Granularity of Conflicts and Dependencies in Graph Transformation Systems: A Two-Dimensional Approach

Leen Lambers[a], Kristopher Born[b], Jens Kosiol[b], Daniel Strüber[c], Gabriele Taentzer[b]

[a] *Hasso-Plattner-Institut, Potsdam, Germany*
[b] *Philipps-Universität Marburg, Germany*
[c] *Universität Koblenz-Landau, Germany*

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 all potential conflicts and dependencies in minimal context as *critical pairs*, for each pair of rules. Yet, critical pairs can be hard to understand; users are mainly interested in core information about conflicts and dependencies occurring in various combinations. In this paper, we present an approach to conflicts and dependencies in graph transformation systems based on *two dimensions of granularity*. The first dimension refers to the *overlap* considered between the rules of a given rule pair; the second one refers to the represented amount of *context* information about transformations in which the conflicts occur. We introduce a variety of new conflict notions, in particular, *conflict atoms*, *conflict reasons*, and *minimal conflict reasons*, relate them to the existing conflict notions of *critical pairs* and *initial conflicts*, and position all of these notions within our granularity approach. Finally, we introduce dual concepts for dependency analysis. As we discuss in a running example, our approach paves the way for an improved

Email addresses: `leen.lambers@hpi.de` (Leen Lambers), `born@informatik.uni-marburg.de` (Kristopher Born), `kosiolje@informatik.uni-marburg.de` (Jens Kosiol), `strueber@uni-koblenz.de` (Daniel Strüber), `taentzer@informatik.uni-marburg.de` (Gabriele Taentzer)

CDA technique.

Keywords: Graph Transformation (Double Pushout Approach), Parallel Independence, Critical Pair Analysis (CPA)

---

1. Introduction

Graph transformation systems (GTSs) are a fundamental modeling concept with applications in a wide range of domains, including software engineering [1], mechanical engineering [2], and biology [3]. 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 first case, the specified control flow might not be executable due to conflicts or dependencies. 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.

To verify the explicit control flow of a GTS (as done, e.g., in [4]) or to better understand an implicit one (as done, e.g., in [5]), 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. It has been applied to many use-cases such as, e.g., to detect conflicting functional requirements [4], conflicts and dependencies between refactorings [5, 6], feature interactions [7], conflicting and dependent change operations for process models [8], causal dependencies of aspects in aspect modeling [9], and to validate service-oriented architectures [10].

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 used then 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 [11]. 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 [12]. 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.

The state-of-the-art CDA technique, which has been used in most of the aforementioned use-cases, is *critical pair analysis* (CPA, [13, 14]). For a given pair of rules, CPA produces a list of *critical pairs*, representing potential conflicts and dependencies between the rules. A critical pair specifies a graph to which both rules can be applied such that a conflict arises from the rule applications; the graph represents a minimal context in the sense that each of its elements stems from the rules' left-hand sides. Confidence in CPA is established by positive fundamental results: Via the *Completeness Theorem*, each pair of conflicting or dependent rule applications is represented by a critical pair. If the transformation system contains any conflicts, users may be interested in knowing if it is possible to resolve all of them to check the system's confluence. The *Local-Confluence Theorem* allows to answer this question by providing a sufficient condition that can be checked statically for each critical pair.

However, experiences with the CPA indicate two drawbacks: (i) understanding the identified critical pairs can be a challenging task since they often display too much information, (ii) calculating the results can be computationally expensive. To address these issues, our recent work [15] introduces a distinguished subset of critical pairs, called *initial conflicts and dependencies*. While keeping the Completeness and Local-Confluence properties, this subset is usually much smaller and therefore, potentially easier to understand and to compute than the set of all critical pairs. Still, this earlier contribution presents only a preliminary step. Users are still overwhelmed with information: each initial conflict or dependency contains a, potentially large, graph obtained by overlapping both rules, and the same root cause of a conflict or dependency is often shown in many different combinations. Currently, the set of initial conflicts can be computed

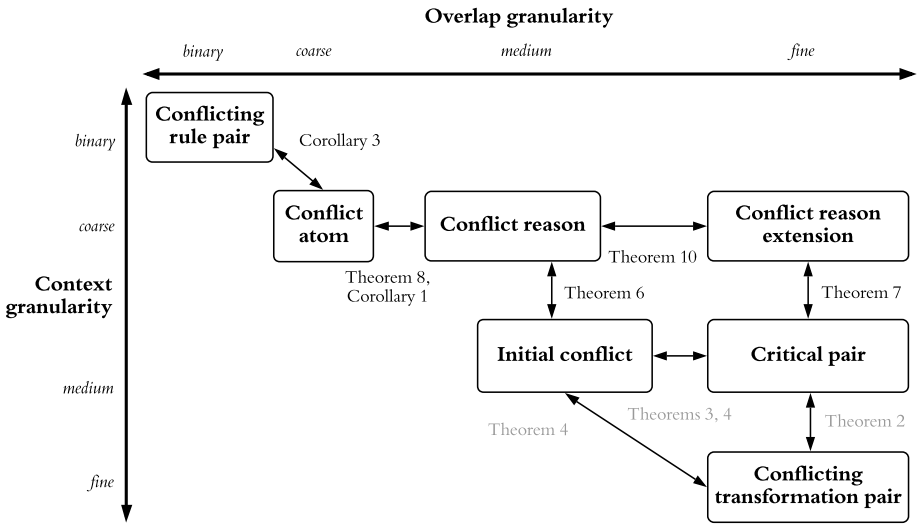


Figure 1: Positioning of conflict notions in two dimensions of granularity.

efficiently only under restricted conditions, as introduced in another work [16].

In this paper, to address the drawbacks of critical pairs as well as initial conflicts and dependencies, we present a new approach to conflicts and dependencies in GTSs. Our approach is based on the paradigm of *granularity* and, building on the existing theory for algebraic graph transformation, focuses on *delete-use*-conflicts. We investigate a variety of new notions for describing conflicts, and put these notions into relation to the existing ones. The new notions alleviate the aforementioned issues by (i) stripping away the potentially unnecessary context of overlap graphs, and the possibility to abstract from potentially irrelevant overlapped elements, and (ii) opening up new possibilities for computing conflicts based on the identified relationships between notions. By clarifying the relations between old as well as new conflict notions along two newly introduced granularity dimensions, our approach paves the way for reusing basic constructions and hence, efficiently computing conflicts on any granularity level.

Figure 1 shows an overview of the considered conflict notions and their relations; relations are underpinned with formal results, including new results (shown in black) and existing ones from the literature (shown in gray). The

conflict notions are aligned along two dimensions of granularity, which are denoted as vertical and horizontal scales in the figure, respectively: *context* and *overlap granularity*. These two dimensions quantify a number of involved elements. Context granularity refers to the amount of context in which a conflict is reported. The binary granularity level refers to a yes-no decision, i.e., the presence or absence of conflicts, without showing any context. A coarse conflict notion in this sense depicts only rule elements, a medium one a graph obtained from overlapping two rules, and a fine one includes additional elements not contained in the given rules. Overlap granularity refers to the overlap between the two considered rules. Binary granularity does not show any details about overlap, coarse only overlaps a smallest set of elements, medium only overlaps deleting elements, and fine incorporates non-deleting ones, being irrelevant for understanding the conflict.

Starting with the maximally fine-grained conflict notion in both dimensions, we consider *conflicting transformation pairs*. A conflicting transformation pair is represented by a graph to which both rules can be applied, so that the first rule application renders the second one impossible. Since the graph may be arbitrary large—in particular, it may contain elements for which no counterpart in either of the rules exists—there can be infinitely many of such pairs for a given pair of rules. Critical pairs and initial conflicts, explained above, strip away this typically unnecessary context. Each conflicting transformation pair can be represented by a unique critical pair which might be an initial conflict. Conversely, each critical pair, including the initial conflicts, is a conflicting transformation pair and can be extended into several ones by embedding it into a bigger context not affected by the rule applications. Each critical pair can be represented by an initial conflict, obtained, in a certain sense, by unfolding it.

Moving to the more coarse-grained notions with respect to context granularity, we now consider only the overlap of the input rules (intuitively, we now consider intersections, rather than unions). *Conflict reason extensions* and *conflict reasons* comprise a set of elements being deleted by the first and used by the second rule. For conflict reason extensions, the considered overlap may in-

clude additional non-deleting elements from the considered rules. Each conflict reason extension can be extended to one critical pair, and each critical pair can be restricted to a conflict reason extension. Conflict reasons roughly correspond to initial conflicts. *Conflict atoms* represent the smallest entities of conflicts, which can be characterized as single conflict-inducing graph elements. Each conflict atom can be embedded into a conflict reason, while each conflict reason is fully covered by conflict atoms. A pair of rules affected by conflicts is called a *conflicting rule pair*; for each such pair, at least one conflict atom exists. The pair notion is binary, i.e., does not include information on overlap or context. Moving from conflict reason extensions to reasons, atoms, and conflicting rule pairs, we reduce the considered overlap in each step, while the context is fixed.

Contributions. In this paper, we investigate the granularity of conflicts and dependencies in GTSs. Specifically, we make the following contributions.

- We present a conceptual consideration of conflicts in GTSs, based on the two dimensions of overlap and context granularity, and focusing on delete-use-conflicts.
- We arrange existing conflict notions along both granularity dimensions and introduce new notions for coarse-grained context granularity.
- We introduce a variety of formal results for interrelating the new conflict notions with each other and with the existing ones. In particular, we relate the new conflict notions to the existing notions of critical pairs as well as initial conflicts.
- We discuss how these notions and results can be transferred to dependencies in a straightforward manner. In particular, we introduce dependency atoms and reasons, the dual notions to those introduced for conflict analysis.

This paper is an extended version of [17], in which we presented an earlier state of our work, focusing on overlap granularity. In the present paper, we

significantly extended the results from the earlier version to align them with the theory of initial conflicts and dependencies [15]. Specifically, we make three extensions: First, we introduce a two-dimensional approach to granularity and position the old and new conflict notions in it. Second, we study the relations between the conflict and dependency notions from [17] and initial conflicts. Third, we present a full account of formal results for relating the conflict notions with each other; results from [17] have been updated and extended based on the following considerations.

In [17], *essential critical pairs*, an earlier candidate for an optimized set of critical pairs, played the same role as initial conflicts do in the present paper. The shift to initial conflicts is motivated by the findings from [15]: Initial conflicts are a cleaner replacement for essential critical pairs, since they avoid certain counter-intuitive cases (related to a phenomenon called *isolated boundary nodes*). To align the new granularity notions with initial conflicts, our starting point is a change in the definition of conflict reasons. We introduce a stronger conflict condition than in [17], which, as we will show, is more suitable for characterizing medium overlap granularity. All results have been extended to take the new conflict condition into account, and new results have been added in order to complete our account of relationships. The running example has been extended as well to illustrate the notion of initial conflicts and the accordingly revised notion of conflict reason.

The rest of this paper is structured as follows: Section 2 illustrates our approach using an example. In Sect. 3, we recall graph transformation concepts and conflict notions from the literature. In Sect. 4, we present the new conflict notions and position the new as well as the existing ones within our two-dimensional approach; formal results for interrelating all presented conflict notions are introduced. We compare with related work and conclude in Sect. 5.

2. Introduction to Granularity of Conflicts by Example

Before starting formal considerations, we illustrate our new conflict notions presented in this article by an example. Readers can get a good impression of

our work by reading this section.

Our running example is concerned with conflicts between software refactorings. Refactoring is a generally acknowledged technique to improve the design of an object-oriented system [18]. To achieve a larger improvement there is typically a sequence of refactorings required. Due to implicit conflicts and 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. To this end, 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.

The structure of an object-oriented software system can be modeled using typed graphs, and refactorings can be specified using graph transformation rules. Since the theory is restricted to delete-use conflicts for now, we consider simple rules without attributes and application conditions here. Figure 2 shows two rules which specify class-structure refactorings *Decapsulate Attribute* and *Pull Up Encapsulated Attribute* in a simple form.

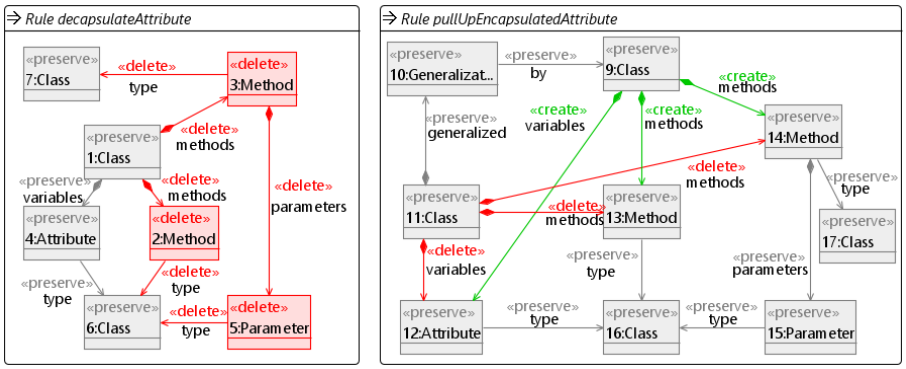


Figure 2: Simplified rules specifying *Decapsulate Attribute* and *Pull Up Encapsulated Attribute*

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 left-hand side (LHS) of a rule, the preserved and created elements form its right-hand side (RHS). A rule is applied by finding the LHS as pattern in an instance graph, removing all elements to be deleted and

adding all elements to be created. Elements with the same annotation form a common condition. 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. Note that we do not consider names here, i.e., getter and setter methods are just distinguished by having no parameter (getter) and one parameter (setter). Not considering attributes, we do not check if the superclass has already an attribute or methods with the same names as the ones we want to pull up. Note that there are two kinds of arrows, those starting in a diamond specifying containment relations, and those without diamond specifying normal references. In what follows, we will not further address this distinction. Instead, edges are distinguished in terms of their types only. Besides names, we also omit the visibility of attributes and methods for simplicity.

Conflict considerations. In the following, we informally explain all main conflict notions depicted in Fig. 1 by our running example. We introduce them along increasing *overlap* and *context granularity levels*. Starting on the binary level, we continue with conflict atoms and conflict reasons on the coarse context granularity level and go over to conflicting transformation pairs representing the medium and fine context granularity levels.

Binary context granularity. Our two example rules in Fig. 2 lead to four pairs of rules. As starting point of our conflict considerations, we check if there are any potential conflicts caused by the first rule on the second rule of a given pair, i.e., if there are *conflicting rule pairs*. If the application of the first rule can make the second rule inapplicable in consequence, the corresponding entry in Table 1 contains '+' while '–' marks that there is no conflict.

Coarse context granularity. In the following, we consider the rule pair (*decapsulateAttribute*, *pullUpEncapsulatedAttribute*) more closely. The root causes of potential conflicts are the three nodes 2:Method, 3:Method and 5:Parameter to

Rule 1 / Rule 2	*decapsulateAttribute*	*pullUpEncapsulatedAttribute*
decapsulateAttribute	+	+
pullUpEncapsulatedAttribute	+	+

Table 1: Overview on conflicting rule pairs

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 candidates for conflict atoms. Only three of them can occur in conflicting transformation pairs. Nodes 2,13:Method, 3,13:Method, and 3,14:Method are actually *conflict atoms*; they are depicted in Fig. 3 on the left. Note that 2,14:Method is not a conflict atom since the corresponding overlap would lead to a dangling parameters-edge, i.e., there is no conflicting rule pair overlapping in this node. A further conflict atom exists for nodes of type Parameter being 5,15:Parameter which is deleted by *decapsulateAttribute* and used by *pullUpEncapsulatedAttribute*. Note that the numbers in atom nodes indicate the overlap of corresponding rule nodes in this figure and in following ones. The first number of each node indicates an element of the left rule while the second number concerns an element of the right rule.

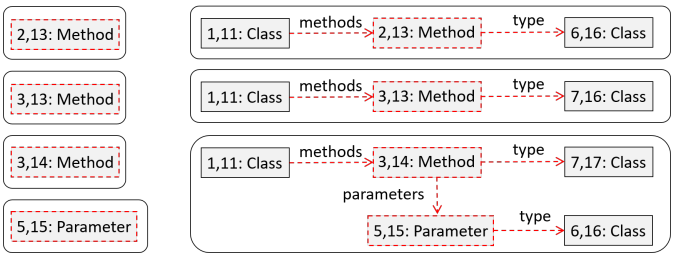


Figure 3: Conflict atoms (left) and minimal conflict reasons (right) of rule pair (*decapsulateAttribute*, *pullUpEncapsulatedAttribute*)

The four conflict atoms are embedded into three *minimal conflict reasons* shown on the right of Fig. 3. Each minimal reason describes a minimal in-

tersection of two rule applications to be conflicting. The two conflict atoms 3,14:Method and 5,15:Parameter can only be covered by a common *minimal conflict reason* since an intersection in only one conflict atom would lead to one or more dangling edges, rendering the first rule inapplicable. Note that elements to be deleted are shown with red dashed frames or lines. Minimal conflict reasons are the building bricks of *conflict reasons*, i.e., all their possible combinations are further conflict reasons. Altogether, we have 4 *conflict reasons*: Each minimal conflict reason alone and the combination of the top and the bottom ones in Fig. 3.

Medium and fine context granularity levels. All conflict reasons show overlaps of the participating rules' LHSs that lead to *conflicting transformation pairs* when applying these rules. Joining the LHSs of both rules along a conflict reason for rule pair (r_1, r_2), potentially combined with another one for pair (r_2, r_1), we get an overlap graph which can be used to construct a conflicting transformation pair in minimal context.

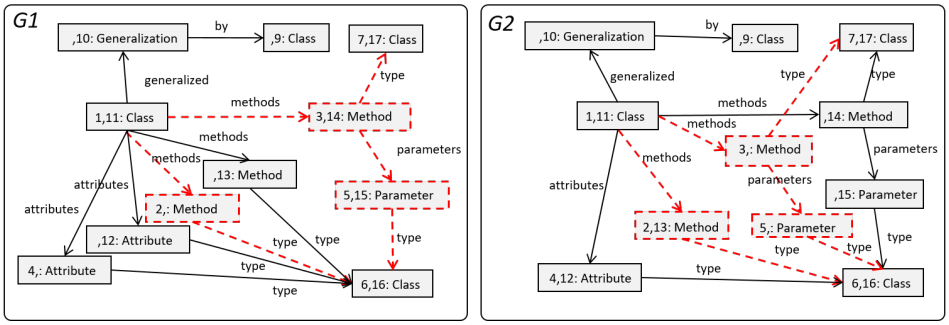


Figure 4: Two overlap graphs for simplified refactoring rules *decapsulateAttribute* and *pullUpEncapsulatedAttribute*, only the left one belongs to an initial conflict

Figure 4 shows two possible overlap graphs G1 and G2. Note that as before the first number in the node labels indicates an element stemming from the left rule while the second number concerns an element stemming from the right rule. If no first or second number occurs, then this means that the node was not overlapped at all and therefore only stems from the left or right rule, respectively.

Both overlap graphs present conflicting situations in minimal context. G1 shows the deletion of a method with parameter which shall be pulled up. G2 shows the deletion of a parameterless method to be pulled up as well. In addition, there are overlaps in 7,17:Class and 4,12:Attribute. Since these nodes are preserved in both rules, their overlaps do not harm and are therefore less interesting for the user. Nevertheless, such overlaps are identified by *conflict reason extensions*. Since 7:Class has incident edges to be deleted it is called *boundary node*. Its overlap with 17:Class does not coincide with overlaps of adjacent edges. Therefore, this overlap is called *isolated*. Joining the rules' LHSs at a conflict reason extension, the corresponding conflicting transformation pair is called *critical pair*. Both graphs, G1 and G2 (Fig. 4), show overlap graphs of critical pairs. Graph G1 shows an overlap graph without overlaps of isolated preserved nodes. Moreover, a preserved node of the first rule's LHS is overlapped with a node of the second rule's LHS only if a deleted adjacent edge is also overlapped (compare, e.g., ,13:Method but 6,16:Class). In general, deleted nodes or deleted edges (with incident nodes) are the only elements of the first rule that are overlapped with elements of the second rule. A critical pair with such an overlap graph is called *initial conflict*. Rule pair (*decapsulateAttribute*, *pullUpEncapsulatedAttribute*) has 13 critical pairs altogether, 4 of them are initial conflicts.

Each overlap graph is the starting graph for a conflicting transformation pair. Figure 5 shows the result graphs H11 and H12 when applying the simplified refactoring rules *decapsulateAttribute* (left) and *pullUpEncapsulatedAttribute* (right) to graph G1 in Fig. 4. We see that rule *pullUpEncapsulatedAttribute* cannot be applied to graph H11 since one method is missing. In contrast, rule *decapsulateAttribute* can still be applied to graph H12, however, not at the original match anymore.

3. Preliminaries

As a prerequisite for our new analysis of conflicts and dependencies, we recall the double-pushout approach to graph transformation as presented in

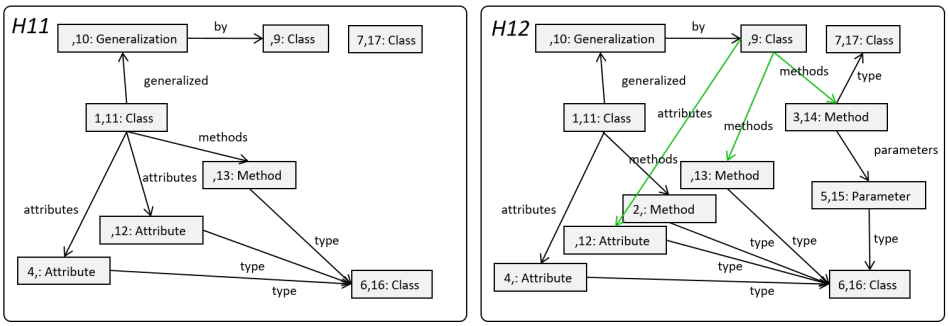


Figure 5: Result graphs when applying the simplified refactoring rules *decapsulateAttribute* (left) and *pullUpEncapsulatedAttribute* (right) to G1 graph in Fig. 4

[14]. Furthermore, we reconsider the notion of conflicts as well as the recently introduced concept of initial conflicts.

3.1. Graph Transformation: Double-Pushout Approach

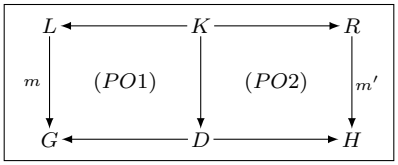
Throughout this paper we consider graphs and graph morphisms as presented in [14] for the category of graphs; all results can be easily extended to the category of typed graphs by assuming that each graph and morphism is typed over some fixed type graph TG. Since most of the definitions and results are given in a category-theoretical way, the extension to e.g. typed, attributed graphs [14] 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 is the graph part that is not changed; it is equal to $L \cap R$ provided that the result is a graph again. 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 *direct graph transformation* $G \overset{m,r}{\Longrightarrow} H$ 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 $H := D \cup m'(R \setminus K)$. The morphism m' has to be chosen 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* [14]. 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 with more ease than for general matches. In categorical terms, a direct transformation step is defined using a so-called double pushout as in the following definition. Thereby step (1) in the previous informal explanation is represented by the first pushout and step (2) by the second one [14].

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. Rule r is *applicable* and the injective morphism $m : L \to G$ is called *match* if there exists a graph D such that $(PO1)$ is a pushout. Rule r is *non-deleting* if $L = K$. A *transformation* is a sequence $G_0 \Rightarrow G_1 \Rightarrow \ldots \Rightarrow G_n$ of direct transformations, written $G_0 \Rightarrow^* G_n$.



Example 1 (Graph transformation)**.** Applying the simplified refactoring rule

[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 in the main text we denote inclusions by $\hookrightarrow$ and all other morphisms by $\to$.

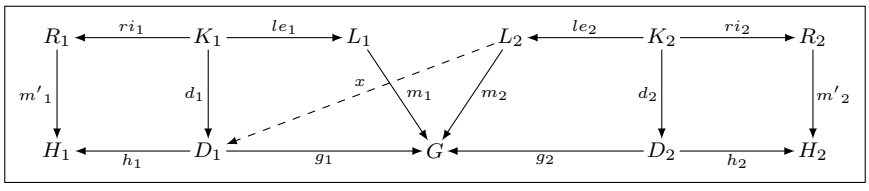
decapsulateAttribute to graph G1 in Fig. 4, there are several matches since we can choose to match 4,:Attribute or ,12:Attribute as well as 2,:Method or ,13:Method. Choosing to match 4,:Attribute and 2,:Method, this partial match can be completed in only one way. It fulfills the dangling condition since, in graph G1, all adjacent edges of nodes to be deleted are explicitly specified in the rule. The resulting graph is H11 depicted in Fig. 5.

3.2. Conflicts

In this subsection, we recall existing conflict notions and critical pairs representing conflicts in a minimal context. We moreover reintroduce a recent result on initial conflicts representing a subset of critical pairs being still complete as well as usable for static local confluence analysis. In particular, we concentrate on delete-use conflicts which means that the first rule application deletes graph items that are used by the second rule application.

The definition of a delete-use conflict [14] states that the match of the second transformation cannot be found anymore after applying the first transformation. This may happen if the second transformation wants to preserve (delete-read) or delete (delete-delete) elements that are deleted by the first transformation. Note that we do not consider delete-use conflicts of the second transformation on the first one explicitly. To get those ones as well, we simply consider the inverse pair of transformations.

Definition 2 (Delete-use conflict)**.** Given a pair of direct transformations $(t_1, t_2) = (G \overset{m_1,r_1}{\Longrightarrow} H_1, G \overset{m_2,r_2}{\Longrightarrow} H_2)$ 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 pair* (t_1, t_2) is *delete-use conflicting* if there does not exist a morphism $x : L_2 \to D_1$ such that $g_1 \circ x = m_2$. *Rule pair* (r_1, r_2) is *delete-use conflicting* if there exists a delete-use conflicting transformation pair $(t_1, t_2) = (G \overset{m_1,r_1}{\Longrightarrow} H_1, G \overset{m_2,r_2}{\Longrightarrow} H_2)$.



Understanding this definition set-theoretically, it means that at least one element is deleted by r_1 and used by r_2, i.e., $m_1(L_1) \cap m_2(L_2) \not\subseteq m_1(le_1(K_1))$. In the rest of the paper we merely consider delete-use conflicts such that in the following we abbreviate *delete-use conflict* with *conflict*.

This work is inspired by an alternative characterization for a pair of transformations to be in delete-use conflict (as introduced in [19]), expressing that at least one deleted element of the first transformation overlaps with some used element of the second transformation. As a prerequisite, we consider closer the deletion information contained in a rule. We use an initial pushout construction [14] over the left-hand side morphism $K \to L$ of a rule to extract the *deletion graph* C stripping away as much preserved graph elements from L as possible. The remaining graph C contains all elements to be deleted completed by those preserved nodes of K that are needed to complete $L \setminus K$ to a graph. These additional nodes are called *boundary nodes* summarized in graph B. Then the overlap characterizing delete-use conflicts in an alternative way is formally expressed by a span s_1 of graph morphisms between the deletion graph of the first rule, and the LHS of the second rule (Fig. 6). In particular, this overlap should not contain only boundary elements, otherwise no conflicting elements between the rule applications exist. In the following section, we will reuse this condition on the span s_1 and call it in particular *weak conflict condition* (compare Def. 6).

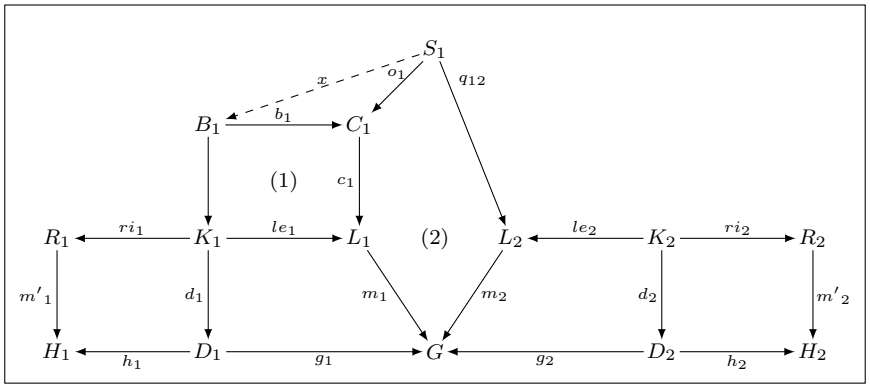


Figure 6: Conflict characterization

Theorem 1 (Conflict characterization [19, 20])**.** *Given a pair of transformations* $(t_1, t_2) = (G \overset{m_1,r_1}{\Longrightarrow} H_1, G \overset{m_2,r_2}{\Longrightarrow} H_2)$ *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. 6 yielding the span* $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\Rightarrow} L_2$*, then the following equivalence holds:* (t_1, t_2) *are conflicting according to Def. 2 iff there does not exist a morphism* $x : S_1 \to B_1$ *such that* $b_1 \circ x = o_1$*.*

Static conflict detection as well as *local confluence analysis* are based on the idea of *critical pairs* representing all possible conflicting transformation pairs in a minimal context. In the classical pair definition this minimal context is materialized by a pair of jointly surjective matches, i.e., each element in the common codomain of the two morphisms has a pre-image in one of the domains of both morphisms.

Definition 3 (Critical pair)**.** A *critical pair* is a conflicting transformation pair $(t_1 : K \overset{r_1,m_1}{\Rightarrow} P_1, t_2 : K \overset{r_2,m_2}{\Rightarrow} P_2)$, where (m_1, m_2) are jointly surjective.

It has been shown that for each conflicting transformation pair there exists a critical pair that represents the same conflict in a minimal context. In particular, so-called extension diagrams are used as a technical means to embed a critical pair into a conflicting transformation pair. Note that the extension morphism f as well as the related vertical morphism f' used in the following extension diagram are not necessarily injective.

Definition 4 (Extension diagram)**.** An *extension diagram* is a diagram (1) as shown on the left of Fig. 7 where $f : G' \to G$ is a morphism, called *extension morphism*, and $t : G \overset{p}{\Longrightarrow} H$ as well as $t' : G' \overset{p}{\Longrightarrow} H'$ are two direct transformations via the same rule p with matches m' and $f \circ m'$ respectively, defined by the four pushouts in the middle of Fig. 7.

A transformation is actually extended by extending its context D' to D. Morphisms $f : G' \to G$ and $f' : H' \to H$ are the resulting pushout morphisms. This means that the context graph D' may be embedded into a larger one;

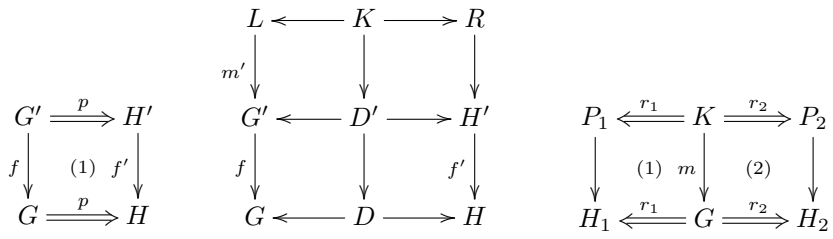


Figure 7: Extension diagram (overview and more detailed), extension diagram for transformation pair

additionally, elements of D may be glued together. Corresponding actions are reflected in f and f'; additional actions may not happen.

If a transformation pair can be embedded into another conflicting pair of transformations, then it inherits this conflict from the first transformation pair. The following lemma clarifies this inheritance.

Lemma 1 (Conflict inheritance)**.** *Given a conflicting transformation pair* $(t_1 : G \overset{r_1}{\Longrightarrow} H_1, t_2 : G \overset{r_2}{\Longrightarrow} H_2)$ *and another transformation pair* $(t_1' : G' \overset{r_1}{\Longrightarrow} H_1', t_2' : G' \overset{r_2}{\Longrightarrow} H_2')$ *that can be embedded into* (t_1, t_2) *via extension morphism* f *and corresponding extension diagrams as depicted in Fig. 8, then* (t_1', t_2') *is also a conflicting transformation pair.*

Proof. This follows from the conflict inheritance lemma in [15] for transformations with general matching. We in addition show that it holds for transformations restricted to injective matching (as assumed in this paper) as well: the matches of the embedded conflict $m_1', m_2' : L_1, L_2 \to G'$ are injective since it holds that $m_1 = f \circ m_1'$ and $m_2 = f \circ m_2'$ with the matches of the original pair of transformations $m_1, m_2 : L_1, L_2 \to G$ being injective. □

The embedded pair of transformations is a critical pair if it has jointly surjective matches. The following theorem states that such a pair exists indeed for each given conflicting transformation pair with extension morphism being injective [14]. In particular, this critical pair can be obtained from the pair

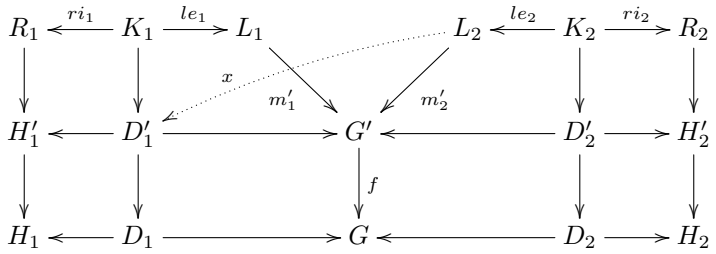


Figure 8: Conflict inheritance

of matches of the conflicting transformation pair via a unique so-called $\mathcal{E}'$-$\mathcal{M}$ pair factorization [15] in the more general context of $\mathcal{M}$-adhesive transformation systems [21, 14] and thus in particular for GTSs.[2]

Theorem 2 (Completeness Theorem for critical pairs)**.** *For each conflicting transformation pair* $(t_1 : G \stackrel{r_1}{\Longrightarrow} H_1, t_2 : G \stackrel{r_2}{\Longrightarrow} H_2)$ *there is a critical pair* $(t'_1 : K \stackrel{r_1}{\Longrightarrow} P_1, t'_2 : K \stackrel{r_2}{\Longrightarrow} P_2)$ *with extension diagrams (1) and (2) and* m *injective as depicted on the right of Fig. 7.*

Proof. This follows from the Completeness Theorem for critical pairs [14], where two direct transformations in conflict are considered independent of the order in which they occur, and Lemma 1 which guarantees that the critical pair given by the Completeness Theorem is conflicting also according to the asymmetric conflict definition (see Def. 2) used here. □

Initial conflicts [15] offer a more *declarative* view on minimal conflicting transformation pairs. In categorical terms, one can use actually the notion of *initial transformation pairs*, representing the "smallest" transformation pair that can be embedded into a given one, to obtain this new view on critical pairs. Initial conflicts are conflicting transformation pairs that coincide with their initial transformation pairs, since they represent already the "smallest" conflicts. In-

[2]Here, $\mathcal{E}'$ and $\mathcal{M}$ refer to a set of jointly surjective and a set of injective morphisms, respectively.

terestingly, it turns out that each initial conflict is a critical pair but not vice versa. By definition, initial conflicts have the important new characteristic that there exists a *unique* initial conflict for each given conflicting transformation pair. Additionally, all initial conflicts still satisfy the Completeness Theorem as well as the Local Confluence Theorem [15].[3] For the category of (typed) graphs, all these requirements hold [14, 15]. The Completeness Theorem states that for every conflict there exists an analogous initial conflict that can be embedded into it. The Local Confluence Theorem states that if all initial conflicts are strictly confluent, then the graph transformation system is locally confluent. Consequently, initial conflicts represent an important subset of critical pairs listing all conflict variants in a minimal context and with maximal unfolding. Restricting static conflict detection as well as local confluence analysis for graph transformation systems to initial conflicts might yield a considerable performance boost since a smaller set of critical pairs is computed. Moreover, the set of initial conflicts gives a better overview on conflict results.

In the following, we do not recall the categorical definition of initial conflicts [15], but simply reintroduce its set-theoretical characterization for the category of (typed) graphs. Note that recently also an alternative constructive characterization of initial conflicts for arbitrary set-valued functor categories has been shown [22]. The idea of initial conflicts is that they are critical pairs into which no different critical pair can be embedded. Otherwise speaking, it describes the "smallest" conflict that can be embedded into a given conflict. In particular, it has been shown [15] for the category of typed graphs that an initial conflict is a conflicting transformation pair with *minimal context* and *maximal unfolding of preserved elements* leading to the following characterization.

Definition 5 (Initial conflict [15])**.** A pair of transformations $ic : (G \overset{r_1,m_1}{\Longrightarrow} H_1, G \overset{r_2,m_2}{\Longrightarrow} H_2)$ for a pair of rules $(r_1 : (L_1 \overset{le_1}{\longleftarrow} K_1 \overset{ri_1}{\longrightarrow} R_1), r_2 : (L_2 \overset{le_2}{\longleftarrow} K_2 \overset{ri_2}{\longrightarrow} R_2))$ with deletion and boundary graphs C_i and B_i over the morphisms

[3] For Completeness of initial conflicts the existence of initial transformation pairs for conflicts is assumed. For Local Confluence, initial POs are required in addition.

$le_i : K_i \to L_i$ for $i = 1, 2$ (compare Fig. 6 for the asymmetric case) is an initial conflict if ic has the following properties:

1. Minimal context: m_1 and m_2 are jointly surjective.
2. At least one conflicting element, i.e. deleted by r_1 and used by r_2:
   $m_1(L_1) \cap m_2(L_2) \not\subseteq m_1(le_1(K_1))$.
3. Overlap in deletion graphs only:
   $m_1(L_1) \cap m_2(L_2) \subseteq (m_1(c_1(C_1)) \cap m_2(L_2)) \cup (m_1(L_1) \cap m_2(c_2(C_2)))$.
4. No isolated boundary node in overlap graph:
   $\forall x \in m_1(c_1(b_1(B_1))) \cap m_2(L_2) :$
   $\exists e \in m_1(c_1(C_1)) \cap m_2(L_2) : x = src(e) \vee x = tgt(e)$ and
   $\forall x \in m_2(c_2(b_2(B_2))) \cap m_1(L_1) :$
   $\exists e \in m_2(c_2(C_2)) \cap m_1(L_1) : x = src(e) \vee x = tgt(e)$.

Note that the first two items of the above characterization describe critical pairs for a given conflicting rule pair (r_1, r_2). The first three items characterize essential critical pairs as Item 3 follows directly from their construction [19]. In the category of (typed) graphs a critical pair is essential if two injective matches overlap in deleted elements and boundary nodes only [19]. Item 4 is an additional condition that needs to hold for initial conflicts ensuring that no further unfolding of elements is possible. Note that we adapted slightly the characterization of initial conflicts compared to their introduction in [15], since we are interested in the asymmetric case of a delete-use-conflict (according to Def. 2) in this paper only. In particular, we adapted Item 2 to the asymmetric case.

Each initial conflict is a critical pair, but not vice versa. This means that, if preserved elements are not unfolded maximally, we do not have an initial conflict.

Theorem 3. *Each initial conflict* $ic : (G \overset{r_1,m_1}{\Longrightarrow} H_1, G \overset{r_2,m_2}{\Longrightarrow} H_2)$ *is a critical pair.*

Proof. Since Item 1 and 2 of Def. 5 characterize critical pairs set-theoretically, this follows directly from Def. 3 and Def. 5. □

Example 2 (Initial conflict)**.** The left overlap graph in Fig. 4 can function as graph G of an initial transformation pair applying the refactoring rules in Fig. 2. These applications show minimal context and minimal overlaps of LHSs just as much as needed to form a conflict. This means that an initial conflict is shown.

The right overlap graph shows another conflict, here each two methods are overlapped. In addition, 7,17:Class and 4,12:Attribute show overlaps of two preserved classes. Node 4:Attribute is neither deleted nor a boundary node (and therefore not an element of graph C_1) but overlapped with node 12:Attribute from the LHS of rule *pullUpEncapsulatedAttribute* so that Item 3 in Def. 5 is not fulfilled. Node 7:Class is a boundary node since incoming adjacent edge type is deleted, but that edge is not overlapped with the corresponding edge type to node 17:Class so that Item 4 in Def. 5 is not fulfilled. Hence, the shown conflict is not initial. But it still depicts a conflict in a minimal context, which means that a critical pair is shown.

Similarly to the Completeness Theorem for critical pairs, it is possible to formulate a Completeness Theorem for initial conflicts. The difference is that initial conflicts are embedded into a conflict via an extension morphism that does not need to be injective.[4]

Theorem 4 (Completeness Theorem for initial conflicts)**.** *For each conflicting transformation pair* $(t_1 : G \overset{p_1}{\Longrightarrow} H_1, t_2 : G \overset{p_2}{\Longrightarrow} H_2)$*, there is an initial conflict* $(t_1^I : K \overset{r_1}{\Longrightarrow} P_1, t_2^I : K \overset{r_2}{\Longrightarrow} P_2)$ *with extension diagrams (1) and (2) as depicted on the right of Fig. 7.*

Proof. This follows from the Completeness Theorem for initial conflicts [15], where two direct transformations in conflict are considered independent of the order in which they occur, and Lemma 1 which guarantees that the initial conflict given by the Completeness Theorem is in conflict also according to the

[4]Therefore critical pairs are also called $\mathcal{M}$-initial conflicts [15], since as opposed to regular initial conflicts the extension morphism in the Completeness Theorem for critical pairs is injective.

asymmetric conflict definition (see Def. 2) used here. □

Finally, note that since a critical pair is a conflicting transformation pair, it is possible to find a unique initial conflict that can be embedded into this critical pair. In particular, this extension morphism will be surjective since both, the critical pair as well as the initial conflict, have jointly surjective match morphisms already.

4. Two-Dimensional Approach to Granularity of Conflicts and Dependencies

In this section, we present our two-dimensional granularity approach to conflicts and dependencies. We distinguish conflict notions with varying context and overlap granularity. Our overall intention from a practical point of view is the possibility to gradually introduce users to conflicts by showing them conflict notions belonging to a specific context or overlap granularity level. Our overall intention from a theoretical point of view is to classify all different conflict notions in a descriptive way according to their context and overlap granularity level and find interesting relationships between all notions varying these granularity dimensions.

As first contribution, in Sect. 4.1 we introduce the two granularity dimensions (overlap as well as context granularity) and specific levels therein (binary, coarse, medium, fine) along which we want to classify all existing and new conflict notions as introduced in this paper. We classify and relate the *existing conflict notions* recalled in Sect. 3 within our two-dimensional granularity approach in Sect. 4.2. The existing conflict notions as reintroduced in the previous section will be of binary (conflicting rule pairs), medium (critical pairs) or fine (conflicting transformation pair) context granularity, in particular. Thereafter, in Sect. 4.3 we start investigating more closely *new conflict notions* on another level of context granularity in between the binary and medium one, called *coarse*. We connect the *new conflict notions* with *coarse context granularity* with the *already known notions* of initial conflicts and classical critical

pairs, having medium context granularity in Sect. 4.4. We show the *interrelations of the new conflict notions* with coarse context granularity in Sect. 4.5 and we introduce *coarse overlap granularity* as a new level of overlap granularity in between the binary and medium one. We connect the new notions with the binary level in Sect. 4.6. Finally, we sketch dual notions for dependencies.

4.1. Two dimensions of granularity

In this paper, we consider existing and new conflict notions describing conflicting transformations with more or less detail.

Context granularity. Our conflict notions shall be able to express conflicts with a varying amount of context. In the previous section, we have considered four different conflict notions already: conflicting rule pair, conflicting transformation pair, initial conflict, and critical pair. Since *rule pairs* do not show any context for the conflicting transformation pairs related to this rule pair, we consider them to be of *binary context granularity*. On the contrary, *conflicting transformation pairs* show conflicts with their complete context such that we classify them to be of *fine context granularity* (see Fig. 1). Critical pairs, and in particular also initial conflicts, show conflicts with minimal context (via jointly surjective matches); we consider them therefore to be of *medium context granularity*. The new conflict notions such as conflict atom, conflict reason and conflict reason extension do not show concrete matches into a graph for which related conflicts occur. They are merely based on spans between the rule's LHSs (or their deletion graphs) indicating how the matches in related conflicting transformations will need to overlap certain elements. We consider them therefore to be of *coarse context granularity*. These observations are summarized in the following description list of context granularity levels.

Given a *conflicting rule pair* (r_1, r_2) with $r_1 : L_1 \overset{le_1}{\hookleftarrow} K_1 \overset{ri_1}{\hookrightarrow} R_1$ and $r_2 : L_2 \overset{le_2}{\hookleftarrow} K_2 \overset{ri_2}{\hookrightarrow} R_2$, we consider the following *context granularity levels*:

- The *context granularity* of the conflicting rule pair (r_1, r_2) is *binary*.

- Given a conflict notion consisting of a span $C_1 \hookleftarrow S_1 \to L_2$ with C_1 the deletion graph of r_1 or span $L_1 \hookleftarrow S \to L_2$, then the *context granularity* of this conflict notion is *coarse*.

- Given a *conflicting transformation pair* $(t_1, t_2) = (G \overset{m_1,r_1}{\Longrightarrow} H_1, G \overset{m_2,r_2}{\Longrightarrow} H_2)$ via rules (r_1, r_2), then the *context granularity* of (t_1, t_2) is *medium* if the pair of matches (m_1, m_2) is *jointly surjective*.

- Given a *conflicting transformation pair* $(t_1, t_2) = (G \overset{m_1,r_1}{\Longrightarrow} H_1, G \overset{m_2,r_2}{\Longrightarrow} H_2)$ via rules (r_1, r_2), the *context granularity* of (t_1, t_2) is *fine* if the pair of matches (m_1, m_2) is not jointly surjective.

In this order, granularity levels show an increasing amount of context: The binary level does not show any context of conflicts arising from rule pairs, the coarse level shows conflicts in the form of spans, the medium level shows a conflict in a minimal context represented by rule overlaps, and the fine level allows context additional to the minimal one.

Overlap granularity. Our conflict notions shall be able to show a varying amount of overlap between rule pairs that leads to conflicts between applications of these rules. Given a conflicting transformation pair, an *overlap* is a specific span between the LHSs of the corresponding conflicting rule pair. This span is derived for each conflict notion reviewed or introduced in this paper according to the following considerations. Overlaps for conflict notions express via this span which elements of both LHSs are mapped to the same element. A conflicting rule pair shows no overlap information at all such that we will consider its overlap to be empty. Conflicting transformation pairs on the other hand describe via their matches which elements from both LHSs are mapped identically. Therefore we can summarize these elements into a span, resprenting the overlap, by building the pullback of the matches. For conflict notions already consisting of spans it is more straightforward to derive their overlap. We summarize these observations in the following description list of overlaps of different conflict notions.

Given a *conflicting rule pair* (r_1, r_2) with $r_1 : L_1 \overset{le_1}{\hookleftarrow} K_1 \overset{ri_1}{\hookrightarrow} R_1$ and $r_2 : L_2 \overset{le_2}{\hookleftarrow} K_2 \overset{ri_2}{\hookrightarrow} R_2$, we characterize the following kinds of overlaps:

- The *overlap* of a *rule pair* (r_1, r_2) is the empty span $L_1 \hookleftarrow \emptyset \to L_2$.
- The *overlap* of a conflict notion given by a span $C_1 \overset{a_1}{\hookleftarrow} S_1 \overset{b_2}{\to} L_2$ with C_1 being the deletion graph of r_1 is the span $L_1 \overset{c_1 \circ a_1}{\hookleftarrow} S_1 \overset{b_2}{\to} L_2$ with c_1 being the embedding of the deletion graph C_1 into L_1.
- The *overlap* of a conflict notion given by a span $L_1 \hookleftarrow S \to L_2$ is this span.
- The *overlap* of a *conflicting transformation pair* $(t_1, t_2) = (G \overset{m_1, r_1}{\Longrightarrow} H_1, G \overset{m_2, r_2}{\Longrightarrow} H_2)$ is the span $L_1 \leftarrow S \to L_2$ being the pullback of matches (m_1, m_2).

Conflict notions related to the same conflicting transformation pair become comparable via their overlaps if there exists an embedding of one overlap into the other one. Such an embedding defines a relation between overlaps as follows: Given two overlaps $s' : L_1 \overset{a'_1}{\hookleftarrow} S' \overset{b'_2}{\to} L_2$ and $s : L_1 \overset{a_1}{\hookleftarrow} S \overset{b_2}{\to} L_2$, overlap s' is *embeddable* into overlap s if there exists an injective morphism $e : S' \to S$ such that $a_1 \circ e = a'_1$ and $b_2 \circ e = b'_2$.

In the following, we will explain why the relation between overlaps of conflict notions corresponding to a given conflicting transformation pair induces a partial ordering on these notions. This allows us to consider four specific *overlap granularity* levels (analogous to context granularity): fine, medium, coarse, and binary. Conflict notions classified into these levels become less detailed w.r.t. overlap granularity when moving from fine over medium and coarse to binary, respectively.

4.2. Classifying Existing Conflict Notions

From our considerations above it follows immediately that rule pairs have binary *context level*, conflicting transformation pairs not being critical pairs have fine context level, critical pairs have medium context level.

Let us introduce *overlap granularity* of the reviewed conflict notions in the following. We consider *conflicting rule pairs* to have *binary* overlap granularity, since they do not show any detail about overlapping elements for a conflicting transformation pair via these rules. On the other hand, we consider *conflicting transformation pairs* to have *fine* overlap granularity. The overlap of a conflicting pair describes completely which elements from both LHSs are matched identically. In Sect. 3, we have recalled the relationships between conflicting transformation and rule pairs as well as between initial conflicts and critical pairs. For each conflicting transformation pair there exists a unique critical pair embeddable into it via an injective extension morphism (see Theorem 2). Both this critical pair as well as the initial conflict for a conflicting transformation pair represent the given conflict on medium context granularity level, but the extension morphism is general for an initial conflict whereas it is injective for a critical pair. This means that the match morphisms of a critical pair glue all elements that are glued by the match morphisms of the conflicting transformation pair. On the contrary, the match morphisms of an initial conflict do not glue elements if this is not necessary for the conflict to be reproduced. Hence, the conflicting transformation pair and its critical pair have the same overlap and therefore overlap granularity, whereas the *overlap granularity* of an *initial conflict* is in general coarser. We consider it to be of *medium overlap granularity level*. The *relationships between existing conflict notions of medium and fine context granularity* are summarized in the lower triangle of Fig. 1.

4.3. Conflict Notions of Coarse Context Granularity

We introduce new conflict notions of *coarse context granularity* by lifting our conflict considerations from transformations to the rule level, i.e., we consider conflicting rule pairs. A rule pair is conflicting if there is a conflicting transformation pair applying the respective rules. We thus largely abstract from the context in which the conflict occurs, but not entirely. Starting from the already known binary description of conflicting rule pairs with binary context granularity, we refine it to conflict notions with coarse context granularity. In

particular, we concentrate on certain spans (see Fig. 6) between rules specifying the conflict reasons (or at least parts of it) for conflicting transformation pairs via these rules; we distinguish several forms of spans showing conflict reasons with *varying overlap granularity*. The idea of considering such spans as conflict reasons for transformations originates from the work on characterizing delete-conflicts (see reintroduced Theorem 1) and so-called essential critical pairs [19]. We will start with introducing *conflict atoms* between rules being of *coarse overlap granularity*. Staying on this coarse context granularity level, more overlap information can be gradually added yielding *conflict reasons* being of *medium overlap granularity* or even *conflict reason extensions* being of *fine overlap granularity*. Adding more context again, we leave the coarse context granularity level and arrive at the already known notion of initial conflicts and more generally, critical pairs both having medium context granularity. We will show the *interrelations of the new conflict notions* in Sect. 4.5, where it becomes clear why a conflict atom specifies less overlapping elements than a conflict reason or conflict reason extension because of embedding relations in between them, respectively.

In case of a conflict 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 graph C_1 and the LHS of the second rule (Fig. 6).

We start focusing on minimal building bricks, called conflict atoms, overlapping as few elements as possible for a conflict to be caused. 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 atoms being involved in the reported conflict are subsumed by it (*completeness condition*) and thus overlapped. The *conflict condition* for the conflict reason ensures moreover that conflicting elements are overlapped *only*. In more details, this means that each graph component in the overlap S_1 of graphs C_1 and L_2 has to fulfill the weak conflict condition, i.e., isolated boundary nodes are not allowed in S_1. In particular, the conflict condition ensures that each "part" of S_1 fulfills the weak conflict condition and is therefore formulated based on constructing all possible coproducts.[5] While conflict reasons overlap in conflicting graph elements (and adjacent boundary nodes) only, conflict reason extensions may overlap in non-conflicting elements of the LHSs of participating rules as well (*extended completeness condition*).

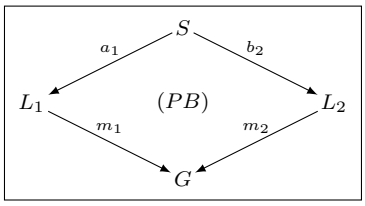
Definition 6 (Overlap conditions)**.** Given rules $r_1 : L_1 \overset{le_1}{\hookleftarrow} K_1 \overset{ri_1}{\hookrightarrow} R_1$ and $r_2 : L_2 \overset{le_2}{\hookleftarrow} K_2 \overset{ri_2}{\hookrightarrow} R_2$ with the initial pushout (1) for $K_1 \overset{le_1}{\hookrightarrow} L_1$ as well as a span $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\to} L_2$ as depicted in Fig. 6, *overlap conditions* for the span s_1 of (r_1, r_2) are defined as follows:

1. *Weak conflict condition*: Span s_1 satisfies the *weak conflict condition* if there does not exist any injective morphism $x : S_1 \to B_1$ such that $b_1 \circ x = o_1$.
2. *Conflict condition*: Span s_1 satisfies the *conflict condition* if for each coproduct $\bigoplus_{i \in I} S_1^i$, where each S_1^i is non-empty and $S_1 = \bigoplus_{i \in I} S_1^i$, each of the induced spans $s_1^i : C_1 \overset{o_1^i}{\hookleftarrow} S_1^i \overset{q_{12}^i}{\to} L_2$ with $o_1^i = o_1|_{S_1^i}$ and $q_{12}^i = q_{12}|_{S_1^i}$ fulfills the weak conflict condition.
3. *Transformation condition*: Span s_1 satisfies the *transformation condition* if there is a pair of transformations $(t_1, t_2) = (G \overset{m_1, r_1}{\Longrightarrow} H_1, G \overset{m_2, r_2}{\Longrightarrow} H_2)$ 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. 6).

[5] In the category of (typed) graphs, an easier definition of the new conflict condition is possible; instead of considering all possible coproducts, it suffices to consider only the one decomposing a graph into its connected components.

4. *Completeness condition*: Span s_1 satisfies the *completeness condition* if there is a pair of transformations $(t_1, t_2) = (G \overset{m_1,r_1}{\Longrightarrow} H_1, G \overset{m_2,r_2}{\Longrightarrow} H_2)$ via (r_1, r_2) such that (2) is the pullback of $(m_1 \circ c_1, m_2)$ in Fig. 6.
5. *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. 9.) 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 $(t_1, t_2) = (G \overset{m_1,r_1}{\Longrightarrow} H_1, G \overset{m_2,r_2}{\Longrightarrow} H_2)$ via (r_1, r_2) such that s arises from the pullback of (m_1, m_2) in the figure on the right.



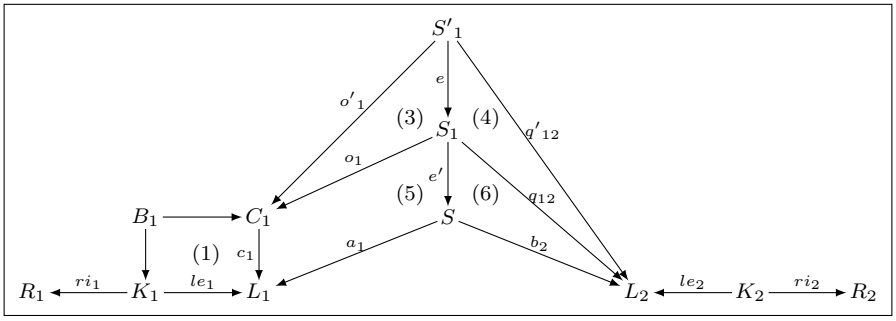


Figure 9: Illustrating span embeddings

Example 3 (Overlap conditions)**.** Figure 10 shows the essential part of a conflicting transformation pair applying simplified refactoring rules *decapsulateAttribute* and *pullUpEncapsulatedAttribute* at graph G1 already shown in Fig. 4.

The weak conflict condition is fulfilled since S1 cannot be embedded into B1. The only coproduct of graphs yielding S1 just consists of S1 itself. Therefore,

the general conflict condition is also fulfilled. Since m1 and m2 are matches for their corresponding rules, there is a pair of transformations. Furthermore, diagram (2) is commuting which means that the transformation condition is satisfied. Actually, S1 is the intersection of C1 and L2 in G1 which means that the completeness condition is also fulfilled. Finally, we can state that any smaller graph than S1 would yield a setting where at least one of the conditions discussed above is violated. Hence, assuming that all the conditions discussed are fulfilled, the minimality condition is also satisfied.

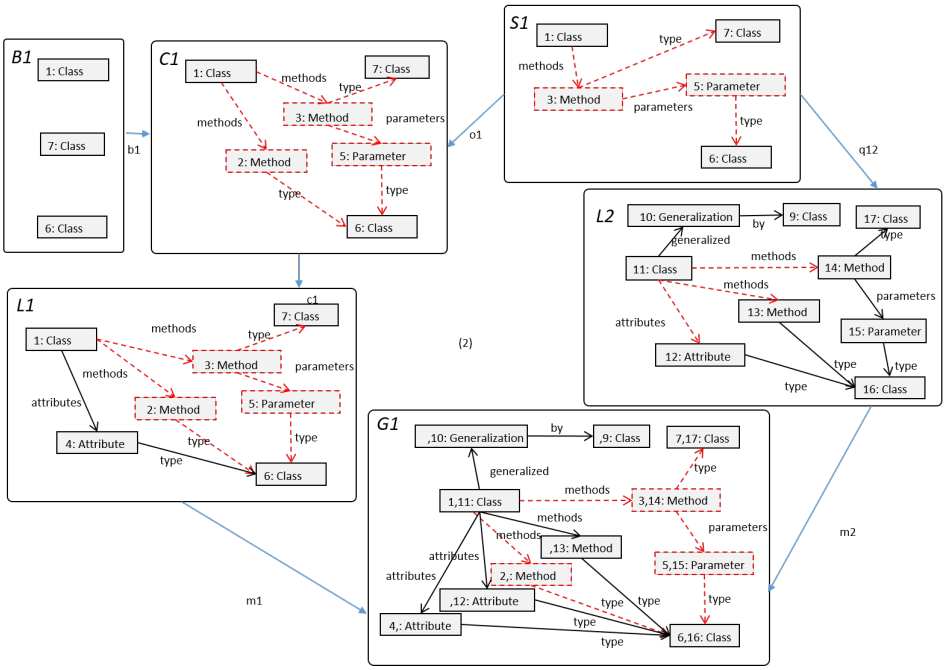


Figure 10: Illustrating overlap conditions on example conflict for simplified refactoring rules *decapsulateAttribute* and *pullUpEncapsulatedAttribute*

In [17], the above introduced weak conflict condition served as conflict condition. As already mentioned above, in the case of graphs the strengthening of this condition ensures that the graph S_1 of a span fulfilling the conflict condition does not contain isolated boundary nodes. These are isolated nodes of S_1 (i.e., without adjacent edge) that get nevertheless mapped by o_1 to a node in C_1

which has a preimage under b_1. This exclusion of isolated boundary nodes from graph S_1 leads to a better comparability with the concept of initial conflicts and proved to lead to much more elegant results (compare, e.g., Cor. 1). The following lemma characterizes the conflict condition in a set-theoretical way.

Lemma 2 (Conflict condition characterization)**.** *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* $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\to} L_2$ *as depicted in Fig. 6 fulfilling the weak conflict condition, then* s_1 *fulfills the conflict condition iff* S_1 *does not contain any isolated boundary nodes.*

Proof. If S_1 contained an isolated boundary node v, i.e., a node that has no adjacent edge but is mapped by o_1 to a node in C_1 with preimage v' in B_1, then $S_1 = \{v\} \oplus (S_1 \setminus \{v\})$ and $x : \{v\} \to B_1$ may be defined as mapping v to v'. By construction $b_1 \circ x = o_1|_{\{v\}}$, so that the conflict condition is violated.

Let span s_1 fulfill the weak conflict condition but not the conflict condition. Then there exists a coproduct $\bigoplus_{i \in I} S_1^i = S_1$ where each S_1^i is non-empty and (at least) one component $S_1^j, j \in I$ can be embedded into B_1 via an injective morphism $x : S_1^j \to B_1$ such that $b_1 \circ x = o_1|_{S_1^j}$. Since there exists a morphism from S_1^j to B_1, the graph S_1^j consists of nodes only. Since S_1^j is a summand of the coproduct, S_1 does not contain an edge between two nodes, one being part of S_1^j and the other being not. Since $b_1 \circ x = o_1|_{S_1^j}$, S_1^j consists of isolated boundary nodes (having no adjacent edges, but preimages under b_1). □

In the following, we define the new building bricks of conflicts with coarse context granularity, but gradually enforcing one overlap condition after the other (i.e. varying the overlap granularity from coarse to fine).

Definition 7 (Conflict notions with coarse context granularity)**.** 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 $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\to} L_2$ as depicted in Fig. 6, 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 $s : L_1 \overset{a_1}{\hookleftarrow} S \overset{b_2}{\rightarrow} L_2$ 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 $e' : S_1 \rightarrow S$ a so-called embedding morphism being injective such that (5) and (6) in Fig. 9 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 4 (Conflict notions with coarse context granularity)**.** Graphs in Fig. 3 refer to the rule pair in Fig. 2 and compactly denote spans indicated by equal numbers in span and rule graphs.

- All depicted spans fulfill the conflict condition since they show overlaps in elements to be deleted (*weak conflict condition*) without including overlaps in isolated preserved nodes (full *conflict condition*).

  Considering the span in Fig. 11, the weak conflict condition holds since elements to be deleted take part in the overlap. The general conflict condition, however, is not fulfilled since 7,9:Class does not fulfill the weak conflict condition. Actually, it is an isolated boundary node.

- Figure 4 shows two graphs, which both can function as graph G in the *transformation condition* to which the rules are applied. Both rule embed-

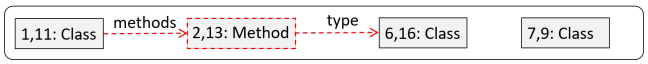


Figure 11: Span over rule pair (*decapsulateAttribute*, *pullUpEncapsulatedAttribute*) with isolated boundary node

dings (again indicated by numbers) are valid rule matches, i.e, fulfill the gluing condition. For three of the left (compactly denoted) spans in Fig. 3 (all except 3,13:Method) graphs G1 or G2 form overlap graphs such that the transformation condition is fulfilled. (There is also one for 3,13:Method not depicted in this paper.) All left spans in Fig. 3 are *conflict atoms* since they are minimal in addition. (The empty graph never fulfills the weak conflict condition.)

Considering the span encoded by 2,14:Method we can first of all state that it is a *conflict atom candidate* since the conflict condition is satisfied. The transformation condition, however, cannot be fulfilled since overlapping the LHSs of both rules along this span would lead to a dangling parameters-edge. Hence, rule *decapsulateAttribute* cannot be applied in this setting.

- The right (compactly denoted) spans in Fig. 3 fulfill the *completeness condition* since there are transformations applying the conflicting rules such that C1 and L2 are overlapped as in the depicted spans. Moreover, each one of them fulfills the conflict and the transformation condition. Hence, they are *conflict reasons*. One example illustrating the fulfilled completeness (as well as the transformation condition) is the left overlap graph in Fig. 4; the intersection of C_1 and L_2 in it is the conflict reason at the bottom of Fig. 3. We get a further *conflict reason* by gluing the top most conflict reason of Fig. 3 with the bottom one at commonly named nodes.

- In addition, all three conflict reasons in Fig. 3 fulfill the *minimality condition*, i.e., they are *minimal conflict reasons*. There cannot be a smaller span fulfilling the completeness condition since all their elements to be

deleted are connected. In addition, there are boundary nodes only. The pullback over any smaller (non-empty) span would lead to an overlap graph such that an application of the left rule would cause dangling edges. However, these smaller spans represent conflict parts since there are transformation pairs they can be embedded into. Additional conflict reasons that are constructed from these three are not minimal since they include the minimal ones.

- The compactly notated spans in the right part of Fig. 3 and the one in Fig. 11 are also *conflict reason extensions* if one interprets them as embedded into L_1 and L_2 (instead of C_1 and L_2). The overlap graphs G1 and G2 in Fig. 4 with the implied matches, for instance, ensure that the *extended completeness condition* for the last span in the right part of Fig. 3 and the span in Fig. 11, respectively, is satisfied. In contrast to conflict reasons, a conflict reason extension may also contain additional overlappings such as 4,12:Attribute and adjacent edges of type variables in Fig. 12. Such an overlapping is not needed to characterize a conflict but it does not hurt.

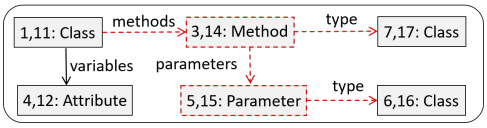


Figure 12: Conflict reason extension of rule pair (*decapsulateAttribute*, *pullUpEncapsulatedAttribute*)

Table 2 provides an overview over the new conflict notions for rules (i.e., with coarse context granularity) and their overlap conditions illustrating the variety in overlap granularity.

We end this section by showing a first rough relationship between a conflicting transformation pair (of medium or fine context granularity) and a conflict part being of coarse context granularity. This result can be considered as a kind of quality criterion for the definition of new conflict notions above since it

Overlap condition / conflict notion	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 2: Overview of conflict notions with coarse context granularity

establishes a relation of these new conflict notions to the classical one in Def. 2. For future reference, we prove this theorem in a way that then can be repeated analogously for finitary $\mathcal{M}$-adhesive categories with $\mathcal{M}$-initial object[6] such that the relationship holds for this more general context as well. However, for graphs the result is true for infinite instances also.

Theorem 5 (Conflicting transformation and conflict part)**. Restriction** *Given a conflicting transformation pair* $(t_1, t_2) = (G \overset{m_1,r_1}{\Longrightarrow} H_1, G \overset{m_2,r_2}{\Longrightarrow} H_2)$ *then there exists a conflict part* $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\to} L_2$ *for* (r_1, r_2) *with (2) in Fig. 6 commuting.*

Extension *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, r_2) *then there exists a conflicting transformation pair* $(t_1, t_2) = (G \overset{m_1,r_1}{\Longrightarrow} H_1, G \overset{m_2,r_2}{\Longrightarrow} H_2)$ *with (2) in Fig. 6 commuting.*

Proof. **Restriction:** Build the pullback of $(m_1 \circ c_1, m_2)$. According to Theorem 1 the arising span $s_1^{\{1\}} : C_1 \overset{o_1^{\{1\}}}{\hookleftarrow} S_1^{\{1\}} \overset{q_{12}^{\{1\}}}{\to} L_2$ fulfills the weak conflict

[6]Finitary $\mathcal{M}$-adhesive categories [23] describe $\mathcal{M}$-adhesive categories with finite objects only. The existence of an $\mathcal{M}$-initial object implies finite coproducts with injections into the coproduct being in $\mathcal{M}$, an assumption that this proof relies on and which holds for the category of finite graphs, in particular.

condition. Suppose that it does not fulfill the conflict condition yet. Then there exists a coproduct summing up to $S_1^{\{1\}}$ with at least one of its components not fulfilling the weak conflict condition. Consequently, we can construct a new span $s_1^{\{2\}} : C_1 \overset{o_1^{\{2\}}}{\hookleftarrow} S_1^{\{2\}} \overset{q_{12}^{\{2\}}}{\rightarrow} L_2$ by omitting the components not fulfilling the weak conflict condition from the coproduct and restricting the morphisms accordingly. This procedure can be repeated as long as the conflict condition is not fulfilled; the newly arising spans $s_1^{\{k\}}$ always inherit the weak conflict condition and the transformation condition from the span originally constructed via the pullback construction. Since at every repetition there is only a finite number of possibilities to partition the graph $S_1^{\{k\}}$ as coproduct and since with every repetition the number of possibilities decreases, this procedure terminates. Assume it terminates with the empty graph as result. Then summing up the deleted parts again, on the one hand results in the original span $s_1^{\{1\}}$, but on the other hand provides a morphism $x : S_1 \rightarrow B_1$ with $b_1 \circ x = o_1$ (sum up the morphisms that caused the violation of the conflict condition at the different steps of the repetition). Since this contradicts $s_1^{\{1\}}$ fulfilling the weak conflict condition, the procedure terminates with a span $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\rightarrow} L_2$ fulfilling the transformation and the conflict condition.

Extension: Because of the transformation condition there exists a pair of transformations $(t_1, t_2) = (G \overset{m_1, r_1}{\Longrightarrow} H_1, G \overset{m_2, r_2}{\Longrightarrow} H_2)$ with (2) in Fig. 6 commuting. We can now build the pullback of $(m_1 \circ c_1, m_2)$. The span arising from this pullback fulfills the weak conflict condition, since otherwise s_1 would not fulfill the weak conflict condition. Therefore because of Theorem 1, we can conclude that (t_1, t_2) is conflicting. □

4.4. Relating Conflict Notions of Coarse and Medium Context Granularity

As illustrated in Fig. 1 and recalled in Sect. 3, the notions of critical pair and initial conflict express conflicts of medium context granularity (i.e., the context only entails elements stemming from the rules). In this section, we show the interrelation of critical pairs (and initial conflicts) with our newly introduced notions of coarse context granularity. In particular, we can establish a 1-1 re-

lationship between critical pairs and conflict reason extensions for a given rule pair. For each critical pair we can determine its overlap, corresponding to a conflict reason extension with the same overlap. Therefore we consider this *conflict reason extension* to have *fine overlap granularity* such as the critical pair itself. Moreover, there is a relationship between initial conflicts and conflict reasons for a given rule pair, where their overlap granularity remains almost the same. This means that an initial conflict overlaps the same elements as prescribed by the embedded conflict reason, but it might overlap some additional elements. This happens if the initial conflict is not only a delete-use, but also a use-delete conflict (i.e., the second transformation in the conflicting transformation pair deletes elements used by the first one).[7] We consider this difference to be marginal in this framework such that we assign both to initial conflicts as well as conflict reasons the *medium granularity level*. The established relationships are illustrated by the vertical arrows between the top and medium row of the conflict notion overview in Fig. 1. The relationships are described more precisely in the following two theorems.

Theorem 6 (Initial conflict and conflict reason)**. Restriction.** *Given an initial conflict* $(t_1, t_2) = (K \overset{m_1,r_1}{\Longrightarrow} P_1, K \overset{m_2,r_2}{\Longrightarrow} P_2)$*, 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)*, there exists an initial conflict* $(t_1, t_2) = (K \overset{m_1,r_1}{\Longrightarrow} P_1, K \overset{m_2,r_2}{\Longrightarrow} P_2)$ *with the pullback of* $(m_1 \circ c_1, m_2)$ *being isomorphic to* s_1*.*

Proof. **Restriction:** The completeness and transformation condition for the span s_1 are fulfilled by construction and Theorem 1 states that the weak conflict

[7] This phenomenon arises from the fact that, in this work, we opted for defining conflict reasons for a conflicting transformation *pair* asymmetrically (i.e., concentrating on only one order of rules). On the contrary, the notion of conflict reasons was originally introduced for conflicting transformations in a symmetrical form in [19].

condition also holds. Assume the conflict condition not to hold. Because of Lemma 2, this implies the existence of an isolated boundary node $v \in S_1$. Its image $m_1(c_1(b_1(x(v)))) \in K$ is an isolated boundary node in the sense of Def. 5, Item 4. That is a contradiction to (t_1, t_2) being an initial conflict.

Extension: Compare Fig. 13 for the following proof: Since s_1 fulfills the completeness condition, there exists a pair of transformations $(t_1', t_2') = (G \overset{m_1', r_1}{\Longrightarrow} H_1, G \overset{m_2', r_2}{\Longrightarrow} H_2)$ with s_1 arising from the pullback of $(m_1' \circ c_1, m_2')$. Because of Theorem 5 and the fact that s_1 fulfills the conflict condition we know that (t_1', t_2') is also conflicting. Due to Theorem 4, for each such conflicting transformation pair, we can build the initial conflict $(t_1, t_2) = (K \overset{m_1, r_1}{\Longrightarrow} P_1, K \overset{m_2, r_2}{\Longrightarrow} P_2)$ for (t_1', t_2') with an extension morphism $m : K \to G$ s.t. $m \circ m_1 = m_1'$ and $m \circ m_2 = m_2'$. Now we show that s_1 is also a pullback of $(m_1 \circ c_1, m_2)$. Assume a graph X and morphisms $x_1 : X \to C_1$ and $x_2 : X \to L_2$ s.t. $m_1 \circ c_1 \circ x_1 = m_2 \circ x_2$. Then, because s_1 is also a pullback of $(m_1' \circ c_1, m_2')$ we can use its pullback property and conclude that since $m_1' \circ c_1 \circ x_1 = m \circ m_1 \circ c_1 \circ x_1 = m \circ m_2 \circ x_2 = m_2' \circ x_2$, it holds that there exists a unique morphism $x : X \to S_1$ s.t. $o_1 \circ x = x_1$ and $q_{12} \circ x = x_2$. □

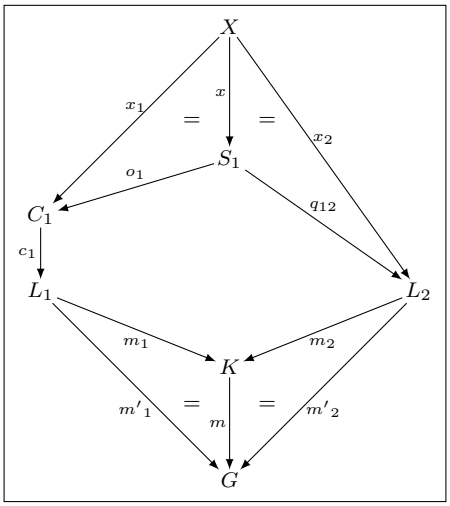


Figure 13: Illustrating the proof of the Extension Case of Theorem 6

The following theorem describes the 1-1 relationship between critical pairs and conflict reason extensions. Both have the same overlap granularity, but critical pairs show more context than conflict reason extensions. In particular, critical pairs show conflicts in their minimal context and conflict reason extensions merely show how the rules' LHSs overlap in order for such conflicts to occur.

Theorem 7 (Critical pair and conflict reason extension)**. Restriction.** *Given a critical pair* $(t_1, t_2) = (K \overset{m_1,r_1}{\Longrightarrow} P_1, K \overset{m_2,r_2}{\Longrightarrow} P_2)$ *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* $s : L_1 \overset{a_1}{\hookleftarrow} S \overset{b_2}{\rightarrow} L_2$ *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* $(t_1, t_2) = (K \overset{m_1,r_1}{\Longrightarrow} P_1, K \overset{m_2,r_2}{\Longrightarrow} P_2)$*.*

Bijective correspondence *The restriction and extension constructions are inverse to each other up to isomorphism.*

Proof. **Restriction:** It is obvious that, by taking the pullback (1) of (m_1, m_2), the extended completeness condition is fulfilled. Because of Theorem 4 we know that an initial conflict can be embedded into this critical pair. Due to Theorem 6 we know that this initial conflict can be restricted to a conflict reason $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\rightarrow} L_2$ arising from taking the pullback of $(m'_1 \circ c_1, m'_2)$ with (m'_1, m'_2) being the matches of the initial conflict. In particular, s_1 can be embedded into the conflict reason extension s via some morphism e'.

Extension: We get by definition of conflict reason extension two transformations $(t'_1, t'_2) = (G \overset{m'_1,r_1}{\Longrightarrow} H_1, G \overset{m'_2,r_2}{\Longrightarrow} H_2)$ s.t. s is the span arising from building the pullback (2) of (m'_1, m'_2). By Theorem 2 we know that we get a critical pair that can be embedded via some injective extension morphism into $(t'_1, t'_2) = (G \overset{m'_1,r_1}{\Longrightarrow} H_1, G \overset{m'_2,r_2}{\Longrightarrow} H_2)$. The diagram consisting of s together with the matches (m_1, m_2) of this critical pair is also a PB. The PB property is inherited from pullback (2). Because of Remark 2.25 in [14] a PB of two injective,

jointly surjective morphisms is also a pushout (PO).

Bijective correspondence: The constructions have a bijective correspondence since, in the category of typed graphs, a PO over at least one injective morphism (morphism in $\mathcal{M}$, resp.) is also a PB [14]. Moreover, because of Remark 2.25 in [14], a PB being built from two injective, jointly surjective morphisms is also a PO. Last but not least, POs and PBs are unique up to isomorphism. □

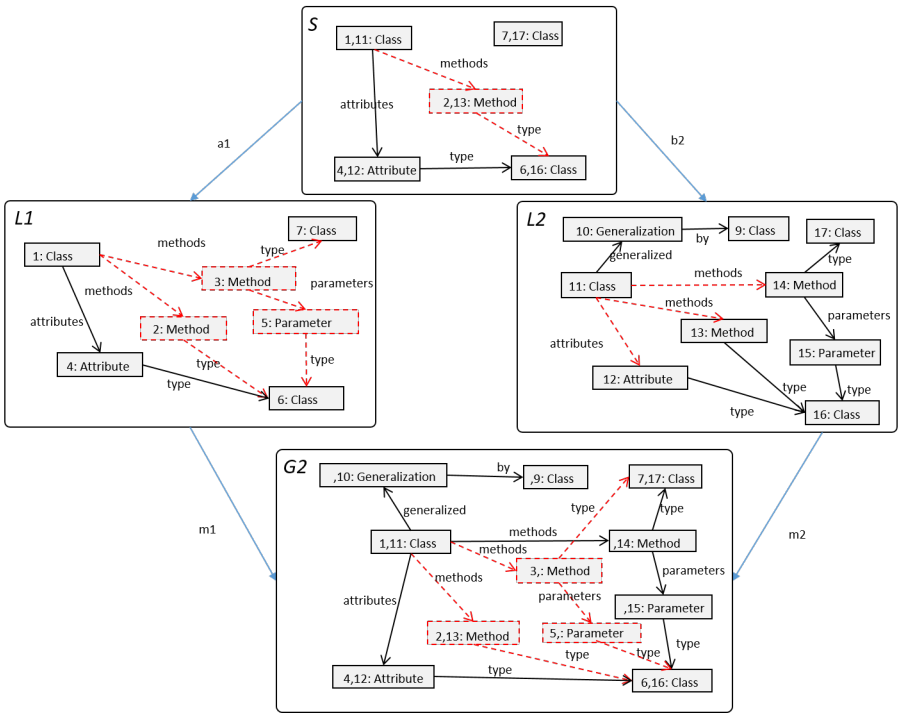


Figure 14: Example of a conflict reason extension for the pair of simplified refactoring rules *decapsulateAttribute* and *pullUpEncapsulatedAttribute*

Example 5 (Critical pair and conflict reason extension)**.** Fig. 14 shows the overlap graph G2 already shown in Fig. 4. It overlaps not only 2:Method with 13:Method and adjacent classes but also 7:Class with 17:Class and 4:Attribute with 12:Attribute. While 2,13:Method and adjacent classes form a conflict reason, the other overlaps in S are additional. 7,17:Class forms an isolated bound-

ary node since 7:Method occurs as boundary node in rule *decapsulateAttribute* (see also Fig. 10) and is not connected to any other node in S. In contrast, 4:Attribute is not a boundary node. 4,12:Attribute may, however, occur in a conflict reason extension, i.e., in S. Gluing the LHSs of rules *decapsulateAttribute* and *pullUpEncapsulatedAttribute* at the graph S we get graph G2, the overlap graph of a critical pair.

We conclude this section by positioning conflict notions of coarse context granularity as they were used in the *conference paper* [17] with the notions of coarse context granularity as considered here. Based on the rationale to align our granularity considerations with the new findings on initial conflicts [15], these former notions are now deprecated. The relationships, shown in Fig. 15, illustrate in particular the contribution of this work w.r.t. the conference paper. In particular, the former notion of conflict reason [17] is aligned with the notion of essential critical pairs [19]. Since the notion of conflict reason introduced in this paper includes a stronger condition than the one given in [17], each conflict reason in the new sense is also a conflict reason according to [17] but not vice versa. As explained before, our adapted conflict reason notion corresponds more closely to initial conflicts, which are therefore preferable to essential critical pairs for representing medium context granularity [15]. On the medium context granularity level, we thus have that each initial conflict is an essential critical pair, but not vice versa. Compared to Def. 5, this means that each initial conflict satisfies also Item 4 (no isolated boundary node in overlap graph) in addition to the first three items that are satisfied by an essential critical pair. This explains why the overlap granularity of the latter is finer than for initial conflicts. On the other hand, each essential critical pair is a critical pair, but not vice versa [19]. Compared to Def. 5, this means that each essential critical pair satisfies Item 3 (overlap in deletion graphs only) in addition to the first two conditions that are satisfied by a critical pair. This explains why the overlap granularity of the latter is finer than the one for essential critical pairs.

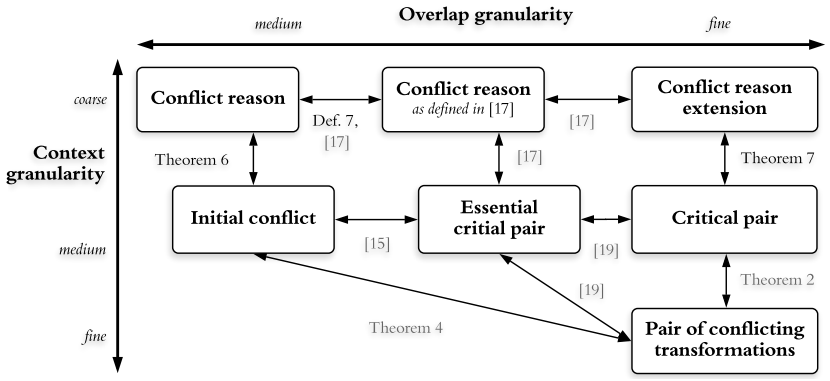


Figure 15: Positioning conflict notions as used in conference paper [17] in two dimensions of granularity

4.5. Interrelating Conflict Notions of Coarse Context Granularity

The subsequent results clarify the main interrelations between the new conflict notions for rules, i.e., conflict notions with coarse context, and varying overlap granularity (see coarse context granularity level in Fig. 1). Basically, we show that conflict notions with coarser overlap granularity can be extended to conflict notions with finer overlap granularity. Conversely, those with finer overlap granularity can be restricted to or are covered by conflict notions with finer overlap granularity. In particular, we will show that conflict reasons are covered by conflict atoms embedded into it. This implies that the overlap of conflict atoms is, in general, coarser than the one of a conflict reason. Therefore, we assign to *conflict atoms* the new remaining *coarse overlap granularity level*.

Theorem 8 (Extension of conflict part to reason)**.** *Given a conflict part* $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\hookrightarrow} L_2$ *for rule pair* (r_1, r_2) *with LHSs* L_1 *and* L_2*, there is a conflict reason for* (r_1, r_2) *such that the conflict part* s_1 *can be embedded into it.*

Proof. Due to Def. 7, a conflict part fulfills the transformation condition. Hence, there exists a pair of direct transformations $(t_1, t_2) = (G \overset{m_1,r_1}{\Longrightarrow} H_1, G \overset{m_2,r_2}{\Longrightarrow} H_2)$ such that $m_1(c_1(o_1(S_1))) = m_2(q_{12}(S_1))$. Then, we construct the initial conflict (with matches m_1^I for rule r_1 and m_2^I for rule r_2) for the given conflicting transformation pair (t_1, t_2) which exists due to Theorem 4. We construct the

pullback over $m_1^I \circ c_1$ and m_2^I yielding span $s_1^I : C_1 \overset{o_1^I}{\hookleftarrow} S_1^I \overset{q_{12}^I}{\rightarrow} L_2$. This span is a conflict reason for (r_1, r_2) due to Theorem 6. Due to the pullback property, there is a unique morphism $e : S_1 \rightarrow S_1^I$ such that $o_1^I \circ e = o_1$ and $q_{12}^I \circ e = q_{12}$. Hence, conflict part s_1 can be embedded into conflict reason s_1^I. □

The following lemma gives a more constructive characterization of conflict atom candidates compared to their introduction in Def. 7. 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 3 (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, 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 $o_1(v_1), o_1(v_2) \in B_1$.*

Proof. Since S_1 is included in the context graph C_1 of rule r_1 (1) each edge of graph S_1 must be deleted by r_1. From the conflict condition it follows that (2) each preserved node of graph S_1 is incident with a deleted edge of S_1 and that (3) S_1 contains at least one graph element that is deleted.

Assume that graph S_1 contains more than one graph element being deleted. We can find an embedding of a graph S'_1 with exactly one graph element being deleted. If an extra node is deleted, we can pick the node itself. If an extra edge is deleted with incident preserved nodes, we pick this edge with its incident nodes. If an extra edge e is deleted with one of its incident nodes n or m being deleted as well, we pick one of the nodes n or m being deleted. These observations contradict with the minimality condition for s_1 such that S_1 cannot contain more than one graph element being deleted. Therefore, the graph elements of S_1 are mapped to C_1 according to cases (1) or (2) above. □

The following theorem states that each conflict part (and therefore also each

conflict reason) is covered by a unique non-empty set of conflict atoms, i.e., all atoms that can be embedded into that conflict part. 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. Note that our notion of covering assumes that all edges incident with deleted nodes contained in some conflict part are covered implicitly, since conflict atoms as shown in Lemma 3 only consist of deleted nodes or deleted edges incident with preserved nodes. Of course, this result also holds for the special case when the conflict reason is minimal.

Theorem 9 (Covering of conflict parts by atoms)**.** *Given a conflict part* $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\rightarrow} L_2$ *for rules* (r_1, r_2)*, then the set* A *of all conflict atoms that can be embedded into* s_1 *is non-empty and* covers s_1*, i.e., for each conflict part* $s'_1 : C_1 \overset{o'_1}{\hookleftarrow} S'_1 \overset{q'_{12}}{\rightarrow} L_2$ *for* (r_1, r_2) *that can be embedded into* s_1*, it holds that* s'_1 *is isomorphic to* s_1 *if each atom in* A *can be embedded into* s'_1*.*

Proof. We first show that the set A is non-empty. Assume that the conflict part s_1 is not already a conflict atom. Then we can construct at least one conflict atom that can be embedded into s_1 as follows: A conflict part fulfills the weak conflict condition; hence there is at least one graph element x in S_1 that is deleted. Then we have three cases: (1) x is a node, then we consider the graph consisting of this node. (2) x is an edge not incident to a deleted node, then we consider the graph consisting of this edge with incident nodes. (3) x is an edge incident to a deleted node, then we consider the graph consisting of one of such an incident deleted node. In all three cases it is obvious that a conflict atom candidate arises that can be embedded into s_1. Since, for the conflict part s_1, two transformations exist which overlap in at least this conflict part, and since each of these conflict atom candidates is embedded into s_1, the transformation condition is simply inherited. Therefore we have found, in particular, a conflict atom.

Now we assume that s'_1 is not isomorphic to s_1: This is possible only if at least one graph element of S_1 is missing in S'_1. This element cannot be a

deleted node or a deleted edge with incident nodes being preserved by the first rule since there are atoms for these cases. The missing element cannot be an isolated boundary node, since this would contradict with a conflict part fulfilling the conflict condition. Hence, the missing element must be an edge e incident with a deleted node. Since e occurs in S_1, there has to be a corresponding edge e_2 in L_2. Since s'_1 is a conflict part for (r_1, r_2) as well, there exists according to Theorem 5 a conflicting transformation pair (t'_1, t'_2). The matches of such a pair (t'_1, t'_2) do not identify e with e_2 (because of the completeness condition), but then, since t'_1 deletes e but not e_2 although they have a common incident deleted node, t'_1 cannot be a transformation since e_2 would dangle. Hence, s'_1 cannot be a conflict part which contradicts our assumption. □

From this Theorem the following Corollary can be derived.[8]

Corollary 1 (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 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* s'_1 *is isomorphic to* s_1 *if each atom in A can be embedded into* s'_1*.*

Proof. This follows directly from Theorem 1. □

Example 6 (Covering of conflict reason by atoms)**.** On the left of Fig. 16 an example of a conflict reason is shown as it occurs for the pair of refactoring rules *decapsulateAttribute* and *pullUpEncapsulatedAttribute*. Conflict atoms 2,13:Method, 3,14:Method, and 5,15:Parameter (which are shown in Fig. 3) can be embedded into it. These three atoms cover the conflict reason on the left of Fig. 16 in the sense that there is no smaller reason which can be embedded

[8] In comparison to the conference paper [17], this result has become much more elegant since our notion of conflict reason relies on a stronger conflict condition now aligned with the notion of initial conflicts. In particular, we can disregard isolated boundary atoms in the coverings of conflict reasons now.

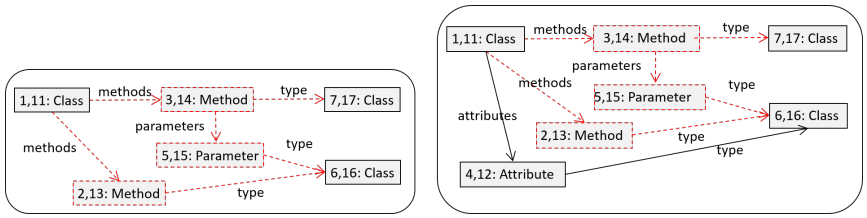


Figure 16: Conflict reason (left) and conflict reason extension (right) for the pair of refactoring rules *decapsulateAttribute* and *pullUpEncapsulatedAttribute*

into this one and where all three atoms can be embedded in. Actually, there are three further conflict reasons as pointed out in Sect. 2. All these three reasons are smaller than this conflict reason; they are all covered by, however, less than three atoms.

Conflict reason extensions contain all graph elements that overlap in a conflicting transformation pair, even elements that are not deleted but used by both participating rules. Hence, a conflict reason extension might show too much overlap information. By definition, for each conflict reason extension, there is a conflict reason which can be embedded into this extension. Hence, a conflict reason extension can always be restricted to a conflict reason.Vice versa, 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 10 (Conflict reason and conflict reason extension)**. Restriction.** *Given a conflict reason extension* $s : L_1 \overset{a_1}{\hookleftarrow} S \overset{b_2}{\to} L_2$ *for rules* (r_1, r_2)*, there is a conflict reason* $s_1 : C_1 \overset{o_1}{\hookleftarrow} S'_1 \overset{q_{12}}{\to} L_2$ *for rules* (r_1, r_2) *which can be embedded into* s*.*

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

Proof. **Restriction:** Follows directly from Def. 7.

Extension: For the conflict reason s_1 there exists a pair of direct transformations $(t_1, t_2) = (G \overset{m_1,r_1}{\Longrightarrow} H_1, G \overset{m_2,r_2}{\Longrightarrow} H_2)$ with s_1 being the pullback of

$(m_1 \circ c_1, m_2)$. Now we can also build the pullback (PB) of (m_1, m_2) such that we get a conflict reason extension s. In particular, the conflict reason s_1 can be embedded into s because of the pullback property of (PB) and the fact that $m_1 \circ c_1 \circ o_1 = m_2 \circ q_{12}$. □

Example 7 (Conflict reason extension)**.** The conflict reason on the left of Fig. 16 can be extended to a conflict reason extension if the conflicting transformation pair is embedded into overlaps in exactly the contained graph elements. On the right of Fig. 16 a slightly larger conflict reason extension is shown which overlaps the attribute nodes and their adjacent edges in addition. The corresponding conflicting transformation pair would also overlap in all graph elements indicated by the conflict reason extension. The conflict reason on the left, however, would also be the corresponding conflict reason of the right conflict reason extension. So, this example shows that a conflict reason can be embedded into a conflict reason extension; there may be more than one extension in general.

4.6. Relating Conflict Notions of Binary and Coarse Context Granularity

Finally, we consider the interrelation of conflict notions having binary and coarse context granularity (see Fig. 1). We show that for each conflicting rule pair (binary granularity) there is a conflict part (coarse context granularity) and vice versa. Finally, we show that for each pair of conflicting rules (binary granularity) there is in particular a conflict atom and vice versa. Note that by transitively combining all other relationships depicted in Fig. 1, the binary level can be related analogously to all other conflict notions.

Corollary 2 (Conflicting rule pair and conflict part)**.** *A rule pair (r_1, r_2) is conflicting if and only if there exists a conflict part for (r_1, r_2).*

Proof. Given a conflicting rule pair (r_1, r_2) as in Def. 2, there is a conflicting transformation pair $(t_1, t_2) = (G \overset{m_1,r_1}{\Longrightarrow} H_1, G \overset{m_2,r_2}{\Longrightarrow} H_2)$. Due to Theorem 5 we know that there exists a conflict part for (r_1, r_2).

Given a conflict part $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\rightarrow} L_2$ for rule pair (r_1, r_2), there is a conflicting transformation pair $(t_1, t_2) = (G \overset{m_1,r_1}{\Longrightarrow} H_1, G \overset{m_2,r_2}{\Longrightarrow} H_2)$ due to

Theorem 5 such that the rule pair (r_1, r_2) is indeed conflicting. □

Corollary 3 (Conflicting rule pair and conflict atom)**.** *A rule pair (r_1, r_2) is conflicting if and only if there exists a conflict atom for (r_1, r_2).*

Proof. From Corollary 2 it follows that there is a conflict part for (r_1, r_2). From Theorem 9 it follows that each conflict part is covered by at least one conflict atom for (r_1, r_2).

Given a conflict atom $s_1 : C_1 \overset{o_1}{\hookleftarrow} S_1 \overset{q_{12}}{\rightarrow} L_2$ for rule pair (r_1, r_2), then, because of Corollary 2, we know that this rule pair is conflicting. □

4.7. Dual Notions for Dependencies

To reason about dependencies, we consider the dual concepts and results that we get by inverting the left transformation of a conflicting transformation pair. This means that we check if $G \overset{p_1^{-1}, m'_1}{\Longleftarrow} H_1 \overset{p_2, m_2}{\Longrightarrow} H_2$ is conflicting, which is equivalent to the sequence $G \overset{p_1, m_1}{\Longrightarrow} H_1 \overset{p_2, m_2}{\Longrightarrow} H_2$ being dependent. This is possible since a transformation is symmetrically defined by two pushouts. They ensure in particular that morphisms $m : L \rightarrow G$ and $m' : R \rightarrow H$ both fulfill the gluing condition.

Dependency parts, atoms, reasons, and reason extensions can be defined analogously to Def. 7. 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. We illustrate this with an example.

Example 8. Figure 17 specifies the encapsulation of an attribute in a slightly simplified way. When checking for *dependencies* between this rule and the rule *pullUpEncapsulatedAttribute* (Fig. 2, right), one can equivalently check for *conflicts* between its inverse rule, *decapsulateAttribute* (Fig. 2, left), and *pullUpEncapsulatedAttribute*. Therefore, the *conflict atoms* and *minimal conflict reasons* of the rule pair (*decapsulateAttribute*, *pullUpEncapsulatedAttribute*) (Fig. 3) are exactly the *dependency atoms* and *minimal dependency reasons* for the rule pair (*encapsulateAttribute*, *pullUpEncapsulatedAttribute*). As a matter of course,

e.g., these dependency atoms cover the dependency reasons for the rule pair (*encapsulateAttribute*, *pullUpEncapsulatedAttribute*) in the same sense in which the *conflict atoms* cover the *conflict reasons* of the rule pair (*decapsulateAttribute*, *pullUpEncapsulatedAttribute*) (compare Corollary 1).

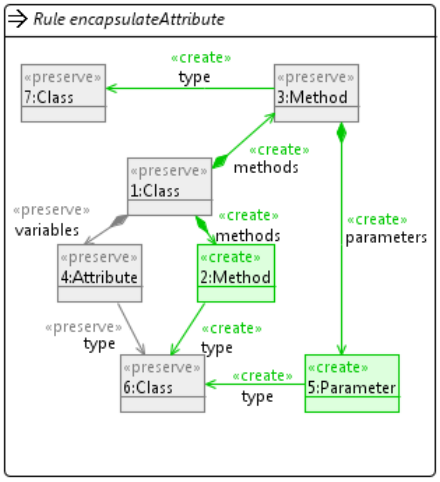


Figure 17: Simplified rule specifying the refactoring *Encapsulate Attribute*

5. Related Work and Conclusion

In this paper, we lay the basis for a refined analysis of conflicts and dependencies by presenting conflict and dependency notions along two different granularity dimensions: overlap and context granularity. Furthermore, we investigate their interrelations.

With this contribution, we aim to improve on critical pair analysis (CPA), the standard technique for detecting conflicts and dependencies in graph transformation systems [13] at design time. Originally being developed for term and term graph rewriting [24], CPA extends the theory of graph transformation and, more generally, of $\mathcal{M}$-adhesive transformation systems [21, 14]. The CPA is available for plain rules as well as rules with application conditions [25]. Moreover, several works [13, 26, 27] exist to design CPA for attributed rules.

Tool support for CPA is offered by the graph transformation tools AGG [28] and Verigraph [29] and by the graph-based model transformation tool Henshin [30]. All of these tools provide the user with a set of (essential) critical pairs for each pair of rules.

The computation of conflicts and dependencies using the concepts introduced in the present work has been prototypically implemented in Henshin [16]. The evaluation of a part of this prototype – where we have chosen to implement the computation of binary results, minimal conflict reasons and conflict reasons – shows improved performance [16] compared to the state-of-the-art CPA implementations. The improved usability of coarser conflict notions w.r.t. overlap granularity in the form of minimal conflict reasons has been confirmed in a user study [16]; investigating the effect of context granularity is left as future work. The formal study of relationships in the present paper lays the basis for extending this solution by implementing the computation of conflict notions on any desired level of granularity, and studying the associated effect on usability and performance.

Azzi et al. conducted similar research aiming to identify root causes of conflicting rule applications in [22] appearing after the publication of [17]. Their work is based upon an alternative characterization of parallel independence [31]. The most important difference is that we define our notions for pairs of rules instead of pairs of transformation as done by Azzi et al. Moreover, they do not consider the different levels of granularity proposed in this work but only what corresponds to conflict reasons and conflict reason extensions (in our terminology). And technically, we consider spans while they are interested in subobjects of the intersection of the rules' LHSs. By computing their so-called *disabling essences* in a slightly different order (first taking pullbacks and afterwards an initial pushout), these do not contain isolated boundary nodes. Moreover, Azzi et al. proved a disabling essence to be a subobject of a (suitably adapted) conflict reason (basically, the conflict graph S_1) as it was defined in [17]. Using the newly introduced conflict condition of this paper it is not difficult to prove the converse in the setting of categories of set-valued functors. This means,

given a pair of rules, a conflict reason is isomorphic to the disabling essence of the corresponding pair of transformations (existing by the transformation condition).

Currently, we restrict our formal considerations to graphs and graph transformations. Since all main conflict notions are based on concepts from category theory, our work is prepared to adapt to more sophisticated forms of graphs or graph transformation. It is up to future work to – if possible – come up with categorical means to define also the granularity dimensions more formally than introduced in this paper. Furthermore, it is interesting to adapt the new notions to transformation rules with negative [20] or more complex nested application conditions [25]. Analogously, to handle attributes within conflicts appropriately it is promising to adapt our approach to lazy graph transformations [32] and to come up with a light-weight conflict analysis complementing the work of Deckwerth et al. [33] where conflicts are detected between edit operations on feature models. They combine the CPA with a 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 [34] is a promising idea to save redundant computation effort.

Acknowledgements. We wish to thank the anonymous reviewers for their thoughtful comments, which helped us a lot to improve the final version of this paper.

This work was partially funded by the German Research Foundation, Priority Program SPP 1593 "Design for Future – Managed Software Evolution" and the project "Tripel Graph Grammars (TGG) 2.0: Reliable and Scalable Model Integration" as well as by the research project Visual Privacy Management in User Centric Open Environments (supported by the EU's Horizon 2020 programme, Proposal number: 653642).

[1] L. Baresi, R. Heckel, Tutorial introduction to graph transformation: A

software engineering perspective, in: International Conference on Graph Transformation, Springer, 2002, pp. 402–429.

[2] S. Finger, J. R. Dixon, A review of research in mechanical engineering design. part ii: Representations, analysis, and design for the life cycle, Research in Engineering Design 1 (2) (1989) 121–137.

[3] F. Rosselló, G. Valiente, Graph transformation in molecular biology, Formal Methods in Software and Systems Modeling (2005) 116–133.

[4] J. H. Hausmann, R. Heckel, 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.

[5] T. Mens, R. Van Der Straeten, M. D'Hondt, Detecting and resolving model inconsistencies using transformation dependency analysis, in: Int. Conf. on Model Driven Engineering Languages and Systems (MoDELS), Springer, 2006, pp. 200–214.

[6] T. Mens, G. Taentzer, O. Runge, Analysing refactoring dependencies using graph transformation, Software and System Modeling 6 (3) (2007) 269–285.

[7] P. Jayaraman, J. Whittle, A. M. Elkhodary, 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 (MoDELS), Springer, 2007, pp. 151–165.

[8] J. M. Küster, C. Gerth, G. Engels, Dependent and conflicting change operations of process models, in: European Conf. on Model Driven Architecture - Foundations and Applications (ECMDA-FA), Vol. 5562, Springer, 2009, pp. 158–173.

[9] K. Mehner-Heindl, M. Monga, G. Taentzer, Analysis of Aspect-Oriented Models Using Graph Transformation Systems, in: Aspect-Oriented Requirements Engineering, Springer, 2013, pp. 243–270.

[10] L. Baresi, R. Heckel, S. Thöne, D. Varró, Modeling and validation of service-oriented architectures: application vs. style, in: Symp. on Foundations of Software Engineering/European Software Engineering Conf. (FSE/ESEC), ACM, 2003, pp. 68–77. doi:10.1145/940071.940082.

[11] C. Ermel, J. Gall, L. Lambers, 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 (FASE), Springer, 2011, pp. 156–170.

[12] D. Strüber, G. Taentzer, S. Jurack, T. Schäfer, Towards a distributed modeling process based on composite models, in: Int. Conf. on Fundamental Approaches to Software Engineering (FASE), Springer, 2013, pp. 6–20.

[13] R. Heckel, J. M. Küster, G. Taentzer, Confluence of Typed Attributed Graph Transformation Systems, in: First Int. Conf. on Graph Transformation (ICGT), Springer, 2002, pp. 161–176.

[14] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer, Fundamentals of Algebraic Graph Transformation, Monographs in Theoretical Computer Science, Springer, 2006.

[15] L. Lambers, K. Born, F. Orejas, D. Strüber, G. Taentzer, Initial Conflicts and Dependencies: Critical Pairs Revisited, in: R. Heckel, G. Taentzer (Eds.), Graph Transformation, Specifications, and Nets, Vol. 10800 of Lecture Notes in Computer Science, Springer, 2018, pp. 105–123.

[16] L. Lambers, D. Strüber, G. Taentzer, K. Born, J. Hübert, Multi-granular conflict and dependency analysis in software engineering based on graph transformation, in: Proceedings of the 40th International Conference on Software Engineering, ACM, New York, NY, USA, 2018, pp. 716–727.

[17] K. Born, L. Lambers, D. Strüber, G. Taentzer, Granularity of conflicts and dependencies in graph transformation systems, in: International Conference on Graph Transformation (ICGT), 2017, pp. 125–141.

[18] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley, Boston, MA, USA, 1999.

[19] L. Lambers, H. Ehrig, F. Orejas, Efficient conflict detection in graph transformation systems by essential critical pairs, Electr. Notes Theor. Comput. Sci. 211 (2008) 17–26.

[20] L. Lambers, Certifying rule-based models using graph transformation, Ph.D. thesis, Berlin Institute of Technology (2010).

[21] H. Ehrig, J. Padberg, U. Prange, A. Habel, Adhesive high-level replacement systems: A new categorical framework for graph transformation, Fundam. Inform. 74 (1) (2006) 1–29.

[22] G. G. Azzi, A. Corradini, L. Ribeiro, On the essence and initiality of conflicts, in: L. Lambers, J. Weber (Eds.), Graph Transformation - 11th International Conference, ICGT 2018, Held as Part of STAF 2018, Toulouse, France, June 25-26, 2018, Proceedings, Vol. 10887 of Lecture Notes in Computer Science, Springer, 2018, pp. 99–117. doi:10.1007/978-3-319-92991-0_7.
URL https://doi.org/10.1007/978-3-319-92991-0_7

[23] K. Gabriel, B. Braatz, H. Ehrig, U. Golas, Finitary -adhesive categories, Mathematical Structures in Computer Science 24 (4). doi:10.1017/S0960129512000321.
URL https://doi.org/10.1017/S0960129512000321

[24] D. Plump, Critical Pairs in Term Graph Rewriting, in: Mathematical Foundations of Computer Science, 1994, pp. 556–566.

[25] H. Ehrig, U. Golas, A. Habel, L. Lambers, F. Orejas, $\mathcal{M}$-adhesive transformation systems with nested application conditions. part 2: Embedding, critical pairs and local confluence, Fundam. Inform. 118 (1-2) (2012) 35–63.

[26] G. Kulcsár, F. Deckwerth, M. Lochau, G. Varró, A. Schürr, Improved conflict detection for graph transformation with attributes, in: A. Rensink,

E. Zambon (Eds.), Proceedings Graphs as Models, GaM@ETAPS 2015, London, UK, 11-12 April 2015., Vol. 181 of EPTCS, 2015, pp. 97–112. doi:10.4204/EPTCS.181.7.
URL https://doi.org/10.4204/EPTCS.181.7

[27] I. Hristakiev, D. Plump, Towards critical pair analysis for the graph programming language gp 2, in: P. James, M. Roggenbach (Eds.), Recent Trends in Algebraic Development Techniques - 23rd IFIP WG 1.3 International Workshop, WADT 2016, Gregynog, UK, September 21-24, 2016, Revised Selected Papers, Vol. 10644 of Lecture Notes in Computer Science, Springer, 2016, pp. 153–169. doi:10.1007/978-3-319-72044-9.
URL https://doi.org/10.1007/978-3-319-72044-9

[28] G. Taentzer, AGG: A graph transformation environment for modeling and validation of software, in: Int. Workshop on Applications of Graph Transformations with Industrial Relevance (AGTIVE), Springer, 2003, pp. 446–453.

[29] Verigraph, Verigraph, https://github.com/Verites/verigraph.

[30] T. Arendt, E. Biermann, S. Jurack, C. Krause, G. Taentzer, Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations, in: Int. Conf. on Model-Driven Engineering Languages and Systems (MoDELS), 2010, pp. 121–135.

[31] A. Corradini, D. Duval, M. Löwe, L. Ribeiro, R. Machado, A. Costa, G. G. Azzi, J. S. Bezerra, L. M. Rodrigues, On the essence of parallel independence for the double-pushout and sesqui-pushout approaches, in: R. Heckel, G. Taentzer (Eds.), Graph Transformation, Specifications, and Nets: In Memory of Hartmut Ehrig, Springer International Publishing, Cham, 2018, pp. 1–18. doi:10.1007/978-3-319-75396-6_1.
URL https://doi.org/10.1007/978-3-319-75396-6_1

[32] F. Orejas, L. Lambers, Lazy graph transformation, Fundam. Inform. 118 (1-2) (2012) 65–96.

[33] F. Deckwerth, G. Kulcsár, M. Lochau, G. Varró, 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, Vol. 206 of EPTCS, 2016, pp. 17–31.

[34] D. Strüber, J. Rubin, T. Arendt, M. Chechik, G. Taentzer, 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.