Improved Conflict Detection for Graph Transformation with Attributes

Géza Kulcsár
Technische Universität Darmstadt
Real-Time Systems Lab
Merckstr. 25
64283 Darmstadt, Germany

Frederik Deckwerth
Technische Universität Darmstadt
Real-Time Systems Lab
Merckstr. 25
64283 Darmstadt, Germany

Malte Lochau
Technische Universität Darmstadt
Real-Time Systems Lab
Merckstr. 25
64283 Darmstadt, Germany

Gergely Varró
Technische Universität Darmstadt
Real-Time Systems Lab
Merckstr. 25
64283 Darmstadt, Germany

Andy Schürr†
Technische Universität Darmstadt
Real-Time Systems Lab
Merckstr. 25
64283 Darmstadt, Germany

{geza.kulcsar|frederik.deckwerth|malte.lochau|gergely.varro|andy.schuerr}@es.tu-darmstadt.de

In graph transformation, a conflict describes a situation where two alternative transformations cannot be arbitrarily serialized. When enriching graphs with attributes, existing conflict detection techniques typically report a conflict whenever at least one of two transformations manipulates a shared attribute. In this paper, we propose an improved, less conservative condition for static conflict detection of graph transformation with attributes by explicitly taking the semantics of the attribute operations into account. The proposed technique is based on symbolic graphs, which extend the traditional notion of graphs by logic formulas used for attribute handling. The approach is proven complete, i.e., any potential conflict is guaranteed to be detected.

1 Introduction

According to the Model-Driven Engineering (MDE) principle, systems under design are represented by graph-based models. The change and evolution of such models is frequently described by the declarative, rule-based approach of graph transformation [4, 13]. However, models arising in real-world application scenarios typically contain numerical as well as textual attributes in addition to the graph-based structure. For this purpose, an extension to graph transformation is required, being capable of representing and manipulating attributes of nodes and edges.

A major challenge in graph transformation is to statically analyse possible conflicts between rule applications. The goal of conflict detection is to check if two graph transformation rules, both potentially applicable concurrently on the same input graph, are in any case arbitrarily serializable, i.e., if the two possible execution sequences result in the same (or at least two isomorphic) output graph(s).

Critical Pair Analysis (CPA) is a common static analysis technique for conflict detection, defining a process of pairwise testing a set of graph transformation rules for possible conflicts [4]. Unfortunately, a naïve adoption of CPA to graph transformation with attributes is too strict: whenever an attribute is modified by a rule application, and another rule application is also accessing the same attribute, they are immediately considered to be in conflict [7].

*Supported by CASED (www.cased.de).
†This work has been co-funded by the DFG within the Collaborative Research Center (CRC) 1053 – MAKI.
In this paper, we propose an improved, less conservative condition for static conflict detection of graph transformation with attributes by explicitly taking the semantics of the attribute operations into account. In particular, we make the following contributions:

- We define direct confluence as an appropriate conflict condition for graph transformation with attributes based on symbolic graphs, which reduces the number of false positives compared to existing conflict detection approaches. Using symbolic graphs further allows for an effective implementation of the proposed approach using a combination of graph transformation tools and off-the-shelf SMT solvers.

- We prove that our approach is still complete [4], i.e., any potential conflict is guaranteed to be detected.

The paper is organized as follows: the basic concepts and definitions are introduced in Section 2. Section 3 proposes direct confluence as an improved conflict condition for rules with attributes and, based on that, conflicting pairs are defined. In Section 4, the procedure for identifying conflicts is presented and proven complete. Section 5 surveys related work and Section 6 concludes the paper.

2 Preliminaries

In this section, we recapitulate the notions of symbolic graphs and symbolic graph transformation [11] that are used as a framework for our approach. Before getting into details of symbolic attributed graphs, we first define graphs and graph transformation without attributes.

**Definition 1 (Graphs and Graph Morphisms).** A graph \( G = (V_G, E_G, s_G, t_G) \) is a tuple consisting of a set of graph nodes \( V_G \), a set of graph edges \( E_G \), and the source and target functions \( s_G, t_G : E_G \rightarrow V_G \), respectively. A graph morphism \( f = (f_V, f_E) : G \rightarrow H \), for mapping a graph \( G \) to a graph \( H \), consists of two functions \( f_V : V_G \rightarrow V_H \) and \( f_E : E_G \rightarrow E_H \) preserving the source and target functions: \( f_V \circ s_G = s_H \circ f_E \) and \( f_V \circ t_G = t_H \circ f_E \). A graph morphism is a monomorphism if \( f_V \) and \( f_E \) are injective functions. A graph morphism is an isomorphism if \( f_V \) and \( f_E \) are bijective functions.

Based on this definition of graphs, graph transformation relies on the notion of pushouts. A pushout has the following meaning (in the category of graphs): given three graphs \( A, B, C \) and two morphisms \( f : A \rightarrow B, g : A \rightarrow C \), their pushout consists of the pushout object \( P \) and two morphisms \( g' : B \rightarrow P, f' : C \rightarrow P \), where \( P \) is the gluing of \( B \) and \( C \) along the elements of \( A \), the latter being, in a way, present in both as \( f(A) \) and \( g(A) \), respectively. Correspondingly, pullbacks are the counterpart of pushouts. Given three graphs \( B, C, P \) and two morphisms \( g' : B \rightarrow P, f' : C \rightarrow P \), their pullback consists of the pullback object \( A \) and the morphisms \( f : A \rightarrow B, g : A \rightarrow C \), where \( A \) can be seen as the intersection of \( B \) and \( C \), i.e., the elements of \( B \) and \( C \) which are overlapping in \( P \).

In the following, we use the double pushout (DPO) approach to define graph transformation [4, 13].

**Definition 2 (Graph Transformation Rule).** A graph transformation rule \( r \) in the DPO approach consists of a left-hand side (LHS) graph \( L \), an interface graph \( K \), and a right-hand side (RHS) graph \( R \) and the morphisms \( l : K \rightarrow L \) and \( r : K \rightarrow R \).

An application of rule \( r \) to a graph \( G \) is defined by the two pushouts (1) and (2) in the diagram below:

\[
\begin{array}{ccc}
L & \xrightarrow{l} & K & \xrightarrow{r} & R \\
\downarrow m & & \downarrow & & \downarrow m' \\
G & \xleftarrow{m} & D & \xrightarrow{m'} & H \\
\end{array}
\]
A rule is applied by first identifying a **match** $m : L \rightarrow G$ of the left-hand side $L$ in graph $G$. In the next step, the **context graph** $D$ is obtained by removing all elements in $G$ which are identified by match $m$, but are not contained in the interface $K$. The result of the rule application, $H$, is obtained by adding all elements of the right-hand side $R$ to the context $D$ which do not have a pre-image in the interface $K$.

A **direct derivation** of rule $r$ at match $m$, denoted as $G \xrightarrow{r,m} H$, is the single step from graph $G$ to graph $H$ derived by applying rule $r$ to graph $G$ at the match $m$.

Until now, we have limited our discussion to plain graphs, i.e., graphs incapable of expressing attributes such as integer variables with corresponding operations. As a first step towards graphs with attributes, we extend their definition to E-graphs [4]. An E-graph is a graph extended by special kinds of label nodes $(V^D)$ and edges $(E^V_L$ and $E^E_L$ for node and edge attribute, respectively) used for carrying the attribute values.

**Definition 3** (E-graphs and E-graph Morphisms [4]). An **E-graph** $EG = (G,D)$ is a tuple consisting of a graph $G$ and a **labeling part** $D = (V^D_G, E^V_L_G, E^E_L_G, \Sigma_G, \Gamma_G, \delta_G, \iota_G)$ with a set of **label nodes** $V^D_G$, two sets of edges $E^V_L_G$ and $E^E_L_G$ for node and edge labeling, respectively, and the source and target functions $\delta_G : E^V_L_G \rightarrow V^D_G, \delta_G : E^V_L_G \rightarrow V^D_G, \delta_G : E^E_L_G \rightarrow E_G$ and $\iota_G : E^E_L_G \rightarrow V^D_G$ assigning the label nodes to the graph nodes an edges, respectively.

An **E-graph morphism** $h = (h_G, h_D, h_{V_L}, h_{E_L})$ consists of a graph morphism $h_G$ and three functions $h_D, h_{V_L}, h_{E_L}$ mapping the label nodes and the labeling edges while preserving source and target functions. An E-graph morphism is a **monomorphism** (isomorphism) if its functions are injective (bijective).

In the following, we omit the $E$-prefix and denote E-graphs using e.g. $G$ instead of $EG$.

The construction of E-graphs contains labels as placeholders for attribute values. In order to be able to define and manipulate those attribute values, we employ a data algebra. A **data algebra** $\mathcal{D}$ is a signature $\Sigma$ consisting of symbols for sorts, functions and predicates; and a mapping of these symbols to sets and functions, assigning meaning to the symbols. For the examples, we use the algebra of natural numbers with addition and equality. This algebra consists of the sort symbol $\mathbb{N}$ representing the (infinite) set of natural numbers, the binary function symbol '+' mapped to addition with the usual meaning, and the binary predicate symbol '=' defined by the equality relation on $\mathbb{N}$. For further details we refer to [5].

The concept of symbolic graphs has been introduced recently to combine the concept of E-graphs for representing attributes and data algebra for the values of those attributes. This way, symbolic graphs provide a convenient representation of graphs with attributes [11]. In particular, a symbolic graph is an E-graph whose label nodes contain variables and the values of these variables are constrained by a first-order logic formula, also being part of the symbolic graph.

Given a $\Sigma$-algebra $\mathcal{D}$ and a set of variables $\mathcal{X}$, a **first-order logic formula** is built from the variables in $\mathcal{X}$, the function and predicate symbols in $\Sigma$, the logic operators $\lor, \land, \neg, \Rightarrow, \Leftrightarrow$, the constants true and false and the quantifiers $\forall$ and $\exists$ in the usual way [14]. A **variable assignment** $\sigma : \mathcal{X} \rightarrow \mathcal{D}$ maps the variables $x \in \mathcal{X}$ to a value in $\mathcal{D}$. A first-order logic formula $\Phi$ is **evaluated** for a given assignment $\sigma$ by first replacing all variables in $\Phi$ according to the assignment $\sigma$ and evaluating the functions and predicates according to the algebra, and the logic operators. We write $\mathcal{D}, \sigma \models \Phi$ if and only if $\Phi$ evaluates to true for the assignment $\sigma$; and $\mathcal{D} \models \Phi$, if and only if $\Phi$ evaluates to true for all assignments.

**Definition 4** (Symbolic Graphs and Symbolic Graph Morphisms [11]). A **symbolic graph** $SG = (G, \Phi_G)$ consists of an E-graph $G$ and a first-order logic formula $\Phi_G$ over a given data algebra $\mathcal{D}$, using the label nodes of $G$ as variables and elements of $\mathcal{D}$ as constants.

A **symbolic graph morphism** $h : (G, \Phi_G) \rightarrow (H, \Phi_H)$ is an E-graph morphism $h : G \rightarrow H$ such that $\mathcal{D} \models \Phi_H \Rightarrow h_\Phi(\Phi_G)$, where $h_\Phi(\Phi_G)$ is the first-order logic formula obtained when replacing each variable $x$ in formula $\Phi_G$ as defined by the mapping for the label nodes $h_D(x)$. The symbolic graphs $SG_1 = (G_1, \Phi_1)$...
and $SG_2 = (G_2, \Phi_2)$ are isomorphic if there is a symbolic graph morphism $h : SG_1 \to SG_2$ that is an E-graph isomorphism and $D \models h_\Phi(\Phi_1) \Leftrightarrow \Phi_2$.

As the variables and, thus, the attribute values are determined by a first-order logic formula, a symbolic graph can be seen as a class of grounded symbolic graphs (GSG). A grounded symbolic graph is a symbolic graph where (i) each attribute value is constant, and (ii) for each value of the data algebra, it contains a corresponding constant label node. A grounded symbolic graph is created by adding to the set of label nodes a variable $c_v$ for each value $v$ in $D$, and extending the formula with the equation $c_v = v$, which assigns a constant value to each constant variable.

**Definition 5** (Grounded Symbolic Graph [11]). A symbolic graph $SG = (G, \Phi_G)$ with data algebra $D$ is grounded, denoted as $SG$, if it includes a variable $c_v \in V_G^D$ for each value $v \in D$, and for each variable assignment $\sigma : V_G^D \to D$ such that $D, \sigma \models \Phi_G$, it holds that $\sigma(c_v) = v$.

A grounded symbolic graph $SH$ is an instance of a symbolic graph $SG$ via $h : SG \to SH$ if $h$ is a symbolic graph morphism, which is injective for all kinds of nodes and edges except the label nodes.

**Example 1** (Symbolic and Grounded Symbolic Graphs). Figure 1a shows a grounded symbolic graph $SG_1 = (G_1, \Phi_1)$ consisting of a single graph node $n_1$ bearing an attribute carrying the variable $c_{42}$, and a formula $\Phi_1$, constraining each variable $c_v$ to value $v \in D$. The grounded symbolic graph $SG_1$ contains an infinite number of label nodes and corresponding equations as indicated by the ‘...’ in Figure 1a.

Figure 1b shows the (non-grounded) symbolic graph $SG_2 = (G_2, \Phi_2)$ whose E-graph part is identical to $G_1$. Consequently, there exists an E-graph morphism $h : G_2 \to G_1$ mapping nodes $n_2$ and $x$ of $G_2$ to nodes $n_1$ and $c_{42}$ of $G_1$, respectively. This morphism is a valid symbolic graph morphism as, according to the mapping of the label nodes ($h_\Phi(c_{42}) = x$), the condition $\Phi_1 \Rightarrow h_\Phi(\Phi_2)$ can be simplified to $(x = 42) \Rightarrow (x \leq 42)$ which evaluates to true. Hence, the grounded symbolic graph $SG_1$ is an instance of the symbolic graph $SG_2$.

Pushouts and pullbacks in symbolic graphs can be defined in terms of pushouts and pullbacks for graphs [11]. More specifically, the symbolic morphisms $f : (A, \Phi_A) \to (B, \Phi_B)$ and $g : (A, \Phi_A) \to (C, \Phi_C)$ are a symbolic pushout $f' : (B, \Phi_B) \to (D, \Phi_D)$ and $g' : (C, \Phi_C) \to (D, \Phi_D)$ with pushout object $(P, \Phi_P)$ if $f'$ and $g'$ are a pushout in E-graphs and $D \models (\Phi_P \Leftrightarrow f'_\Phi(\Phi_A) \land g'_\Phi(\Phi_C))$. A pullback is defined analogously where the formula $\Phi_A$ of the pullback object is given by the disjunction of $\Phi_B$ and $\Phi_C$.

A symbolic graph transformation rule is a graph transformation rule additionally equipped with a first-order logic formula.

**Definition 6** (Symbolic Graph Transformation Rule and Symbolic Direct Derivation [11]). A symbolic graph transformation rule $r$ is a pair $(L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R, \Phi)$, where $(L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ is an E-graph transformation rule and $\Phi$ is a single first-order logic formula shared by $L, K$ and $R$. The E-graph morphisms $l$ and $r$ are of a class $\mathcal{M}$ of morphisms injective for graph nodes and all kinds of edges and bijective for label nodes.
A symbolic direct derivation $SG \xrightarrow{r,m} SH$ is the application of a symbolic rule $r = (L \leftarrow K \rightarrow R, \Phi)$ on the symbolic graph $SG = (G, \Phi_G)$ at match $m : L \rightarrow SG$, resulting in the symbolic graph $SH = (H, \Phi_H)$, where $m$ is a symbolic graph morphism, which is injective for all kinds of nodes and edges except for the label nodes and $SH$ is produced as a DPO diagram in E-graphs.

**Fact 1** (Properties of Symbolic Direct Derivations [12]). The restrictions on morphisms $l$ and $r$ ensure that for any symbolic direct derivation $SG \xrightarrow{r,m} SH$,

(i) the set of label nodes and the formula remain unaltered, i.e., $V_G^D = V_H^D$ and $D \models \Phi_G \iff \Phi_H$, and

(ii) if $SG$ is grounded, then so is $SH$.

Note that (i) also implies coincidence on label nodes of the match $m : L \rightarrow SG$ and the comatch $cm : R \rightarrow SH$, i.e., $m_\Phi = cm_\Phi$.

Although, it seems counterintuitive at a first glance that we require $L, K$ and $R$ to share the same formula and set of label nodes, it does not mean that attribute values cannot be changed by a rule application, since attribute values are modified by redirecting label edges.

**Example 2** (Symbolic Graph Transformation Rule and Symbolic Direct Derivation). Figure 2 shows a symbolic graph transformation rule $r = (L \leftarrow K \rightarrow R, \Phi)$ (depicted in the upper part). The rule takes a graph node $n$ that has at least one attribute (denoted by the label edge between $n$ label node $x$) and increases it by one. This is achieved by introducing a new label node $x'$ to represent the attribute value after the rule application and constraining it to $x' = x + 1$ as defined by the formula $\Phi$. The attribute value is changed from the old value $x$ to the new value $x'$ by first deleting the label edge between $n$ and the old value $x$ and afterwards creating a new label edge assigning the new value $x'$ to $n$. The result from applying the rule to grounded symbolic graph $SG$ is shown on the bottom of Figure 2. The only valid mapping for match $m$ to satisfy $\Phi_G \Rightarrow m_\Phi(\Phi)$ is to map $x$ to $c_{42}$ and $x'$ to $c_{43}$. Then the resulting direct derivation $SG \xrightarrow{r,m} SH$ changes the attribute value from 42 (in grounded symbolic graph $SG$) to 43 in grounded symbolic graph $SH$ as expected.

In the following, we use symbolic graphs and symbolic graph transformation to present our approach.

### 3 A Conflict Notion for Graph Transformation with Attributes

In this section, we present an improved detection technique for potential rule conflicts for graph transformation with attributes. To this end, we define a notion of conflict on the level of direct derivations, and
we review parallel dependence as an existing sufficient condition for our notion of conflict. Thereupon, we show by means of an illustrative example that parallel dependence is too conservative especially in an attributed setting, i.e., rejecting too many conflict-free direct derivations. To overcome these deficiencies, we present a new condition, called direct confluence, that is sufficient for detecting conflicting direct derivations, but less restrictive than parallel dependence. Finally, to reason about conflicts on the rule level, we lift the direct confluence condition by defining conflicting pairs.

With the concept of conflicts, we grasp the situation where, given two rules \( r_1 \) and \( r_2 \) applicable on the same graph, we obtain different results depending on which rule is applied first. We characterize a conflict in terms of two alternative direct derivations that can not be arbitrarily serialized. In this case, applying the second transformation after the first leads to a different result than vice versa.

**Definition 7** (Conflict). Given a grounded symbolic graph \( SG \), the two alternative direct derivations \( SH_1 \xleftarrow{r_1} G \xrightarrow{r_2} SH_2 \) are a conflict if no direct derivations \( SH_1 \xrightarrow{r_1} SX_1 \) and \( SH_2 \xrightarrow{r_2} SX_2 \) exist with \( SX_1 \) and \( SX_2 \) being isomorphic.

Note that since \( SG \) is grounded, \( SH_1, SH_2, SX_1 \) and \( SX_2 \) are grounded, too.

This definition of conflicts leaves open how to practically determine that two given alternative direct derivations are a conflict. A corresponding condition to check if two direct derivations are a conflict is referred to as a conflict condition.

### 3.1 Parallel Dependence as a Conflict Condition

In the literature of graph transformation, a common conflict condition is the notion of parallel dependence \([13, 4]\). Intuitively, two direct derivations are parallel dependent if they are mutually exclusive, i.e., after one of the direct derivations, the other rule is not applicable anymore and/or vice versa. We adapt the notion of parallel dependence to symbolic graphs as follows.

**Definition 8** (Parallel Dependence). The symbolic direct derivations \( (H_1, \Phi) \xleftarrow{r_1} (G, \Phi) \xrightarrow{r_2} (H_2, \Phi) \) are parallel dependent iff the direct (E-graph) derivations \( H_1 \xleftarrow{r_1} G \xrightarrow{r_2} H_2 \) are parallel dependent, i.e., there does not exist E-graph morphism \( i : L_1 \to D_2 \) or \( j : L_2 \to D_1 \) such that \( m_1 = g_2 \circ i \) and \( m_2 = g_1 \circ j \), as in the diagram below.

![Diagram](image)

Two direct derivations not being parallel dependent are called parallel independent.

Note that the non-existence of morphism \( i \) means that the application of rule \( r_2 \) deletes at least one element which is required for the match of \( r_1 \) and vice versa for \( j \).

**Example 3** (Parallel Dependence). Figure 3 shows an example of two parallel dependent direct derivations. The two symbolic rules \( r_1 = (L_1 \xleftarrow{r_1} K_1 \xrightarrow{r_1} R_1, \Phi_1) \) and \( r_2 = (L_2 \xleftarrow{r_2} K_2 \xrightarrow{r_2} R_2, \Phi_2) \) are shown in the upper part of the figure. Both rules take a single graph node \( n \) with a single attribute (label node \( x \)); while rule \( r_1 \) increases the value of the attribute by 1, rule \( r_2 \) adds 2 to the attribute value. The bottom part of Figure 3 shows the application of the rules on the grounded symbolic graph \( SG \). As the morphisms
i : L₁ → D₂ and j : L₂ → D₁ do not exist because of a missing labeling edge, the depicted direct derivations are parallel dependent and, therefore, they are declared to be a conflict by parallel dependence.

However, if focusing on the intention of these rules, it seems rather intuitive that the direct derivations are not a conflict as the operations expressed by the rules are commutative, i.e., \( x + 2 = x + 1 + 1 \).

Concluding our example, although this technique is practical, efficient and only the two direct derivations are required for the decision process, it seems too strict (i.e., it produces too many false positives) for the desired attributed setting. The problem is that using the notion of parallel dependence, two rules are considered to have a (potential) conflict whenever an attribute is modified by one rule, that is accessed by the other rule (as also stated in [7]). The root of the problem resides in the construction of the underlying E-graphs, which do not reflect the intention of attribute operations, but rather delete and recreate the labeling edges whenever a new value is assigned to an attribute.

### 3.2 Direct Confluence as an Improved Conflict Condition

To overcome the deficiencies of parallel dependence as a conflict condition, we propose an alternative approach. Our proposal is based on the observation that the definition of conflicts (Def. 7) allows for directly checking if the different application sequences of the two rules result in isomorphic graphs. In particular, the proposed approach relies on our notion of direct confluence. To be more precise, two direct derivations which are not directly confluent are a conflict.

The definition of direct confluence has to fulfill that (i) given a pair of direct derivations for two rules \( r₁ \) and \( r₂ \) on the same input graph, there exists two derivation sequences (i.e. first \( r₁ \) and then \( r₂ \) and vice versa) whose resulting graphs are isomorphic and (ii) in both derivation sequences, the second direct derivations preserves at least the elements as the first direct derivations and send these to the same elements in the common result.

**Definition 9** (Direct Confluence). Given a pair of direct derivations \( SH₁ \xrightarrow{r₁,m₁} SG \xrightarrow{r₂,m₂} SH₂ \) with \( SG = (G, Φ_G) \), \( SH₁ = (H₁, Φ_{H₁}) \) and \( SH₂ = (H₂, Φ_{H₂}) \) being symbolic graphs, they are directly confluent if there exist direct derivations \( SH₁ \xrightarrow{r₂,m₂} SX₁ \) and \( SH₂ \xrightarrow{r₁,m₁} SX₂ \) such that

I. \( SX₁ = (X₁, Φ_{X₁}) \) and \( SX₂ = (X₂, Φ_{X₂}) \) are isomorphic, and

II. matches \( m₁ \) and \( m₂ \) are chosen in a way that (2), (3) and (4) commute, where (1) is the pullback of \((SD₁ → SG ← SD₂)\) and the graphs \( SD₁, SD₂, SQ₁ \) and \( SQ₂ \) are the context graphs of the corresponding direct derivations.
Property I ensures that the given direct derivations are not a conflict. Property II serves as a means of tracking for the matched elements after the direct derivations. This way, it is guaranteed that the second direct derivations are applied to the images of the same elements as the first ones. In other words, the symbolic graph $SZ$ contains all elements from the input graph that are preserved by both original direct derivations and the commuting rectangles of Property II guarantee that these elements are in the context graphs of the second direct derivations and (through the lower rectangle) that they are embedded in the resulting graph in the same way. In the following, when using the concept of direct confluence, we always assume that the matches are chosen appropriately according to Property II. Note that the definition of direct confluence is a specialization of strict confluence as defined in [4] (Def. 6.26), with the lower transformation chains consisting of exactly one direct derivation.

**Example 4** (Direct Confluence as an Improved Conflict Condition). Figure 4 shows (in the top right and top left corner) the results $SH_1$ and $SH_2$ of the alternative direct derivations $SH_1 \xRightarrow{r_1,m_1} SG \xRightarrow{r_2,m_2} SH_2$ presented in Example 3 (shown in Figure 3). On the bottom (from left to right), the symbolic rules $r_2 = (L_2 \xrightarrow{\Phi_2} K_2 \xrightarrow{r_2} R_2, \Phi_2)$ and $r_1 = (L_1 \xrightarrow{\Phi_1} K_1 \xrightarrow{r_1} R_1, \Phi_1)$ are shown. In order to check direct confluence, both rules are applied to $SH_1$ and $SH_2$, resulting in the direct derivations $SH_1 \xRightarrow{r_2,m_2} SX_1$ and $SH_2 \xRightarrow{r_1,m_1} SX_2$. As grounded symbolic graphs $SX_1$ and $SX_2$ are isomorphic, direct confluence declares, in contrast to parallel dependence, that the two alternative derivation $SH_1 \xleftarrow{\Phi_2} SG \xrightarrow{r_2,m_2} SH_2$ are not a conflict.

We have shown that direct confluence as a conflict condition is in accordance with our notion of conflicts and is, therefore, suitable for conflict detection in the presence of attributes. However, in most applications, one is rather interested in a conflict detection on the level of rules instead of their applications.
3.3 Lifting Conflicts to Rule Level

In the following, we show how we lift our notion of direct confluence from the direct derivation level to the rule level. As a starting point, we recall the well-known concept of critical pairs that is used to lift the parallel dependence condition to the rule level. First, we adapt critical pairs to our setting of symbolic graphs. Afterwards, we show that this criterion is too conservative, however, it is used as a first necessary condition in the decision process as if two rules are parallel independent, they are also directly confluent (note that this does not necessarily hold the other way around). To improve conflict detection, we proceed by showing how direct confluence can be lifted to an adequate rule conflict condition in the presence of attributes.

A critical pair for two given rules consists of a minimal context and two parallel dependent direct derivations. A minimal context of two rules is a graph (i) on which both rules are applicable and (ii) which only contains elements being matched by at least one of the rules. The intention behind critical pairs essentially consists in identifying those minimal conflict instances representing each possible conflict of the rules on any possible input graph. Practically, this requirement means that whenever two direct derivations are a conflict on some graph \( SG \), there is an element in the corresponding set of minimal conflict instances which is embedded in \( SG \). Embedding one pair of direct derivations (with input graph \( SK \)) into another pair of direct derivations (with input graph \( SG \)) means that there exist monomorphisms from the graphs of the first pair of derivations to the graphs of the second one.

The definition of critical pairs has only been considered in the framework of plain and attributed graphs before [4]. Nevertheless, it can be extended to symbolic graphs as follows.

**Definition 10** (Symbolic Critical Pair). A pair of symbolic rule applications \( SP_1 \xleftarrow{r_1} SK \xrightarrow{r_2} SP_2 \) with rules \( r_1 = (L_1 \xleftarrow{k_1} K_1 \xrightarrow{r_1} R_1, \Phi_1) \) and \( r_2 = (L_2 \xleftarrow{k_2} K_2 \xrightarrow{r_2} R_2, \Phi_2) \) on the input graph \( SK = (K, \Phi_K) \) is a symbolic critical pair if it is parallel dependent, \( D \models \Phi_K \iff o_1,\Phi(\Phi_1) \land o_2,\Phi(\Phi_2) \), and \( K \) is minimal meaning that each E-graph element \( ge \in K \) (i.e., node or edge in \( K \)) has a pre-image in the LHS of rule \( r_1 \) or \( r_2 \), i.e., \( ge \in o_1(L_1) \) or \( ge \in o_2(L_2) \).

**Example 5** (Symbolic Critical Pair). Figure 5 provides an example for a symbolic critical pair according to Definition 10. Again, we consider the rules \( r_1 \) and \( r_2 \) shown in the upper part of the figure. Contrary to the example for parallel dependence, the rules are now applied to the minimal context \( SK \) that contains only the elements required for applying the rules \( r_1 \) and \( r_2 \). As the resulting pair of direct derivations \( SP_1 \xleftarrow{r_1} SK \xrightarrow{r_2} SP_2 \) can be embedded into the direct derivations \( SH_1 \xleftarrow{r_{m1}} SG \xrightarrow{r_{m2}} SH_2 \) of Example 3, the pair \( SP_1 \xleftarrow{r_1} SK \xrightarrow{r_2} SP_2 \) is a minimal conflict instance of the conflict \( SH_1 \xleftarrow{r_{m1}} SG \xrightarrow{r_{m2}} SH_2 \).
This example has shown that the parallel dependence condition can be lifted to rule level by the concept of symbolic critical pairs. Analogously, we also lift the direct confluence condition to the level of rules instead of direct derivations, using a construction similar to minimal contexts. Unfortunately, when considering (general) symbolic graphs and symbolic graph transformation, a general problem arises when checking direct confluence, as is illustrated in the following example.

**Example 6** (Problem of Checking Direct Confluence). Figure 6 shows (in the upper part) the results $SP_1$ and $SP_2$ of the alternative direct derivations $SP_1 \xrightarrow{l_1, o_1} SK \xrightarrow{r_2, o_2} SP_2$ presented in Example 5 (shown in Figure 5). On the bottom (from left to right), the symbolic rules $r_2 = (L_2 \xrightarrow{l_2} K_2 \xrightarrow{r_2} R_2, \Phi_2)$ and $r_1 = (L_1 \xrightarrow{l_1} K_1 \xrightarrow{r_1} R_1, \Phi_1)$ are shown. In order to check direct confluence, both rules have to be applied to $SP_1$ and $SP_2$. However, this is not possible. If we want to find a symbolic match $o'_2 : (L_2, \Phi_2) \rightarrow SP_1$ from the left-hand side of rule $r_1$ defined by $(L_2, \Phi_2)$ to the symbolic graph $SP_1 = (P_1, \Phi_{P_1})$, we have to map label node $x$ of $L_2$ to label node $x'$ of $SP_1$. Mapping $x'$ of $L_2$ to $SP_1$ introduces two problems. The first problem is that no mapping of the label node $x''$ of $L_2$ to a label node in $SP_1$ exists such that $D \models (\Phi_{P_1} \Rightarrow o'_{2, \Phi}(\Phi_2))$. We can overcome this problem by assuming that $SP_1$ still includes an additional variable, not assigned to any node or edge and not appearing in the formula of $SP_1$. Generally, we assume from now on that a symbolic graph also contains an unlimited number of variables. Nevertheless, we have a second problem: we still cannot apply $r_2$ to $SP_1$ because $x' = x + 1$ and $x'' = x + 2$ does not imply $o'_{2, \Phi}(\Phi_2)$ which is $x''' = x' + 2$, where $x'''$ is the new additional variable for mapping $x''$ of $L_2$ to $P_1$ (i.e., $m_\Phi(x'') = x'''$).

This problem in Example 6 can be solved by narrowing graph transformation [12]. Instead of requiring that $\Phi_{P_1} \Rightarrow o'_{2, \Phi}(\Phi_2)$ holds before the transformation (as in the case of symbolic direct derivation), in the narrowing case, the transformation of the E-graph part is performed first and, afterwards, the satisfiability of $\Phi_{P_1} \land o'_{2, \Phi}(\Phi_1)$ is checked to ensure that the resulting symbolic graph has at least one instance.

**Definition 11** (Narrowing Graph Transformation [12]). Given a symbolic graph $SG = (G, \Phi_G)$, a symbolic graph transformation rule $r = (L \leftarrow K \rightarrow R, \Phi)$ and an E-graph morphism $m : L \rightarrow G$, the narrowing direct derivation of the rule $r$ on $SG$ at match $m$, denoted as $SG \Rightarrow_{r, m} SH$, leading to symbolic graph $SH = (H, \Phi_H)$, is given by the (E-graph) double pushout diagram below:
such that $\Phi_H := \Phi_G \land m'_p(\Phi)$ is satisfiable.

Now, we lift the notion of direct confluence to the rule level by using narrowing graph transformation.

**Definition 12** (Conflicting Pair). A symbolic critical pair $SCP = SP_1 \overset{r_1 \cdot o_1}{\Rightarrow} SK \overset{r_2 \cdot o_2}{\Rightarrow} SP_2$ is a conflicting pair if there do not exist narrowing direct derivations $SP_1 \Rightarrow_{r_2 \cdot o_2} SX_1$ and $SP_2 \Rightarrow_{r_1 \cdot o_1} SX_2$ such that $SCP$ is directly confluent.

Having these new concepts at hand, we can now revisit the concurrent applications of Example 3 to see if a conflict detection based on conflicting pairs is now capable of handling that situation.

**Example 7** (Non-conflicting Pair). Figure 7 depicts the construction process for a conflicting pair according to Definition 12, where $SP_1$ and $SP_2$ are part of the critical pair $SP_1 \overset{r_1 \cdot o_1}{\Rightarrow} SK \overset{r_2 \cdot o_2}{\Rightarrow} SP_2$ derived in Example 5 (Figure 5). Contrary to the previous example (Example 6) the rules $r_1$ and $r_2$ (depicted at the bottom right and right of Figure 7, respectively) are now applied using narrowing transformation as defined in Definition 11. We also assume that symbolic graphs $SP_1$ and $SP_2$ both include a new label node $x''$, which is used as image of the label nodes $x''$ and $x'$ in the (E-graph) matches $o'_1 : L_2 \rightarrow P_1$ and $o'_1 : L_1 \rightarrow P_2$, respectively. These mappings are depicted by the captions $[x'' \rightarrow x''']$ and $[x' \rightarrow x''']$ at the corresponding morphism arrows in Figure 7, respectively. The other mappings are depicted similarly, if the mapping differs from the mapping given by the node identifiers. The graphs $SY_1$ and $SY_2$ contain the results of the direct narrowing derivations of $r_1$ and $r_2$ at the matches $o'_1$ and $o'_2$. Consequently, the formula $\Phi_{Y_1} := \Phi_{P_1} \land o'_1(\Phi_{2})$ can be simplified to $\Phi_{Y_1} := \{x' = x + 1 \land x'' = x + 2 \land x''' = x'' + 2\}$ as we have mapped $x$ to $x'$ and $x''$ to $x'''$. Having $\Phi_{Y_2}$ transformed similarly, we have $\Phi_{Y_2} := \{x'' = x + 1 + 2 \land x'' = x + 2\}$ and $\Phi_{Y_2} := \{x''' = x + 2 + 1 \land x'' = x + 2\}$ which are equivalent. Hence, symbolic graphs $SY_1$ and $SY_2$ are isomorphic as both have the same graph structure and equivalent formulas.

Concluding the example, direct confluence as a conflict condition can be used on the rule level as well, if we adapt the way how graph transformation is performed.
4 An Improved Conflict Detection Process based on Direct Confluence

The notion of conflicting pairs (Def. 12) provides a basis for an improved conflict detection process. In this section, we describe this process. Thereupon, we show that the resulting set of conflicting pairs is 

*complete* in the usual sense, i.e., whenever there is a conflict, we have a conflicting pair embedded in the input graph, which represents the cause of the conflict [4].

A conflict detection based on conflicting pairs is not completely independent of a (classical) conflict detection based on critical pairs, but rather can be conceived as an extension to it. Such a conflict detection is performed on the rule level instead of the direct derivation level. Figure 8 summarizes the decision procedure.

In particular, given a pair of symbolic rules

\[ r_1 = (L_1 \leftarrow K_1 \rightarrow R_1, \Phi_1) \quad \text{and} \quad r_2 = (L_2 \leftarrow K_2 \rightarrow R_2, \Phi_2), \]

the overall process consists of the following steps:

1. A symbolic critical pair (Def. 10) is constructed if possible, based on \( L_1, L_2 \) and the matches. If the graph parts of \( L_1 \) and \( L_2 \) are non-overlapping, or \( D \not\models \Phi_1 \land \Phi_2 \) holds, there is no conflicting pair based on these two rules and the process terminates. Note that, for the E-graph part, there is always at least one minimal graph according to Def. 10.

2. If an appropriate \( SK = (K, \Phi_K) \) with a minimal \( K \) has been found in step 1, the direct derivations \( SK \xrightarrow{r_1 \circ o_1} SP_1 \) and \( SK \xrightarrow{r_2 \circ o_2} SP_2 \) (with the unique matches \( o_1 \) and \( o_2 \)) are to be checked for parallel dependence. In case they are parallel independent, there is no conflicting pair based on these two rules and the process terminates.

3. The rules are applied in both sequences to \( SK \); in case they are *not* directly confluent, then \( SK \), the rules \( r_1 \) and \( r_2 \) and their (unique) matches constitute a conflicting pair.

In the following, we prove that a conflict detection process defined this way is complete, i.e., when applied to a set of rules, the resulting set of conflicting pairs represents all possible conflict causes. This means that if for an arbitrary (symbolic) graph \( SG \), two direct derivations are not directly confluent, then a corresponding conflicting pair is embedded within \( SG \). In our proof, we rely on the construction of initial pushouts in symbolic graphs, analogously to the proof of Theorem 6.28 in [4].
**Definition 13** (Construction of Initial Pushouts in Symbolic Graphs). The diagram below is an *initial pushout in symbolic graphs* if (i) the morphisms $b, c \in M$, (ii) it is an initial pushout in E-graphs (see Def. 6.1 in [4]) and (iii) $\mathcal{D} \models (\Phi_B \Leftrightarrow \Phi_Y)$ and $\mathcal{D} \models (\Phi_C \Leftrightarrow \Phi_X)$.

$$
\begin{array}{c}
(B, \Phi_B) \xrightarrow{b \in M} (Y, \Phi_Y) \\
\downarrow \quad \downarrow \\
(C, \Phi_C) \xrightarrow{c \in M} (X, \Phi_X)
\end{array}
$$

**Theorem 1** (Completeness of Conflicting Pairs). Given a grounded symbolic graph $SG$ and a pair of not directly confluent direct derivations $\text{Der}_G = (SH_1 \xrightarrow{r_1, m_1} SG \xrightarrow{r_2, m_2} SH_2)$ of rules $r_1 = (L_1 \xrightarrow{L_1} K_1 \xrightarrow{r_1} R_1, \Phi_1)$ and $r_2 = (L_2 \xrightarrow{L_2} K_2 \xrightarrow{r_2} R_2, \Phi_2)$, there exists a conflicting pair $\text{Der}_K = (SP_1 \xrightarrow{r_1, o_1} SK \xrightarrow{r_2, o_2} SP_2)$ such that $\text{Der}_K$ can be embedded in $\text{Der}_G$ by $f : SK \rightarrow SG$, $g : SP_1 \rightarrow SH_1$ and $h : SP_2 \rightarrow SH_2$, shown in the diagram:

![Diagram showing initial pushouts in symbolic graphs](image)

**Proof.** First, we show that symbolic morphisms $f$, $g$ and $h$ exist.

As $\text{Der}_G$ is not directly confluent, it has to be parallel dependent. Due to the completeness of critical pairs (Lemma 6.22 in [4]), there exists a critical pair $\text{Der}_K$ in E-graphs with E-graph morphisms $f$, $g$ and $h$. Consequently, assuming that $\text{Der}_K$ is a symbolic critical pair (according to Def. 10), we have to show that $f$, $g$ and $h$ are symbolic graph morphisms.

Due to the existence of $\text{Der}_G$, we have $\mathcal{D} \models ((\Phi_G \Rightarrow m_1, \Phi(\Phi_1)) \land (\Phi_G \Rightarrow m_2, \Phi(\Phi_2)))$ which is equivalent to $\mathcal{D} \models (\Phi_G \Rightarrow m_1, \Phi(\Phi_1) \land m_2, \Phi(\Phi_2))$. From the minimality of critical pairs (i.e., $\mathcal{E} \Rightarrow \mathcal{M}$ pair factorization [4]), it follows that $m_1, \Phi = f_\Phi \circ o_1, \Phi$ and $m_2, \Phi = f_\Phi \circ o_2, \Phi$. We have $(m_1, \Phi(\Phi_1) \land m_2, \Phi(\Phi_2)) \Leftrightarrow (f_\Phi(o_1, \Phi(\Phi_1)) \land f_\Phi(o_2, \Phi(\Phi_2)))$. By factoring out $f_\Phi$, we get $f_\Phi(o_1, \Phi(\Phi_1) \land o_2, \Phi(\Phi_2)) \Leftrightarrow f_\Phi(\Phi_K)$. Hence, $\mathcal{D} \models (\Phi_G \Rightarrow f_\Phi(\Phi_K))$ and, thus, $f$ is a symbolic graph morphism.

To show that $g$ and $h$ are symbolic graph morphisms, we require $(\Phi_H \Leftrightarrow \Phi_{H_1} \Leftrightarrow \Phi_G)$ and $(\Phi_P \Leftrightarrow \Phi_K)$ as well as $(f_\Phi = g_\Phi = h_\Phi)$, which are consequences of Fact 1. If $\mathcal{D} \models (\Phi_G \Rightarrow f_\Phi(\Phi_K))$, also $\mathcal{D} \models (\Phi_H \Rightarrow g_\Phi(\Phi_P))$ and $\mathcal{D} \models (\Phi_H \Rightarrow h_\Phi(\Phi_P))$ and hence, $g$ and $h$ are symbolic graph morphisms.

We prove the rest of the theorem by contradiction. Let us suppose that there exist no symbolic direct derivations $SH_1 \xrightarrow{r_2, m_1'} SX_1$ and $SH_2 \xrightarrow{r_1, m_1'} SX_2$ with $SX_1$ and $SX_2$ being isomorphic, whereas, for the narrowing direct derivations $SP_1 \xrightarrow{r_2, o_1'} SY_1$ and $SP_2 \xrightarrow{r_1, o_1'} SY_2$, it holds that $SY_1$ and $SY_2$ are isomorphic. In order to prove that this supposition is indeed a contradiction, it suffices to show that if $SY_1$ and $SY_2$ are isomorphic, then $SH_1 \xrightarrow{r_2, m_1'} SX_1$ and $SH_2 \xrightarrow{r_1, m_1'} SX_2$ exist, and $SX_1$ and $SX_2$ are isomorphic.
In the following, we rely on the technique used in the proof of the Local Confluence Theorem (Theorem 6.28 in [4]), which is based on initial pushouts. We adapt this procedure to our setting of symbolic graphs with $M$-morphisms. Analogously to that proof, we first create an initial pushout over the morphism $f$ according to Def. 13. The pullback object $SZ$, defined in Property II of direct confluence (Def. 9) together with the closure property of initial pushouts (Lemma 6.5 in [4]) ensure that for each of the embedding morphisms, we have an initial pushout with $a : SB \to SC$. The diagram below shows the last step of this construction. As symbolic graphs with $M$-morphisms constitute an adhesive HLR category [4], we only have to show that the results of the narrowing transformations are compatible with the construction of initial pushouts.

In particular, we have to show that if (2a) is an initial pushout in symbolic graphs, then (1a) is a pushout in symbolic graphs.

As (1a) is a pushout in E-graphs, this statement is equivalent to show that (i) morphisms $c'_1 \circ c_1$ and $b'_1 \circ b_1$ are symbolic graph morphisms and (ii) for the pushout (1a), $D \models (Φ_Xi \Rightarrow p_Φ(Φ_Yi) \land c_{1,Φ}(c'_1)(Φ_C))$ holds.

(i). Since $c_1$ and $c'_1$ are both in $M$, we can assume (without loss of generality) that $Φ_C$ and $Φ_Xi$ are the same formulas, and $V^C = V^D_{Xi}$ are the same sets of variables. Hence, $c_{1,Φ}(c'_1)$ is the identity and, therefore, it is a symbolic graph morphism as $D \models (Φ_Xi \Rightarrow c_{1,Φ}(c'_1)(Φ_C))$ trivially holds. For morphism $b'_1 \circ b_1$, we have to show that $D \models (Φ_Yi \Rightarrow b_{1,Φ}(b'_1,Φ_b(Φ_B)))$ holds. By the definition of narrowing graph transformation, we have $Φ_Yi := Φ_Pi \land co_{2,Φ}(Φ_2)$. It follows from the existence of the initial pushout $(SC \leftarrow SB \to SP_1)$ that $Φ_B \equiv Φ_Pi$ and hence, we have that $D \models (Φ_Yi \Rightarrow b_{1,Φ}(b'_1,Φ_b(Φ_B)))$ is equivalent to $D \models ((Φ_Pi \land co_{2,Φ}(Φ_2)) \Rightarrow b_{1,Φ}(b'_1,Φ_b(Φ_B)))$ which holds as $b_1$ and $b'_1$ are both in $M$ and, therefore, $b_{1,Φ}(b'_1,Φ_b)$ is the identity.

(ii). $\Rightarrow$: We have to show that $D \models (Φ_Xi \Rightarrow p_Φ(Φ_Yi))$ and $D \models (Φ_Xi \Rightarrow c_{1,Φ}(c_{1,Φ}(Φ_C)))$ holds. While the latter has been already shown above, it remains to show that $D \models (Φ_Xi \Rightarrow p_Φ(Φ_Yi))$. With $Φ_Yi := Φ_Pi \land co_{2,Φ}(Φ_2)$ (from the definition of narrowing transformation), we have $(Φ_Xi \Rightarrow p_Φ(Φ_Yi)) \iff (Φ_Xi \Rightarrow p_Φ(Φ_Pi \land co_{2,Φ}(Φ_2)))$ which is equivalent to $(Φ_Xi \Rightarrow p_Φ(Φ_Pi)) \land (Φ_Xi \Rightarrow p_Φ(co_{2,Φ}(Φ_2)))$. Due to Fact 1, we have $g_Φ = p_Φ$; by the construction of the symbolic direct derivation $SH_1 \overset{m_1, m_2}{\Rightarrow} SX_1$, $Φ_Hi \iff Φ_Xi$ holds; therefore, $D \models (Φ_Xi \Rightarrow p_Φ(Φ_Pi))$ is equivalent to $D \models (Φ_Hi \Rightarrow g_Φ(Φ_Pi))$, which is given by the existence of the symbolic graph morphism $g : SP_1 \to SH_1$. It remains to show that $D \models (Φ_Xi \Rightarrow p_Φ(co_{2,Φ}(Φ_2)))$.

\[\begin{align*}
(L_2, Φ_2) & \overset{1}{\leftarrow} (K_2, Φ_2) \overset{2}{\rightarrow} (R_2, Φ_2) \\
(S_P2, s_1) & \overset{1}{\leftarrow} (S_N2, s_1) \overset{2}{\rightarrow} (S_Y2, s_1) \\
(S_H1, s_1) & \overset{1}{\leftarrow} (S_S1, s_1) \overset{2}{\rightarrow} (S_X1, s_1)
\end{align*}\]
which can be reformulated as $\Phi_X \Rightarrow cm'_2,\Phi$. The implication $D \models (\Phi_X \Rightarrow cm'_2,\Phi)$ holds due to the existence of the symbolic graph morphism $cm'_2$.

(ii). $\Leftarrow$: By the construction of initial pushouts, we have that $c'_1,\Phi(c_1,\Phi(\Phi_C)) \iff \Phi_X$ and hence $p_\Phi(\Phi_X) \land c'_1,\Phi(c_1,\Phi(\Phi_C)) \Rightarrow \Phi_X$.

We can show in the same way that (1b) is a pushout in symbolic graphs as well. It follows from the uniqueness of the pushout object that if $SY_1$ and $SY_2$ are isomorphic, so are $SX_1$ and $SX_2$.

This way, we have shown that our supposition contains a contradiction and, therefore, if $Der_K$ is not directly confluent, then $Der_K$ is a conflicting pair which can be embedded into $Der_G$.

This proof shows that our proposed notion of conflicting pairs effectively represents the minimal conflict instances based on direct confluence and, thus, provides a means to lift conflict detection to rule level. Moreover, the general nature of the proof also demonstrates that the proposed technique is not restricted to the attributed setting used as motivation. In fact, direct confluence and conflicting pairs can be effectively used as an incremental extension of the existing conflict results for plain graphs as well.

## 5 Related Work

**Symbolic graphs.** Symbolic graphs and symbolic graph transformation have been introduced by Orejas and Lambers in [11, 12] as a generalized and convenient representation for attributed graphs and attributed graph transformation. However, a proper notion of conflicts and a corresponding conflict detection process have not been considered in this framework.

**Conflicts.** The concept of conflicts has been adopted to graph transformation with negative application conditions and to attributed graph transformation with inheritance [8, 6]. In contrast to the proposed technique, these approaches rely on the notion of parallel dependence for determining conflicts. As a consequence, they still recognize a conflict whenever two rules access the same attribute and at least one modifies its value (regardless of the semantics of the access operations actually performed).

The concept of local confluence, which is a generalization of direct confluence, has its origins in term rewriting systems. The applicability of local confluence to attributed graph transformation is shown in [7]. However, in contrast to direct confluence, local confluence is undecidable even for graphs without attributes. Additionally, the transformation of term attributed graphs, which is required to check local confluence, requires term unification to be performed at every derivation step. Contrary, in the symbolic case, where the formula is constructed stepwise at the syntactical level and is validated afterwards, e.g., by using off-the-shelf SMT solvers.

**Refining conflict detection.** To the best of our knowledge, the only approach except for ours to formally capture and extend the notion of critical pairs is that of Lambers et al. [9]. They also try to narrow the set of actual conflicts, however, their approach is based on directly expressing the actual conflict cause by means of categorical notions and not on giving a new condition for checking which conflicts are considered relevant.

From a practical perspective, the approach of Cabot et al. [1] presents a fully-fledged graph transformation tool framework which also incorporates an analysis of graph transformation rules to verify certain properties, where their concept of conflict and independence strongly corresponds to our notion of direct confluence. The authors also remark that, similar to our technique, they only have to test the minimal models for those properties. Nevertheless, the approach of [1] is completely practical and it is based on a preceding translation of the rules into OCL expressions and, therefore, the theoretical aspects of our approach are not considered at all.
6 Conclusion

In this paper, we have proposed an improved conflict detection procedure for graph transformation with attributes. Our approach uses symbolic graphs as a framework and is based on the notion of conflicts and direct confluence. This way, we are able to explicitly take the intention of the attribute operations during conflict detection into account and to potentially exclude some false positive conflicts, emerging from the conservative conflict condition of earlier approaches, while still retaining completeness.

Based on this formal framework, we aim at implementing the approach using an off-the-shelf SMT solver, e.g., Z3, MathSAT or SMTInterpol [10, 3, 2] and perform experiments regarding applicability and performance. Furthermore, we plan to apply this implementation to conduct case studies comprising modeling languages apparent in model-driven engineering.

References

[1] Jordi Cabot, Robert Clarisó, Esther Guerra & Juan de Lara (2010): A UML/OCL Framework for the Analysis of Graph Transformation Rules. SoSyM 9(3), pp. 335–357, doi:10.1007/s10270-009-0129-0.

[2] Jürgen Christ, Jochen Hoenicke & Alexander Nutz (2012): SMTInterpol: An Interpolating SMT Solver. In: Model Checking Software - 19th International Workshop, SPIN 2012, Oxford, UK, pp. 248–254, doi:10.1007/978-3-642-31759-0_19.

[3] Alessandro Cimatti, Alberto Griggio, Bastiaan Schaafsma & Roberto Sebastiani (2013): The MathSAT5 SMT Solver. In: Proceedings of TACAS, LNCS 7795, Springer, doi:10.1007/978-3-642-36742-7_7.

[4] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange & Gabriele Taentzer (2006): Fundamentals of Algebraic Graph Transformation. Springer.

[5] Hartmut Ehrig & Bernd Mahr (1985): Fundamentals of Algebraic Specification I. Springer, doi:10.1007/978-3-642-69962-7.

[6] Ulrike Golas, Leen Lambers, Hartmut Ehrig & Fernando Orejas (2012): Attributed Graph Transformation with Inheritance: Efficient Conflict Detection and Local Confluence Analysis using Abstract Critical Pairs. Theoretical Computer Science 424(0), pp. 46 – 68, doi:10.1016/j.tcs.2012.01.032.

[7] Reiko Heckel, Jochen Malte Küster & Gabriele Taentzer (2002): Confluence of Typed Attributed Graph Transformation Systems. In: Proc. of the 1st ICGT, LNCS 2505, Springer, pp. 161–176, doi:10.1007/3-540-45832-8_14.

[8] Leen Lambers, Hartmut Ehrig & Fernando Orejas (2006): Conflict Detection for Graph Transformation with Negative Application Conditions. In: Graph Transformations, LNCS 4178, Springer, pp. 61–76, doi:10.1007/11841883_6.

[9] Leen Lambers, Hartmut Ehrig & Fernando Orejas (2008): Efficient Conflict Detection in Graph Transformation Systems by Essential Critical Pairs. ENTCS 211, pp. 17–26, doi:10.1016/j.entcs.2008.04.026.

[10] Leonardo de Moura & Nikolaj Bjørner (2008): Z3: An Efficient SMT Solver. In: Tools and Algorithms for the Construction and Analysis of Systems, LNCS 4963, Springer, pp. 337–340, doi:10.1007/978-3-540-78800-3_24.

[11] Fernando Orejas & Leen Lambers (2010): Symbolic Attributed Graphs for Attributed Graph Transformation. In: Proc. of the ICGT, Electronic Communications of the EASST 30.

[12] Fernando Orejas & Leen Lambers (2012): Lazy Graph Transformation. Fundam. Inf. 118(1-2), pp. 65–96. Available at http://dl.acm.org/citation.cfm?id=2385016.2385020.

[13] G. Rozenberg, editor (1997): Handbook of Graph Grammars and Computing by Graph Transformation: Volume I. Foundations. World Scientific Publishing Co., Inc., River Edge, NJ, USA.

[14] Joseph R Shoenfield (1967): Mathematical logic. 21, Addison-Wesley Reading.