direct transformations (t_1, t_2). Here, we lift the notion of conflict reason to a given pair of rules and relate it with the notion of conflict part. In fact, the above definition of conflict reason for rules requires that at least one pair of transformations exists with exactly this conflict part as conflict reason. While a pair of conflicting transformations has a unique conflict reason, two rules may be related by multiple conflict reasons. Note, moreover, that our conflict reason notion for rules is not completely analogous to the notion of conflict reason for transformations in [12]. It would be analogous if we considered conflict reasons where both rules are responsible together for delete-use-conflicts. Since such conflict reasons would be constructed from the other ones, and since we aim for compact representations of conflicts, we opted for not including this case separately.

Table 1 provides a conflict notion overview and basic conditions.

basic condition / conflict concept	conflict condition	transf. condition	compl. condition	minimality condition
conflict part candidate	x			
conflict part	x	x		
conflict atom candidate	x			x
conflict atom	x	x		x
conflict reason	x	x	x	
min. conflict reason	x	x	x	x

Table 1. Overview of conflict concepts

3.2 Relations between Conflict Notions of Different Granularities

The subsequent results clarify the main interrelations between the new description forms for conflicting rules. All proofs of new results can be found in [14].

In the following extension theorem we state that each conflict part can be extended to a conflict reason. As a special case, it follows automatically that each conflict atom (being a special conflict part) can be extended to a conflict reason.

Theorem 2 (Extension of conflict part to reason). *Given a conflict part* $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\to} L_2$ *for rule pair* $(r_1 : L_1 \overset{le_1}{\hookleftarrow} K_1 \overset{ri_1}{\hookrightarrow} R_1, r_2 : L_2 \overset{le_2}{\hookleftarrow} K_2 \overset{ri_2}{\hookrightarrow} R_2)$, *there is a conflict reason* ${s'}_1 : C_1 \overset{{o'}_1}{\hookleftarrow} {S'}_1 \overset{{q'}_{12}}{\rightarrow} L_2$ *for* (r_1, r_2) *such that the conflict part* s_1 *can be embedded into it.*

The following lemma gives a more constructive characterization of conflict atom candidates compared to their introduction in Def. 4. This result helps us to characterize conflict atom candidates for a given pair of rules. Candidates are either nodes deleted by rule r_1 and used by rule r_2 or edges deleted by r_1 and used by r_2 if their incident nodes are preserved by r_1. Edges with at least one incident deleted node are not considered as atom candidates since their deletion is caused by node deletions.

Lemma 1 (Conflict atom candidate characterization). *A conflict atom candidate* $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\rightarrow} L_2$ *for rules* $(r_1 : L_1 \overset{le_1}{\hookleftarrow} K_1 \overset{ri_1}{\hookrightarrow} R_1, r_2 : L_2 \overset{le_2}{\hookleftarrow} K_2 \overset{ri_2}{\hookrightarrow} R_2)$ *has a conflict graph* S_1 *either consisting of a node* v *s.t.* $o_1(v) \in C_1 \setminus B_1$ *or consisting of an edge* e *with its incident nodes* v_1 *and* v_2 *s.t.* $o_1(e) \in C_1 \setminus B_1$ *and* $o_1(v_1), o_1(v_2) \in B_1$.

Note that, for attributed graphs, the edge in a conflict atom may also be an attribute edge. In this case, the conflict atom would describe an attribute change which is in conflict with an attribute use.

The following theorem states that each conflict reason is covered by a unique set of atoms, i.e. all atoms that can be embedded into that conflict reason. With atoms we mean conflict atoms as well as boundary atoms, where a latter one consists merely of a single boundary node. This means that by investigating the set of conflict atoms one gets a complete overview of graph elements that can cause conflicts in a given conflict reason. Moreover, the set of boundary atoms indicates how this conflict reason might be still enlarged with other conflict-inducing edges. Of course, this result also holds for the special case that the conflict reason is minimal.

Definition 5 (Boundary atom). *A span* $s_1^b : C_1 \overset{o_1^b}{\hookleftarrow} S_1^b \overset{q_{12}^b}{\rightarrow} L_2$ *is a* boundary part *for rules* (r_1, r_2) *with initial pushout (1) as in Fig. 3 if there is a morphism* $s_B : S_1^b \rightarrow B_1$ *such that* $b_1 \circ s_B = o_1^b$ *and* s_1^b *fulfills the transformation condition. A non-empty boundary part* s_1^b *is a* boundary atom *if it fulfills the minimality condition w.r.t. the set of boundary parts for* (r_1, r_2).

It is straightforward to show that graph S_1^b of a boundary atom consists of exactly one boundary node being the source or target node of an edge that is potentially conflict-inducing.

Theorem 3 (Covering of conflict reasons by atoms). *Given a conflict reason* $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\rightarrow} L_2$ *for rules* (r_1, r_2)*, then the set* A *of all conflict atoms together with the set* A^B *of all boundary atoms that can be embedded into* s_1 covers s_1*, i.e. for each conflict reason* ${s'}_1 : C_1 \overset{{o'}_1}{\hookleftarrow} {S'}_1 \overset{{q'}_{12}}{\rightarrow} L_2$ *for* (r_1, r_2) *that can be embedded into* s_1 *it holds that, if each atom in* $A \cup A^B$ *can be embedded into* s'_1*, then* s'_1 *is isomorphic to* s_1.

Conflict reason extensions contain all graph elements that overlap in a pair of conflicting transformations, even elements that are not deleted and at the same time used by any of the two participating rules. Hence, a conflict reason extension might show too much information. By definition, for each conflict reason extension, there is a conflict reason which can be embedded into this extension. Hence, an extension can always be restricted to a conflict reason. Vice versa, the following theorem shows that each conflict reason (being defined over C_1 and L_2) can be extended to at least one conflict reason extension (being defined over L_1 and L_2).

Theorem 4 (Extension of conflict reason to conflict reason extension). *Given a conflict reason $s_1 : C_1 \overset{o_1}{\hookleftarrow} S'_1 \overset{q_{12}}{\rightarrow} L_2$ for rules (r_1, r_2), there exists at least one conflict reason extension $s : L_1 \overset{a_1}{\hookleftarrow} S \overset{b_2}{\rightarrow} L_2$ for rules (r_1, r_2) such that s_1 can be embedded into s.*

3.3 Relations of Conflicting Rule Concepts to Critical Pairs

As illustrated in Fig. 1, for each *critical pair*, there exists an essential critical pair that can be embedded into it (see Completeness Theorem 4.1 in [12]). Match pairs of each (essential) critical pair are jointly surjective (according to the minimal context idea). Thus a critical pair might overlap in elements that are just read by both rules and are not boundary nodes, and exactly these overlaps are *unfolded* again in the essential critical pair. This is because the latter overlaps do not contribute to a new kind of conflict. The set of essential critical pairs is thus smaller than the set of critical pairs and, in particular, each essential critical pair is a critical pair (see Fact 3.2 in [12]).

The following two theorems formalize, on the one hand, the relations between conflict reasons for rule pairs as introduced in this paper and essential critical pairs, and on the other hand, the relations between conflict reason extensions and critical pairs. Note that, as explained in Remark 1, there is no 1-1 correspondence of conflict reasons for rules and essential critical pairs, since we abstract from building symmetrical conflict reasons on the rule level for compactness reasons.

Theorem 5 (Essential critical pair and conflict reason). Restriction. *Given an essential critical pair $(t_1, t_2) = (P_1 \overset{m_1, r_1}{\Longleftarrow} K \overset{m_2, r_2}{\Longrightarrow} P_2)$ such that t_1 causes a delete-use conflict on t_2 then the span $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\rightarrow} L_2$ arising from taking the pullback of $(m_1 \circ c_1, m_2)$ is a conflict reason for (r_1, r_2).*
Extension. *Given a conflict reason $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\rightarrow} L_2$ for rule pair (r_1, r_2) then there exists an essential critical pair $(t_1, t_2) = (P_1 \overset{m_1, r_1}{\Longleftarrow} K \overset{m_2, r_2}{\Longrightarrow} P_2)$ such that t_1 causes a delete-use conflict on t_2 with the pullback of $(m_1 \circ c_1, m_2)$ being isomorphic to s_1.*

Theorem 6 (Critical pair and conflict reason extension). Restriction. *Given a critical pair $(t_1, t_2) = (P_1 \overset{m_1, r_1}{\Longleftarrow} K \overset{m_2, r_2}{\Longrightarrow} P_2)$ such that t_1 causes a delete-use conflict on t_2 then the span arising from taking the pullback of (m_1, m_2) is a conflict reason extension for (r_1, r_2).*

Extension. *Given a conflict reason extension* $s : L_1 \overset{a_1}{\hookleftarrow} S \overset{b_2}{\hookrightarrow} L_2$ *for* (r_1, r_2) *then the cospan arising from building the pushout of* (a_1, b_2) *defines the matches* (m_1, m_2) *of a critical pair* $(t_1, t_2) = (P_1 \overset{m_1, r_1}{\Longleftarrow} K \overset{m_2, r_2}{\Longrightarrow} P_2)$ *such that* t_1 *causes a delete-use conflict on* t_2*.*
Bijective correspondence. *The restriction and extension constructions are inverse to each other up to isomorphism.*

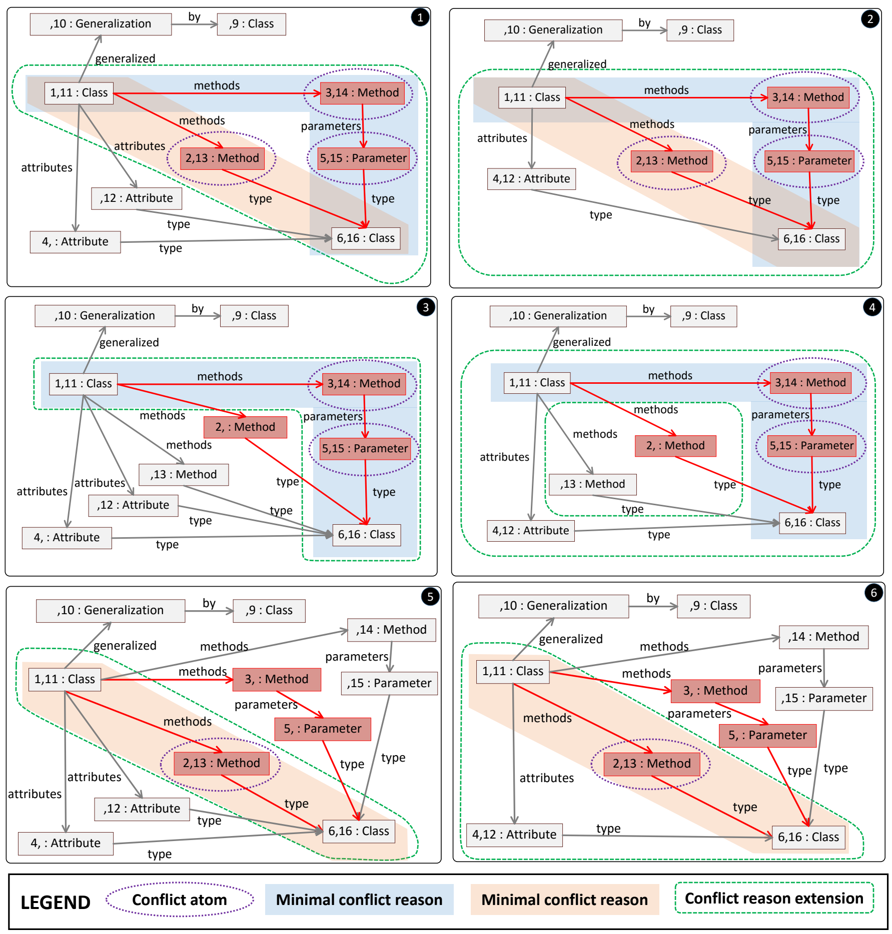


Fig. 6. Representation of six critical pairs arising from the application of rule *decapsulateAttribute* so that rule *pullUpEncapsulatedAttribute* becomes inapplicable; examples of newly introduced conflict notions are indicated.

Example 3 (Conflict reason extension). Fig. 5 focuses on the conflict atoms and minimal conflict reasons of the rule pair (*decapsulateAttribute*, *pullUpEncapsulatedAttribute*). Fig. 6 relates these new conflict notions with the six critical pairs of the considered rule pair. The two *minimal conflict reasons* sufficiently characterize the overlap in the results 3 and 5. Result 1 presents the combination of both *minimal conflict reasons*. Since these results make no use of further overlapping of non-deleting elements they are also conflict reason extensions. Moreover, they correspond to the results of the essential critical pair analysis. *1,11:Class* and *6,16:Class* are two *boundary atoms*. Additional overlapping of the *Attribute*-nodes of both rules in *4,12:Attribute* leads to larger *conflict reason extensions* and to the remaining three results 2,4, and 6. Adding the remaining elements of the LHS of both rules, we obtain a compact representation of all six critical pairs.

3.4 Dual Notions for Dependencies

To reason about dependencies of rules and transformations, we consider the dual concepts and results that we get when inverting the left transformation of a conflicting pair. This means that, instead of considering conflicting transformations $(t_1, t_2) = (H_1 \overset{m_1,r_1}{\Longleftarrow} G \overset{m_2,r_2}{\Longrightarrow} H_2)$, we consider dependent transformations $(t_1; t_2) = (H_1 \overset{m_1^{-1},r_1^{-1}}{\Longleftarrow} G \overset{m_2,r_2}{\Longrightarrow} H_2) = (G \overset{m_1,r_1}{\Longrightarrow} H_1 \overset{m_2,r_2}{\Longrightarrow} H_2)$. This is possible since a transformation is symmetrically defined by two pushouts. They ensure in particular that morphisms $m : L \to G$ as well as $m' : R \to H$ fulfill the gluing condition.

Dependency parts, atoms, reasons, and reason extensions can be defined analogously to Def. 4. They characterize graph elements being produced by the first rule application and used by the second one. Results presented for conflicts above can be formulated and proven for dependencies in an analogous way.

4 Related Work and Conclusion

The critical pair analysis (CPA) has developed into the standard technique for detecting conflicts and dependencies in graph transformation systems [1] at design time. Originally being developed for term and term graph rewriting [15], it extends the theory of graph transformation and, more generally, of $\mathcal{M}$-adhesive transformation systems [16, 2]. The CPA is not only available for plain rules but also for rules with application conditions [17].

In this paper, we lay the basis for a refined analysis of conflicts and dependencies by presenting conflict and dependency notions of different granularity. Furthermore, we investigate their interrelations. The formal consideration shall be used in a new CDA technique where conflict and dependency analysis can go from coarse-grained information about the potential existence of conflicts or dependencies and their main reasons, to fine-grained considerations of conflict and dependency reasons in different settings.

The CPA is offered by the graph transformation tools AGG [18] and Verigraph [19] and the graph-based model transformation tool Henshin [20]. All of them provide the user with a set of (essential) critical pairs for each pair of rules as analysis result. The computation of conflicts and dependencies using the concepted introduced in the present work has been prototypically implemented in

Henshin. First tests indicate that our analysis is very fast and yields concise results that are promising to facilitate understandability. However, it is up to future work to further investigate this aspect in a user study.

Currently, we restrict our formal considerations to graphs and graph transformations. Since all main concepts are based on concepts from category theory, our work is prepared to adapt to more sophisticated forms of graphs or graph transformation. Furthermore, it is interesting to adapt the new notions to transformation rules with negative [21] or more complex nested application conditions [17]. Analogously, to handle attributes within conflicts appropriately it is promising to adapt our approach to lazy graph transformations [22] and to come up with a light-weight conflict analysis complementing the work of Deckwerth et al. [23] on conflict detection of edit operations on feature models. They combine CPA with an SMT solver for an improved handling of conflicts based on attribute changes. Performance is still a limiting factor for applying the CPA to large rule sets. A *family-based analysis* based on the unification of multiple similar rules [24] is a promising idea to save redundant computation effort.

Acknowledgements. We wish to thank Jens Kosiol and the anonymous reviewers for their constructive comments. This work was partially funded by the German Research Foundation, Priority Program SPP 1593 "Design for Future – Managed Software Evolution". This research was partially supported by the research project Visual Privacy Management in User Centric Open Environments (supported by the EU's Horizon 2020 programme, Proposal number: 653642).

References

1. R. Heckel, J. M. Küster, and G. Taentzer, "Confluence of Typed Attributed Graph Transformation Systems," in *First Int. Conf. on Graph Transformation (ICGT)*, ser. LNCS, vol. 2505. Springer, 2002, pp. 161–176.
2. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer, *Fundamentals of Algebraic Graph Transformation*, ser. Monographs in Theoretical Computer Science. Springer, 2006.
3. J. H. Hausmann, R. Heckel, and G. Taentzer, "Detection of Conflicting Functional Requirements in a Use Case-Driven Approach: A Static Analysis Technique Based on Graph Transformation," in *22rd Int. Conf. on Software Engineering (ICSE)*. ACM, 2002, pp. 105–115.
4. P. Jayaraman, J. Whittle, A. M. Elkhodary, and H. Gomaa, "Model composition in product lines and feature interaction detection using critical pair analysis," in *Int. Conf. on Model Driven Engineering Languages and Systems*. Springer, 2007, pp. 151–165.
5. J. M. Küster, C. Gerth, and G. Engels, "Dependent and conflicting change operations of process models," in *European Conf. on Model Driven Architecture - Foundations and Applications*, vol. 5562. Springer, pp. 158–173.
6. K. Mehner-Heindl, M. Monga, and G. Taentzer, "Analysis of Aspect-Oriented Models Using Graph Transformation Systems," in *Aspect-Oriented Requirements Engineering*, A. Moreira, R. Chitchyan, J. Araújo, and A. Rashid, Eds. Springer, 2013, pp. 243–270.
7. T. Mens, R. Van Der Straeten, and M. D'Hondt, "Detecting and resolving model inconsistencies using transformation dependency analysis," in *9th Int. Conf. on Model Driven Engineering Languages and Systems*, ser. MoDELS'06. Springer, 2006, pp. 200–214.

8. T. Mens, G. Taentzer, and O. Runge, "Analysing refactoring dependencies using graph transformation," *Software and System Modeling*, vol. 6, no. 3, pp. 269–285, 2007.
9. L. Baresi, R. Heckel, S. Thöne, and D. Varró, "Modeling and validation of service-oriented architectures: application vs. style," in *ACM SIGSOFT Symposium on Foundations of Software Engineering held jointly with 9th European Software Engineering Conference*. ACM, 2003, pp. 68–77.
10. C. Ermel, J. Gall, L. Lambers, and G. Taentzer, "Modeling with plausibility checking: Inspecting favorable and critical signs for consistency between control flow and functional behavior," in *Int. Conf. on Fundamental Approaches to Software Engineering*. Springer, 2011, pp. 156–170.
11. D. Strüber, G. Taentzer, S. Jurack, and T. Schäfer, "Towards a distributed modeling process based on composite models," in *Int. Conf. on Fundamental Approaches to Software Engineering*. Springer, 2013, pp. 6–20.
12. L. Lambers, H. Ehrig, and F. Orejas, "Efficient conflict detection in graph transformation systems by essential critical pairs," *Electr. Notes Theor. Comput. Sci.*, vol. 211, pp. 17–26, 2008.
13. M. Fowler, *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley, 1999.
14. K. Born, L. Lambers, D. Strüber, and G. Taentzer, "Granularity of conflicts and dependencies in graph transformation systems: Extended version," Philipps-Universität Marburg, Tech. Rep., 2017. [Online]. Available: www.uni-marburg.de/fb12/swt/research/publications
15. D. Plump, "Critical Pairs in Term Graph Rewriting," in *Mathematical Foundations of Computer Science*, vol. 841, 1994, pp. 556–566.
16. H. Ehrig, J. Padberg, U. Prange, and A. Habel, "Adhesive high-level replacement systems: A new categorical framework for graph transformation," *Fundam. Inform.*, vol. 74, no. 1, pp. 1–29, 2006.
17. H. Ehrig, U. Golas, A. Habel, L. Lambers, and F. Orejas, "$\mathcal{M}$-adhesive transformation systems with nested application conditions. part 2: Embedding, critical pairs and local confluence," *Fundam. Inform.*, vol. 118, no. 1-2, pp. 35–63, 2012. [Online]. Available: http://dx.doi.org/10.3233/FI-2012-705
18. G. Taentzer, "AGG: A graph transformation environment for modeling and validation of software," in *Int. Workshop on Applications of Graph Transformations with Industrial Relevance*. Springer, 2003, pp. 446–453.
19. Verigraph, "Verigraph," https://github.com/Verites/verigraph.
20. T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer, "Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations," in *Model Driven Engineering Languages and Systems*, ser. LNCS, vol. 6394, pp. 121–135, http://www.eclipse .org/henshin/.
21. L. Lambers, "Certifying rule-based models using graph transformation," Ph.D. dissertation, Berlin Institute of Technology, 2010.
22. F. Orejas and L. Lambers, "Lazy graph transformation," *Fundam. Inform.*, vol. 118, no. 1-2, pp. 65–96, 2012.
23. F. Deckwerth, G. Kulcsár, M. Lochau, G. Varró, and A. Schürr, "Conflict detection for edits on extended feature models using symbolic graph transformation," in *Int. Workshop on Formal Methods and Analysis in Software Product Line Engineering*, ser. EPTCS, vol. 206, 2016, pp. 17–31.
24. D. Strüber, J. Rubin, T. Arendt, M. Chechik, G. Taentzer, and J. Plöger, "Rulemerger: Automatic construction of variability-based model transformation rules," in *Int. Conf. on Fundamental Approaches to Software Engineering*. Springer, 2016, pp. 122–140.