Local Confluence for Rules with Nested Application Conditions

Hartmut Ehrig[1], Annegret Habel[2], Leen Lambers[3], Fernando Orejas[4], and Ulrike Golas[1]

[1] Technische Universität Berlin, Germany
{ehrig,ugolas}@cs.tu-berlin.de
[2] Carl v. Ossietzky Universität Oldenburg, Germany
habel@informatik.uni-oldenburg.de
[3] Hasso Plattner Institut, Universität Potsdam, Germany
Leen.Lambers@hpi.uni-potsdam.de
[4] Universitat Politècnica de Catalunya, Spain
orejas@lsi.upc.edu

Abstract. Local confluence is an important property in many rewriting and transformation systems. The notion of critical pairs is central for being able to verify local confluence of rewriting systems in a static way. Critical pairs are defined already in the framework of graphs and adhesive rewriting systems. These systems may hold rules with or without negative application conditions. In this paper however, we consider rules with more general application conditions — also called nested application conditions — which in the graph case are equivalent to finite first-order graph conditions. The classical critical pair notion denotes conflicting transformations in a minimal context satisfying the application conditions. This is no longer true for combinations of positive and negative application conditions — an important special case of nested ones — where we have to allow that critical pairs do not satisfy all the application conditions. This leads to a new notion of critical pairs which allows to formulate and prove a Local Confluence Theorem for the general case of rules with nested application conditions. We demonstrate this new theory on the modeling of an elevator control by a typed graph transformation system with positive and negative application conditions.

Keywords: Critical pairs, local confluence, nested application conditions.

1 Introduction

Confluence is a most important property for many kinds of rewriting systems. For instance, in the case of rewriting systems that are used as a computation model, confluence is the property that ensures the functional behaviour of a given system [1]. This is important in the case of graph transformation systems (GTSs) that are used to specify the functionality of a given software system,

or to describe model transformations (see, e.g. [2]). Unfortunately, confluence of rewriting systems is undecidable. A special case is local confluence which, in the case of terminating rewriting systems, is equivalent to confluence. In addition, local confluence is also interesting in the sense that it shows the absence of some kinds of conflicts in the application of rules. The standard approach for proving local confluence was originally developed by Knuth and Bendix [3] for term rewriting systems (TRSs) [1]. The idea of this approach is to check the joinability (or confluence) of *critical pairs*, which represent conflicting rules in a minimal context, the minimal possible sources of non-confluence. This technique has also been applied to check local confluence for GTSs (see, e.g. [4,5,2]). However, checking local confluence for GTSs is quite more difficult than for TRSs. Actually, local confluence is undecidable for terminating GTSs, while it is decidable for terminating TRSs [5].

In standard GTSs, if we find a valid match of the left-hand side of a rule into a given graph, that rule can be applied to that graph. But we may want to limit the applicability of rules. This leads to the notion of *application conditions (AC)*, constraining the matches of a rule that are considered valid. An important case of an application condition is a *negative application condition (NAC)*, as introduced in [6], which is just a graph N that extends the left hand side of the rule. Then, that rule is applicable to a graph G if – roughly speaking – N is not present in G. However, the use of application conditions poses additional problems for constructing critical pairs. The first results on checking local confluence for GTS with ACs are quite recent and are restricted to the case of NACs [7]. Although NACs are quite useful already, they have limited expressive power: we cannot express forbidden contexts which are more complex than a graph embedding, nor can we express positive requirements on the context of applications, i.e. positive application conditions. The running example used in this paper, describing the specification of an elevator system, shows that this increase of descriptive power is needed in practical applications.

To overcome the expressive limitations of NACs, in [8] a very general kind of conditions, called *nested application conditions*, is studied in detail. In particular, in the graph case, this class of conditions has the expressive power of the first-order fragment of Courcelle's logic of graphs [9]. Following that work, in this paper we study the local confluence of transformation systems, where the rules may include arbitrary nested application conditions. However, dealing with this general kind of conditions poses new difficulties. In particular, NACs are in a sense monotonic. If a match does not satisfy a NAC then any other match embedding that one will not satisfy the NAC either [7]. This is not true for ACs in general and the reason why a different kind of critical pair is needed.

The paper is organized as follows: In Section 2, we review the problems related to checking the confluence of GTSs. In Section 3, we introduce the concepts of nested application conditions and rules. Moreover, we also introduce our running example on an elevator control, which is used along the paper to illustrate the main concepts and results. In Section 4, we define our new notion of critical pairs and state a completeness theorem (Theorem 1), saying that every pair of

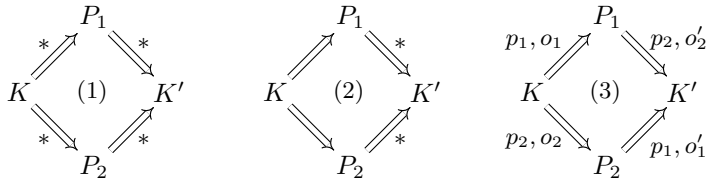
dependent direct derivations is an extension of a critical pair. In Section 5, we state as main result the Local Confluence Theorem (Theorem 2) for rules with ACs. A conclusion including related work is given in Section 6.

The presentation is, to some extent, informal (e.g., no proofs are included), but a technical report [10] supplies all the missing technical details. Moreover, our explanations may suggest that our results just apply to standard graph transformation systems, even if our examples use typed graphs. Actually, as can be seen in [10], our results apply to transformation systems[5] over any adhesive category [11] with an epi-$\mathcal{M}$-factorization (used in Lemma 1), a unique $\mathcal{E}$-$\mathcal{M}$ pair factorization (used in Thm 1) where $\mathcal{M}$ is the class of all monomorphisms, and with initial pushouts over $\mathcal{M}$-morphisms (used in Thm 2). The category $\mathbf{Graphs}_{TG}$ of graphs and morphisms typed over TG is adhesive and holds these properties [2]. In particular, a unique $\mathcal{E}$-$\mathcal{M}$ pair factorization is obtained when $\mathcal{E}$ is the class of pairs of jointly surjective graph morphisms and $\mathcal{M}$ is the class of injective morphisms.

2 Confluence in Graph Transformation

In this section, we review some results about the local confluence of graph transformation systems. In this sense, we assume that the reader has a reasonably good knowledge of the double-pushout approach to graph transformation. For more detail, the interested reader may consult e.g. [2].

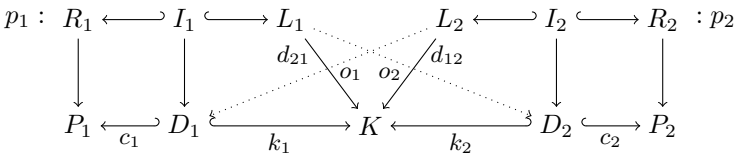
Confluence is the property ensuring the functional behavior of transformation systems. A system is confluent if whenever a graph K can be transformed into two graphs P_1 and P_2, these graphs can be transformed into a graph K', as shown in diagram (1) below. A slightly weaker property than confluence is local confluence, represented by diagram (2) below. In this case, we just ask that P_1 and P_2 can be transformed into K' when these graphs can be obtained in exactly one transformation step from K. Confluence coincides with local confluence when the given transformation system is terminating, as shown by Newman in [12].



A result in the theory of Graph Transformation that ensures confluence of some derivations is the Local Church-Rosser Theorem for parallel independent transformations (see, e.g., [2]). In particular, the application of two rules p_1 and p_2

[5] The results can be generalized also to transformation systems over (weak) adhesive high-level replacement categories [2] like petri nets, hypergraphs, and algebraic specifications.

with matches o_1 and o_2 to a graph K are parallel independent, essentially, if none of these applications deletes any element which is part of the match of the other rule. More precisely, if given the diagram below depicting the two double pushouts defined by these applications there exist morphisms $d_{12}\colon L_1 \to D_2$ and $d_{21}\colon L_2 \to D_1$ such that the diagram commutes. Then, the Local Church-Rosser Theorem states that in this situation we may apply these rules to P_1 and P_2 so that we obtain K' in both cases, as depicted by diagram (3) above. In general, however, not all pairs of transformations are parallel independent. It remains to analyze the parallel dependent ones.



The intuition of using critical pairs to check (local) confluence is that we do not have to study all the possible cases of pairs of rule applications which are parallel dependent, but only some minimal ones which are built by gluing the left-hand side graphs of each pair of rules. A *critical pair* for the rules p_1, p_2 is a pair of parallel dependent transformations, $P_1 \Leftarrow_{p_1,o_1} K \Rightarrow_{p_2,o_2} P_2$, where o_1 and o_2 are jointly surjective. A completeness lemma, showing that every pair of parallel dependent rule applications embeds a critical pair, justifies why it is enough to consider the confluence of these cases.

As shown in [4,5], the confluence of all critical pairs is not a sufficient condition for the local confluence of a graph transformation system (as it is in the case of term rewriting). Suppose that we have two parallel dependent rule applications $G_1 \Leftarrow G \Rightarrow G_2$. By the completeness of critical pairs, there exists a critical pair $P_1 \Leftarrow K \Rightarrow P_2$ which is embedded in $G_1 \Leftarrow G \Rightarrow G_2$. Now, suppose that there are derivations from P_1 and P_2 into a common graph K', i.e. $P_1 \overset{*}{\Rightarrow} K' \overset{*}{\Leftarrow} P_2$. We could expect that these two derivations could be embedded into some derivations $G_1 \overset{*}{\Rightarrow} G' \overset{*}{\Leftarrow} G_2$ and, hence, the confluence of the critical pair would imply the confluence of $G_1 \Leftarrow G \Rightarrow G_2$. However, this is not true in general. For instance, if we try to apply in a larger context the rules in the derivations $P_1 \overset{*}{\Rightarrow} K' \overset{*}{\Leftarrow} P_2$ the gluing conditions may not hold and, as a consequence, applying these rules may be impossible. But this is not the only problem. Even if the transformations $P_1 \overset{*}{\Rightarrow} K'$ and $P_2 \overset{*}{\Rightarrow} K'$ can be embedded into transformations of $G_1 \overset{*}{\Rightarrow} H_1$ and $G_2 \overset{*}{\Rightarrow} H_2$, respectively, in general we cannot ensure that H_1 and H_2 are isomorphic. To avoid this problem, a stronger notion of confluence is needed. This notion is called *strict confluence*, but in this paper we will call it *plain strict confluence*, in the sense that it applies to transformations with *plain rules*, i.e. rules without application conditions. Essentially, a critical pair $P_1 \Leftarrow_{p_1,o_1} K \Rightarrow_{p_2,o_2} P_2$ is plain strictly confluent if there exist derivations $P_1 \overset{*}{\Rightarrow} K' \overset{*}{\Leftarrow} P_2$ such that every element in K which is preserved by the two transformations defining the critical pair is also preserved by the two derivations. In [5,2] it is

shown for plain rules that plain strict confluence of all critical pairs implies the local confluence of a graph transformation system.

3 Rules with Nested Application Conditions

In this section, we introduce nested application conditions (in short, application conditions, or just ACs) and define how they can be used in graph transformation. Moreover, we introduce our running example on the modeling of an elevator control using typed graph transformation rules with application conditions.

Example 1 (*Elevator*). The type of control that we consider is meant to be used in buildings where the elevator should transport people from or to one main stop. This situation occurs, for example, in apartment buildings or multi-storey car parks. Each floor in the building is equipped with one button in order to call the elevator. The elevator car stops at a floor for which an internal stop request is given. External call requests are served by the elevator only if it is in downward mode in order to transport people to the main stop. The direction of the elevator car is not changed as long as there are remaining requests in the running direction. External call requests as well as internal stop requests are not deleted until the elevator car has arrived.

On the right of Fig. 1, a type graph TG for *Elevator* is depicted. This type graph expresses that an elevator car of type elevator exists, which can be on a specific floor. Moreover, the elevator can be in upward or downward mode. Floors are connected by next_up edges expressing which floor is directly above another floor. Moreover, higher_than edges express that a floor is arranged higher in the building than another floor. Each floor can hold requests of two different types. The first type is a call request expressing that an external call for the elevator car on this floor is given. The second type is a stop request expressing that a call in the elevator car is given for stopping it on this floor. On the left of Fig. 1 a graph G, typed over this type graph is shown, describing a four-storey building, where the elevator car is on the second floor in downward mode with a call request on the ground floor. Note that G contains higher_than edges from each floor which is higher than each other floor (corresponding to transitive closure of opposite edges of next_up), but they are not depicted because of visibility reasons. In Example 3, some graph transformation rules with ACs modeling the elevator control are presented. We continue with the running example in Section 4 and 5, where we explain how to conclude local confluence for the parallel dependent pair of transformations with ACs as depicted in Fig. 3 by analyzing the corresponding critical pair with ACs.

Nested application conditions are defined in [8]. They generalize the corresponding notions in [6,13,14], where a negative (positive) application condition, short NAC (PAC), over a graph P, denoted $\neg\exists a$ ($\exists a$) is defined in terms of a morphism $a : P \to C$. Informally, a morphism $m : P \to G$ satisfies $\neg\exists a$ ($\exists a$) if there does not exist a morphism $q : C \to G$ extending m (if there exists q extending

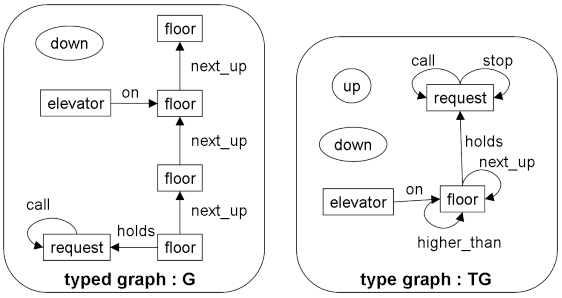


Fig. 1. A typed graph of *Elevator* together with its type graph

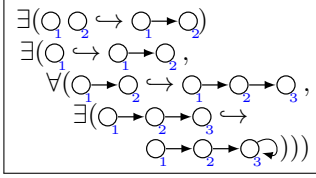
m). Then, an AC (also called *nested AC*) is either the special condition true or a pair of the form $\exists(a, \mathrm{ac}_C)$ or $\neg\exists(a, \mathrm{ac}_C)$, where the first case corresponds to a PAC and the second case to a NAC, and in both cases ac_C is an additional AC on C. Intuitively, a morphism $m : P \to G$ satisfies $\exists(a, \mathrm{ac}_C)$ if m satisfies a and the corresponding extension q satisfies ac_C. Moreover, ACs (and also NACs and PACs) may be combined with the usual logical connectors.

Definition 1 (application condition). An *application condition* ac_P over a graph P is inductively defined as follows: true is an application condition over P. For every morphism $a\colon P \to C$ and every application condition ac_C over C, $\exists(a, \mathrm{ac}_C)$ is an application condition over P. For application conditions c,c_i over P with $i \in I$ (for all index sets I), $\neg c$ and $\wedge_{i\in I} c_i$ are application conditions over P. We define inductively when a morphism *satisfies* an application condition: Every morphism satisfies true. A morphism $p\colon P \to G$ satisfies an application condition $\exists(a, \mathrm{ac}_C)$, denoted $p \models \exists(a, \mathrm{ac}_C)$, if there exists an injective morphism q such that $q \circ a = p$ and $q \models \mathrm{ac}_C$. A morphism $p\colon P \to G$ satisfies $\neg c$ if p does not satisfy c and satisfies $\wedge_{i\in I} c_i$ if it satisfies each c_i $(i \in I)$.

$$\exists\Big(\ P \xrightarrow{\ a\ } C,\ \blacktriangleleft\ \mathrm{ac}_C\ \Big) \qquad \begin{array}{ccc} & & C \\ & \overset{=}{\nearrow} & \not\models \\ P & & \\ & {\scriptstyle p}\searrow \quad \swarrow{\scriptstyle q} & \\ & G & \end{array}$$

Note that $\exists a$ abbreviates $\exists(a, \mathrm{true})$ and $\forall(a, \mathrm{ac}_C)$ abbreviates $\neg\exists(a, \neg\mathrm{ac}_C)$.

Example 2. Examples of ACs are given below, where the first one is a standard PAC considered already in [6], while the second one is properly nested. Note that $\hookrightarrow$ denotes the inclusion.



There is an edge from the image of 1 to that of 2.

For the image of node 1, there exists an outgoing edge such that, for all edges outgoing from the target, the target has a loop.

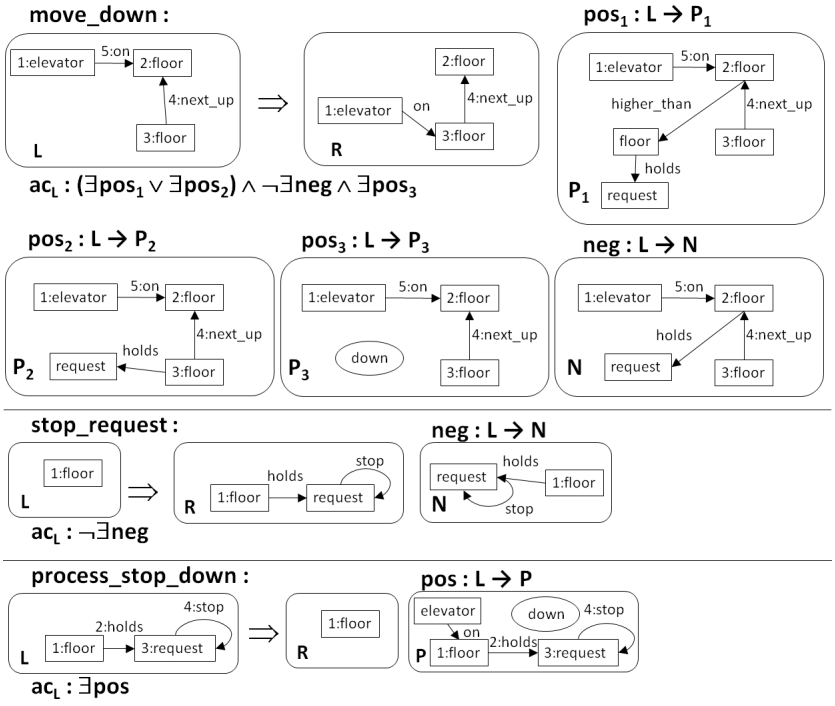


Fig. 2. Rules for *Elevator*

ACs are used to restrict the application of graph transformation rules to a given graph. The idea is to equip the left-hand side of rules with an application condition[6]. Then we can only apply a given rule to a graph G if the corresponding match morphism satisfies the AC of the rule. However, for technical reasons[7], we also introduce the application of rules *disregarding* the associated ACs.

Definition 2 (rules and transformations with ACs). A *rule* $\rho = \langle p, \mathrm{ac}_L \rangle$ consists of a *plain* rule $p = \langle L \hookleftarrow I \hookrightarrow R \rangle$ with $I \hookrightarrow L$ and $I \hookrightarrow R$ injective morphisms and an application condition ac_L over L.

$$\begin{array}{ccccc} \mathrm{ac}_L & L & \hookleftarrow & I & \hookrightarrow R \\ & \Vdash & & & \\ & m \downarrow & (1) & \downarrow & (2) \quad \downarrow m^* \\ & G & \hookleftarrow & D & \hookrightarrow H \end{array}$$

A *direct transformation* consists of two pushouts (1) and (2), called DPO, with match m and comatch m^* such that $m \models \mathrm{ac}_L$. An *AC-disregarding direct trans-*

[6] We could have also allowed to equip the right-hand side of rules with an additional AC, but Lemma 2 shows that this case can be reduced to rules with left ACs only.

[7] For example, a critical pair with ACs (see Def. 5) consists of transformations that do not need to satisfy the associated ACs.

formation $G \Rightarrow_{\rho,m,m^*} H$ consists of DPO (1) and (2), where m does not necessarily need to satisfy ac_L.

Example 3 (typed graph rules of *Elevator*). We show three rules (only their left and right-hand sides[8]) modeling part of the elevator control as given in Example 1. In Fig. 2, first, we have rule move_down with combined AC on L, consisting of three PACs ($\mathrm{pos}_i \colon L \to P_i, i = 1, \ldots, 3$) and a NAC ($\mathrm{neg} \colon L \to N$), describing that the elevator car moves down one floor under the condition that some request is present on the next lower floor (pos_2) or some other lower floor (pos_1)[9], no request is present on the elevator floor (neg), and the elevator car is in downward mode (pos_3). As a second rule, we have stop_request, describing that an internal stop request is made on some floor under the condition that no stop request is already given for this floor. Rule process_stop_down describes that a stop request is processed for a specific floor under the condition that the elevator is on this floor and it is in downward mode.

Application conditions can be shifted over morphisms.

Lemma 1 (shift of ACs over morphisms [8,15]). There is a transformation Shift from morphisms and application conditions to application conditions such that for each application condition ac_P and for each morphisms $b \colon P \to P'$, Shift transforms ac_P via b into an application condition $\mathrm{Shift}(b, \mathrm{ac}_P)$ over P' such that for each morphism $n \colon P' \to H$ it holds that $n \circ b \models \mathrm{ac}_P \Leftrightarrow n \models \mathrm{Shift}(b, \mathrm{ac}_P)$.

Construction. The transformation Shift is inductively defined as follows:

$$\begin{array}{ccc} P & \xrightarrow{b} & P' \\ {\scriptstyle a}\downarrow & (1) & \downarrow{\scriptstyle a'} \\ C & \xhookrightarrow[b']{} & C' \\ \triangle & & \\ \mathrm{ac}_C & & \end{array}$$

$\mathrm{Shift}(b, \mathrm{true}) = \mathrm{true}$.
$\mathrm{Shift}(b, \exists(a, \mathrm{ac}_C)) = \bigvee_{(a',b') \in \mathcal{F}} \exists(a', \mathrm{Shift}(b', \mathrm{ac}_C))$ if $\mathcal{F} = \{(a', b') \mid (a', b')$ jointly epimorphic, $b' \in \mathcal{M}$, (1) commutes$\} \neq \emptyset$
$\mathrm{Shift}(b, \exists(a, \mathrm{ac}_C)) = \mathrm{false}$ if $\mathcal{F} = \emptyset$. For Boolean formulas over ACs, Shift is extended in the usual way.

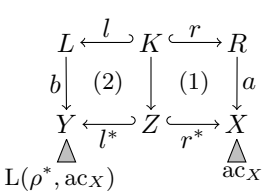
Application conditions of a rule can be shifted from right to left and vice versa. We only describe the right to left case here, since the left to right case is symmetrical.

Lemma 2 (shift of ACs over rules [8]). There is a transformation L from rules and application conditions to application conditions such that for every application condition ac_R on R of a rule ρ, L transforms ac_R via ρ into the application condition $\mathrm{L}(\rho, \mathrm{ac}_R)$ on L such that we have for every direct transformation $G \Rightarrow_{\rho,m,m^*} H$ that $m \models \mathrm{L}(\rho, \mathrm{ac}_R) \Leftrightarrow m^* \models \mathrm{ac}_R$.

[8] Thereby, the intermediate graph I and span monomorphisms can be derived as follows. Graph I consists of all nodes and edges occurring in L and R that are labeled by the same number. The span monomorphisms map nodes and edges according to the numbering. Analogously, the morphisms of the ACs consist of mappings according to the numbering of nodes and edges.

[9] Since we check that ACs are satisfied by inspecting the presence or absence of injective morphisms, we need pos_1 as well as pos_2.

Construction. The transformation L is inductively defined as follows:



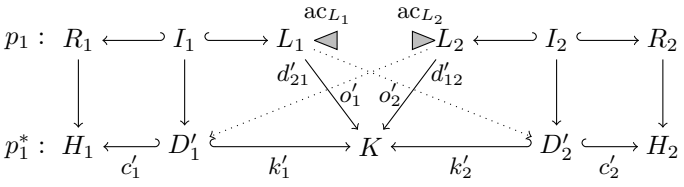
$\mathrm{L}(\rho, \mathrm{true}) = \mathrm{true}$
$\mathrm{L}(\rho, \exists(a, \mathrm{ac}_X)) = \exists(b, \mathrm{L}(\rho^*, \mathrm{ac}_X))$ if $\langle r, a \rangle$ has a pushout complement (1) and $\rho^* = \langle Y \hookleftarrow Z \hookrightarrow X \rangle$ is the derived rule by constructing the pushout (2).
$\mathrm{L}(\rho, \exists(a, \mathrm{ac}_X)) = \mathrm{false}$, otherwise. For Boolean formulas over ACs, L is extended in the usual way.

4 Critical Pairs and Completeness

The use of application conditions in transformation rules complicates considerably the analysis of the confluence of a graph transformation system. For instance, two rule applications that are parallel independent if we disregard the ACs may not be confluent when considering these conditions, as we can see in Example 4 below. In this section we present the kind of critical pairs that are needed for ensuring the local confluence of a transformation system. First, we present the notion of parallel dependence for the application of rules with ACs, in order to characterize all the confluence conflicts. Then, we present a simple but weak notion of critical pair, which is shown to be complete, in the sense that every parallel dependent pair of rule applications embeds a weak critical pair. However, not every weak critical pair may be embedded in a parallel dependent pair of rule applications. To end this section, we present the adequate notion of critical pair, in the sense that they are also complete and each of them is embedded in, at least, one case of parallel dependence.

The intuition of the concept of *parallel independence*, when working with rules with ACs, is quite simple. We need not only that each rule does not delete any element which is part of the match of the other rule, but also that the resulting transformation defined by each rule application still satisfies the ACs of the other rule application. More precisely:

Definition 3 (parallel independence with ACs). A pair of direct transformations $H_1 \Leftarrow_{\rho_1, o_1'} G \Rightarrow_{\rho_2, o_2'} H_2$ with ACs is parallel independent if there exists a morphism $d_{12}' \colon L_1 \to D_2'$ such that $k_2' \circ d_{12}' = o_1'$ and $c_2' \circ d_{12}' \models \mathrm{ac}_{L_1}$ and there exists a morphism $d_{21}' \colon L_2 \to D_1'$ such that $k_1' \circ d_{21}' = o_2'$ and $c_1' \circ d_{21}' \models \mathrm{ac}_{L_2}$.



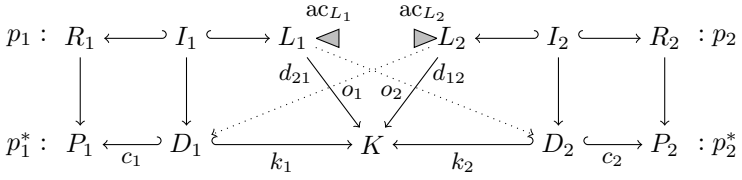
The intuition of the notion of *weak critical pair* is also quite simple. We know that all pairs of rule applications are potentially non confluent, even if they are

parallel independent when disregarding the ACs. Then, we define as weak critical pairs all the minimal contexts of all pairs of AC-disregarding rule applications.

Definition 4 (weak critical pair). Given rules $\rho_1 = \langle p_1, \mathrm{ac}_{L_1}\rangle$ and $\rho_2 = \langle p_2, \mathrm{ac}_{L_2}\rangle$, a *weak critical pair* for $\langle \rho_1, \rho_2\rangle$ is a pair $P_1 \Leftarrow_{\rho_1,o_1} K \Rightarrow_{\rho_2,o_2} P_2$ of AC-disregarding transformations, where o_1 and o_2 are jointly surjective.Every weak critical pair induces the ACs ac_K and ac^*_K on K defined by:
$\mathrm{ac}_K = \mathrm{Shift}(o_1, \mathrm{ac}_{L_1}) \wedge \mathrm{Shift}(o_2, \mathrm{ac}_{L_2})$, called *extension AC*, and
$\mathrm{ac}^*_K = \neg(\mathrm{ac}^*_{K,d_{12}} \wedge \mathrm{ac}^*_{K,d_{21}})$, called *conflict-inducing AC*
with $\mathrm{ac}^*_{K,d_{12}}$ and $\mathrm{ac}^*_{K,d_{21}}$ given as follows

$$\begin{array}{l}\text{if } (\exists\ d_{12} \text{ with } k_2 \circ d_{12} = o_1) \text{ then}\\ \quad \mathrm{ac}^*_{K,d_{12}} = \mathrm{L}(p^*_2, \mathrm{Shift}(c_2 \circ d_{12}, \mathrm{ac}_{L_1})) \text{ else } \mathrm{ac}^*_{K,d_{12}} = \mathrm{false}\\ \text{if } (\exists\ d_{21} \text{ with } k_1 \circ d_{21} = o_2) \text{ then}\\ \quad \mathrm{ac}^*_{K,d_{21}} = \mathrm{L}(p^*_1, \mathrm{Shift}(c_1 \circ d_{21}, \mathrm{ac}_{L_2})) \text{ else } \mathrm{ac}^*_{K,d_{21}} = \mathrm{false}\end{array}$$

where $p^*_1 = \langle K \overset{k_1}{\hookleftarrow} D_1 \overset{c_1}{\hookrightarrow} P_1\rangle$ and $p^*_2 = \langle K \overset{k_2}{\hookleftarrow} D_2 \overset{c_2}{\hookrightarrow} P_2\rangle$ are defined by the corresponding double pushouts.



The two ACs, ac_K and ac^*_K, are used to characterize the extensions of K that may give rise to a confluence conflict. If $m : K \to G$ and $m \models \mathrm{ac}_K$ then $m \circ o_1$ and $m \circ o_2$ are two matches of p_1 and p_2, respectively, that satisfy their associated ACs. If these two rule applications are parallel independent when disregarding the ACs then ac^*_K is precisely the condition that ensures that the two applications are parallel dependent when considering the ACs. That is, if $m \models \mathrm{ac}^*_K$ then the two transformations $H_1 \Leftarrow_{\rho_1, m\circ o_1} G \Rightarrow_{\rho_2, m\circ o_2} H_2$ are parallel dependent.

Example 4 (weak critical pair). Consider the parallel dependent pair of direct transformations $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ with ACs in Fig. 3. Note that these transformations are plain parallel independent. However, they are parallel dependent because (see Fig. 2) rule move_down can not be applied to H_2 since rule stop_request adds a stop request to the elevator floor, which is forbidden by the AC of rule move_down. The weak critical pair $P_1 \Leftarrow K \Rightarrow P_2$ for the rules move_down and stop_request that is embedded in the above parallel dependent transformations is depicted in Figure 5 at the end of Section 5. In this case, K coincides with the left-hand side of move_down. Note that this weak critical pair consists of a pair of AC-disregarding transformations. In particular, ac_L of rule move_down is not fulfilled in K, because e.g. the PAC $\exists \mathrm{pos}_3$ is not satisfied by

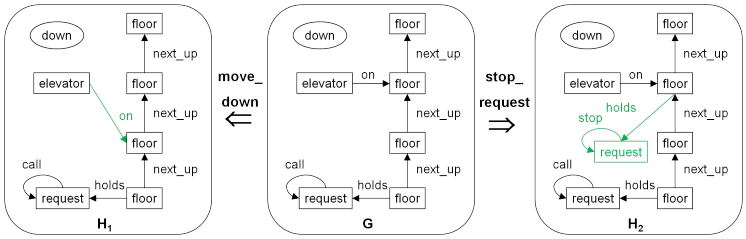


Fig. 3. Parallel dependent pair of direct transformations with ACs of *Elevator*

$o_1 = id_L$. The extension condition ac_K of this weak critical pair is shown in Fig. 4. It is equal to the conjunction of the AC of rule move_down and $\nexists \mathrm{neg}_2$, stemming from the NAC of rule stop_request by shifting over morphism o_2 (see Lemma 1). The conflict-inducing AC ac^*_K is a bit more tedious to compute, but it turns out to be equivalent to true for each monomorphic extension morphism, because it holds a PAC of the form $\exists \mathrm{id}_K$. In particular, this means that any pair of transformations $H_1 \Leftarrow G \Rightarrow H_2$, embedding that critical pair, are parallel dependent since the corresponding extension would trivially satisfy ac^*_K.

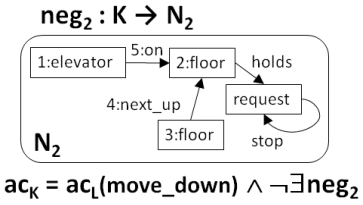


Fig. 4. ac_K for critical pair of *Elevator*

As a consequence of the definition of ac_K and ac^*_K, it can be proven that an extension $m : K \to G$ satisfies ac_K and ac^*_K if and only if the transformations $H_1 \Leftarrow_{\rho_1, m \circ o_1} G \Rightarrow_{\rho_2, m \circ o_2} H_2$ are parallel dependent. This means that weak critical pairs without an extension m satisfying ac_K and ac^*_K are useless for checking local confluence. Our notion of critical pair makes use of this fact. In particular, a critical pair is a weak critical pair such that there is at least one extension satisfying ac_K and ac^*_K.

Definition 5 (critical pair). Given rules $\rho_1 = \langle p_1, \mathrm{ac}_{L_1} \rangle$ and $\rho_2 = \langle p_2, \mathrm{ac}_{L_2} \rangle$, a *critical pair* for $\langle \rho_1, \rho_2 \rangle$ is a weak critical pair $P_1 \Leftarrow_{\rho_1, o_1} K \Rightarrow_{\rho_2, o_2} P_2$ with induced ACs ac_K and ac^*_K on K, if there exists an injective morphism $m \colon K \to G$ such that $m \models \mathrm{ac}_K \wedge \mathrm{ac}^*_K$ and $m_i = m \circ o_i$, for $i = 1, 2$, satisfy the gluing conditions.

Note that this new critical pair notion is different from the one for rules with NACs [7]. For example, here critical pairs are AC-disregarding. One may wonder if, in the case where all the ACs are NACs, our current critical pairs coincide with the notion defined in [7]. The answer is no, although they are in some sense equivalent. Each critical pair with NACs, according to the previous notion, corresponds to a critical pair with NACs, according to the new notion. In [7] produce-forbid critical pairs, in addition to an overlap of the left-hand sides of the rules, may also contain a part of the corresponding NACs. In the new notion, this additional part would be included in ac^*_K.

As said above it can be proven that critical pairs are complete. Moreover, the converse property also holds, in the sense that every critical pair can be extended to a pair of parallel dependent rule applications (see Lemma 5 in [10]).

Theorem 1 (completeness of critical pairs with ACs [10]). For each pair of parallel dependent direct transformations $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ with ACs, there is a critical pair $P_1 \Leftarrow_{\rho_1,o_1} K \Rightarrow_{\rho_2,o_2} P_2$ with induced ac_K and ac^*_K and an "extension diagram" with an injective morphism m and $m \models \mathrm{ac}_K \wedge \mathrm{ac}^*_K$.

$$\begin{array}{ccccc} P_1 & \xleftarrow{\rho_1,\, o_1} & K & \xrightarrow{\rho_2,\, o_2} & P_2 \\ \downarrow & & \downarrow{\scriptstyle m} & & \downarrow \\ H_1 & \xleftarrow[\rho_1,\, m_1]{} & G & \xrightarrow[\rho_2,\, m_2]{} & H_2 \end{array}$$

Moreover, for each critical pair $P_1 \Leftarrow_{\rho_1,o_1} K \Rightarrow_{\rho_2,o_2} P_2$ for (ρ_1, ρ_2) there is a parallel dependent pair $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ and injective morphism $m\colon K \to G$ such that $m \models \mathrm{ac}_K \wedge \mathrm{ac}^*_K$ leading to the above extension diagram.

Example 5 (critical pair). Because of Theorem 1, the weak critical pair described in Example 4 is a critical pair. In particular, the parallel dependent transformations depicted in Fig. 3 satisfy ac_K and ac^*_K.

Example 6 (critical pair with non-trivial ac^*_K). As noticed already in Example 4, the conflict-inducing condition ac^*_K is equivalent to true for each monomorphic extension morphism of the critical pair, leading to conflicting transformations for each extension of the critical pair. Now we illustrate by a toy example that, in general, ac^*_K may also be of a non-trivial kind. Consider the rule $r_1 : \bullet \leftarrow \bullet \rightarrow \bullet\bullet$, producing a node, and the rule $r_2 : \bullet \leftarrow \bullet \rightarrow \bullet\bullet$ with NAC $\nexists\mathrm{neg} : \bullet \rightarrow \bullet\bullet\bullet$, producing a node only if two other nodes do not exist already. In this case, $\mathrm{ac}_K = \nexists\mathrm{neg} : \bullet \rightarrow \bullet\bullet\bullet$ and $\mathrm{ac}^*_K = (\exists\mathrm{pos}_1 : \bullet \rightarrow \bullet\bullet) \vee (\exists\mathrm{pos}_2 : \bullet \rightarrow \bullet\bullet\bullet)$ with $K = \bullet$. The extension condition ac_K expresses that each extension morphism $m : K \to G$ leads to a pair of valid direct transformations via r_1 and r_2, whenever G does not contain two additional nodes to the one in K. The conflict-inducing condition ac^*_K expresses that each extension morphism $m : K \to G$ into a graph G leads to a pair of conflicting transformations via r_1 and r_2, whenever G contains one additional node to the one in K. The graph $G = \bullet\bullet$, holding one additional node to K, demonstrates that the weak critical pair $P_1 \Leftarrow_{r_1} K \Rightarrow_{r_2} P_2$ is indeed a critical pair.

5 Local Confluence

In this section, we present the main result of this paper, the Local Confluence Theorem for rules with ACs, based on our new critical pair notion. Roughly speaking, in order to show local confluence, we have to require that all critical pairs are confluent. However, as we have seen in Section 2, even in the case of plain graph transformation rules this is not sufficient to show local confluence [4,5]. Suppose that we have two one-step transformations $H_1 \Leftarrow G \Rightarrow H_2$. By completeness of critical pairs, there is a critical pair $P_1 \Leftarrow K \Rightarrow P_2$, with associated conditions ac_K and ac^*_K, which is embedded in $H_1 \Leftarrow G \Rightarrow H_2$ via an extension $m : K \to G$ satisfying ac_K and ac^*_K. If critical pairs are plain strictly confluent, we have transformations $P_1 \overset{*}{\Rightarrow} K' \overset{*}{\Leftarrow} P_2$, which can be embedded into transformations $H_1 \overset{*}{\Rightarrow} H \overset{*}{\Leftarrow} H_2$, if we disregard the ACs. However, if we consider the ACs, we may be unable to apply some rule in these derivations if the corresponding match fails to satisfy the ACs of that rule. Now, for every AC-disregarding transformation $\bar{t} : K \overset{*}{\Rightarrow} K'$ we may compute, as described below, a condition $\mathrm{ac}(\bar{t})$ which is equivalent to all the AC's in the transformation, in the sense that for every extension $m : K \to G$ we have that m satisfies $\mathrm{ac}(\bar{t})$ if all the match morphisms in the extended transformation $G \overset{*}{\Rightarrow} G'$ satisfy the ACs of the corresponding rule. Then, if we can prove *AC-compatibility*, i.e. that ac_K and ac^*_K imply $\mathrm{ac}(\bar{t})$ and $\mathrm{ac}(\overline{t'})$, where $\bar{t} : K \Rightarrow P_1 \overset{*}{\Rightarrow} K' \overset{*}{\Leftarrow} P_2 \Leftarrow K : \overline{t'}$, we would know that all the rule applications in the transformations $H_1 \overset{*}{\Rightarrow} H \overset{*}{\Leftarrow} H_2$ satisfy the corresponding ACs, since $m : K \to G$ satisfies ac_K and ac^*_K. AC-compatibility plus plain strict confluence is called *strict AC-confluence*.

Let us now describe how we can compute $\mathrm{ac}(t)$ for a transformation $P_1 \Rightarrow_{\rho_1,m_1} P_2 \Rightarrow_{\rho_2,m_2} \cdots \Rightarrow_{\rho_n,m_n} P_n$. First, we know that using the shift construction in Lemma 1, we may translate the AC, ac_{L_i}, on the rule ρ_i into an equivalent condition $\mathrm{Shift}(m_i, \mathrm{ac}_{L_i})$ on P_i. On the other hand, we also know that, applying repeatedly the construction in Lemma 2, we can transform every condition $\mathrm{Shift}(m_i, \mathrm{ac}_{L_i})$ on P_i into an equivalent condition ac'_i on P_1. Then, $\mathrm{ac}(t)$ is just the conjunction of all these conditions, i.e. $\mathrm{ac}(t) = \bigwedge_i \mathrm{ac}'_i$.

Theorem 2 (Local Confluence with ACs [10]). A transformation system with ACs is locally confluent, if all critical pairs are strictly AC-confluent, i.e. AC-compatible and plain strict confluent.

Remark 1. As proven in [8], in the case of graphs, nested application conditions are expressively equivalent to first order graph formulas. This means that the satisfiability problem for ACs is undecidable and, as a consequence, constructing the set of critical pairs for a transformation system with arbitrary ACs would be a non-computable problem. Similarly, showing logical consequence, and in particular AC-compatibility, and therefore showing strict confluence, is also undecidable. However in [16,17,18], techniques are presented to tackle the satisfiability and the deduction problems in pratice. Obviously, this kind of techniques would be important in this context. Nevertheless, it must be taken into

account that, as shown in [5], checking local confluence for terminating graph transformation systems is undecidable, even in the case of rules without ACs.

Example 7 (Local Confluence with ACs). In order to apply Theorem 2 we show that the critical pair in Example 4 and 5 is strictly AC-confluent. First of all, rule process_stop_down and then rule move_down can be applied to P_2 leading to $K' = P_1$ (see Fig. 5, where $G' = H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ is shown explicitly in Fig. 3). This is a strict solution, since it deletes no floor, nor the elevator, the only structures which are preserved by the critical pair. Moreover, we can see that this solution is AC-compatible. Let $\bar{t}_1 : K \Rightarrow P_1 = K'$ and $\bar{t}_2 : K \Rightarrow P_2 \Rightarrow P_3 \Rightarrow K'$. At first, we can conclude that $\mathrm{ac}_K \Rightarrow \mathrm{ac}(\bar{t}_2)$ (see ac_K in Fig. 4), because $\mathrm{ac}(\bar{t}_2)$ is, in particular, equivalent to ac_K. Therefore, also $\mathrm{ac}_K \wedge \mathrm{ac}_K^* \Rightarrow \mathrm{ac}(\bar{t}_1) \wedge \mathrm{ac}(\bar{t}_2)$, since $\mathrm{ac}(\bar{t}_1) = \mathrm{Shift}(o_1, \mathrm{ac}_{L_1})$, where $\mathrm{ac}_K = \mathrm{Shift}(o_1, \mathrm{ac}_{L_1}) \wedge \mathrm{Shift}(o_2, \mathrm{ac}_{L_2})$, and as mentioned in Example 4 ac_K^* is equivalent to true. By Theorem 2 we can conclude that the pair $G' = H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ is locally confluent as shown in the outer diagram of Fig. 5. This means that the elevator in downward mode in the four-storey building with a request on the lowest floor can first process the generated stop request and then continue moving downward instead of moving downward immediately.

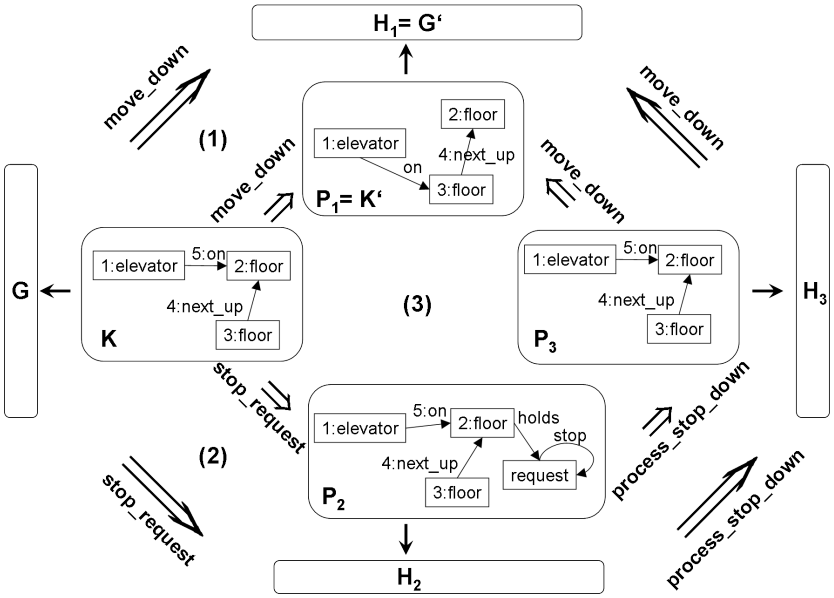


Fig. 5. strictly AC-confluent critical pair of *Elevator*

6 Conclusion, Related and Future Work

In this paper we have presented for the first time a new method for checking local confluence for graph transformation systems with ACs. This kind of ACs provide a considerable increase of expressive power with respect to the NACs that were used up to now. Moreover, as said in the introduction, all the results presented apply not only to the categories of graphs or typed graphs, but to arbitrary adhesive categories and, more generally, to (weak) adhesive high-level replacement (HLR) categories with some additional properties. An example describing the specification of an elevator system shows that this increase of descriptive power is really necessary. The new method is not just a generalization of the critical pair method introduced in [19,7] to prove local confluence of GTS with NACs. In those papers the critical pairs are defined using graph transformations that satisfy the given NACs. This cannot be done when dealing with nested conditions, because they may include positive conditions which may not be satisfied by the critical pair but only by the embedding of these transformations into a larger context. As a consequence, a new kind of critical pairs had to be defined. As main results we have shown a Completeness and Local Confluence Theorem for the case with ACs.

Graph conditions with a similar expressive power as nested conditions were introduced in [20], while nested conditions were introduced in [8], where we can find a detailed account of many results on them, including their expressive power.

The use of critical pairs to check confluence in GTS was introduced in [4]. On the other hand, graph transformation with the important special kind of negative application conditions was introduced in [6], where it was shown how to transform right application conditions into left application conditions and graph constraints into application conditions. These results were generalized to arbitrary adhesive HLR categories [14] and a critical pair method to study the local confluence of a GTS, and in general of an adhesive rewriting system, with this kind of NACs was presented in [19,7].

Since the construction of critical pairs with NACs is implemented already in the AGG system, an extension to the case with nested application conditions is planned. However, before starting this kind of implementation, some aspects of our techniques need some additional refinement. In particular, additional work is needed on suitable implementations of satisfiability (resp. implication) solvers [16,17] for nested application conditions in order to prove AC-compatibility.

References

1. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
2. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs of Theoretical Computer Science. Springer (2006)

3. Knuth, N.E., Bendix, P.B.: Simple word problems in universal algebra. In J. Leech, editor, *Computational Problems in Abstract Algebra* (1970) 263–297
4. Plump, D.: Hypergraph rewriting: Critical pairs and undecidability of confluence. In: Term Graph Rewriting: Theory and Practice. John Wiley (1993) 201–213
5. Plump, D.: Confluence of graph transformation revisited. In: Processes, Terms and Cycles: Steps on the Road to Infinity: Essays Dedicated to Jan Willem Klop on the Occasion of His 60th Birthday. Volume 3838 of LNCS., Springer (2005) 280–308
6. Habel, A., Heckel, R., Taentzer, G.: Graph grammars with negative application conditions. Fundamenta Informaticae **26** (1996) 287–313
7. Lambers, L., Ehrig, H., Prange, U., Orejas, F.: Embedding and confluence of graph transformations with negative application conditions. In: Graph Transformations (ICGT 2008). Volume 5214 of LNCS., Springer (2008) 162–177
8. Habel, A., Pennemann, K.H.: Correctness of high-level transformation systems relative to nested conditions. Mathematical Structures in Computer Science **19** (2009) 245–296
9. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In: Handbook of Graph Grammars and Computing by Graph Transformation. Volume 1. World Scientific (1997) 313–400
10. Lambers, L., Ehrig, H., Habel, A., Orejas, F., Golas, U.: Local Confluence for Rules with Nested Application Conditions based on a New Critical Pair Notion. Technical Report 2010-7, Technische Universität Berlin (2010)
11. Lack, S., Sobociński, P.: Adhesive categories. In: Foundations of Software Science and Computation Structures (FOSSACS'04). Volume 2987 of LNCS., Springer (2004) 273–288
12. Newman, M.H.A.: On theories with a combinatorial definition of "equivalence". Annals of Mathematics (43,2) (1942) 223–243
13. Koch, M., Mancini, L.V., Parisi-Presicce, F.: Graph-based specification of access control policies. Journal of Computer and System Sciences **71** (2005) 1–33
14. Ehrig, H., Ehrig, K., Habel, A., Pennemann, K.H.: Theory of constraints and application conditions: From graphs to high-level structures. Fundamenta Informaticae **74(1)** (2006) 135–166
15. Ehrig, H., Habel, A., Lambers, L.: Parallelism and concurrency theorems for rules with nested application conditions. In: Essays Dedicated to Hans-Jörg Kreowski on the Occasion of His 60th Birthday. Volume 26 of Electronic Communications of the EASST. (2010)
16. Orejas, F., Ehrig, H., Prange, U.: A logic of graph constraints. In: Fundamental Approaches to Software Engineering (FASE 2008). Volume 4961 of LNCS., Springer (2008) 179–198
17. Pennemann, K.H.: Resolution-like theorem proving for high-level conditions. In: Graph Transformations (ICGT'08). Volume 5214 of LNCS., Springer-Verlag (2008) 289–304
18. Pennemann, K.H.: An algorithm for approximating the satisfiability problem of high-level conditions. In: Proc. Int. Workshop on Graph Transformation for Verification and Concurrency (GT-VC'07). Volume 213 of ENTCS. (2008) 75–94 Long version: http://formale-sprachen.informatik.uni-oldenburg.de/pub/index.html.
19. Lambers, L., Ehrig, H., Orejas, F.: Conflict detection for graph transformation with negative application conditions. In: Graph Transformations (ICGT 2006). Volume 4178 of LNCS., Springer (2006) 61–76
20. Rensink, A.: Representing first-order logic by graphs. In: Graph Transformations (ICGT'04). Volume 3256 of LNCS., Springer (2004) 319–335