Technische Universität Berlin

Forschungsberichte
der Fakultät IV – Elektrotechnik und Informatik

Local Confluence for Rules with Nested Application Conditions based on a New Critical Pair Notion

Leen Lambers, Hartmut Ehrig, Annegret Habel,
Fernando Orejas, and Ulrike Golas

Bericht-Nr. 2010/7
ISSN 1436-9915

Local Confluence for Rules with Nested Application Conditions based on a New Critical Pair Notion

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

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

Abstract. Local confluence is an important property in many rewriting 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 — that are known to be 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 allow that critical pairs do not satisfy the application conditions. This leads to a new notion of critical pairs which allows to formulate and prove a Local Confluence Theorem for rules with nested application conditions in the framework of adhesive rewriting systems based on the DPO-approach. It builds on a new Embedding Theorem and Completeness Theorem for critical pairs based on 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.

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

(GTS) 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, in general, undecidable. A special case is local confluence which, in the case where the rewriting system is terminating, coincides with 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 (TRS) [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 GTS (see, e.g. [4,5,2]). Unfortunately, the results for TRS cannot be adapted straightforwardly to the case of GTS. In general, joinability or confluence of critical pairs does not ensure the local confluence of a GTS. Instead, a stronger notion of *strict confluence of a critical pair*, had to be defined as a sufficient condition for having local confluence. This condition is semi-decidable (resp. decidable for a terminating GTS), because each critical pair can be computed statically and moreover, if there exists some strict solution for the critical pair, then it can be found at some point. For a terminating GTS, the complete state space starting from the critical pair is computable and can be scanned for strict solutions. In this way, although showing confluence for GTS is undecidable [4] in general, at least for showing local confluence there exists a semi-decidable sufficient condition (resp. decidable condition for terminating GTS).

In standard rewriting systems, whenever we find a valid match of the left-hand side of a rule into a given object, that rule can be applied to that object. However, in many occasions, we want to limit the applicability of rules. This leads to the notion of conditional rewriting system, where a rule can only be applied to a given object if some conditions, which are part of the rule, are satisfied. In the case of conditional TRS, conditions are (logical) equalities that must be proved before the given rule can be applied. In this case, critical pair techniques to check local confluence can be extended to cover this case (see, e.g. [6]). In GTS, conditions to restrict the applicability of rules are very different. In this case, *application conditions (AC)* constrain the context of the match of the rule. For instance, an important case of an application condition is a *negative application condition (NAC)*, as introduced in [7], 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 spoken — N is not present in G. Unfortunately, the use of this kind of application conditions poses additional problems for constructing critical pairs and checking local confluence. Actually, the first results on checking local confluence for GTS with ACs are quite recent and are restricted to the case of these NACs [8]. Although NACs are quite useful already, they have limited expressive power: we cannot express forbidden contexts which are more complex than just 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 [9] 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 [10]. Moreover, in this work conditions are not defined for any specific kind of graph, but for objects of any adhesive category [11]. This means that the results apply to a large class of graphical structures. Following that work, in this paper, we study the local confluence of rewriting systems over adhesive categories, where the rules may include arbitrary nested application conditions. Dealing with this general kind of conditions poses new difficulties with respect to previous results. In particular, due to the fact that a nested condition may include combinations of positive and negative conditions, we defined a new notion of critical pairs in order to obtain a corresponding Local Confluence Theorem for the nested case.

The paper is organized as follows: In Section 2, we review the notions of graphs and high-level structures in the context of adhesive categories. In Section 3, we introduce our running example on an elevator control using nested application conditions (ACs) and introduce the concepts of nested application conditions and rules. In Section 4, we proof the Embedding and Extension Theorems for rules with ACs, allowing to extend a transformation to a larger context. In Section 5, first we review the notion of parallel dependency for transformations via rules with ACs. Then, we define our new notion of critical pairs and state a completeness theorem (Theorem 3) saying that every pair of dependent direct derivations is an extension of a critical pair. In Section 6, we state as main result the Critical Pair lemma or Local Confluence Theorem (Theorem 4) for rules with ACs. The concepts are illustrated by the rule-based modeling of an elevator control with means of a typed graph transformation system. A conclusion including related work is given in Section 7.

2 Graphs and High-Level Structures

In this section, we review the definitions of typed graphs and graph morphisms as well as adhesive categories and some special properties.

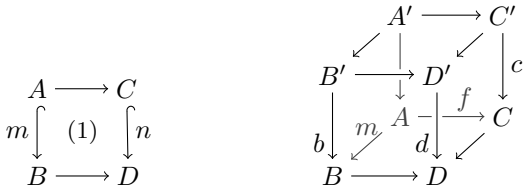
Definition 1 ((typed) graphs and graph morphisms). A *graph* $G = (G_V, G_E, s, t)$ consists of a set G_V of vertices, a set G_E of edges and two mappings $s, t : G_E \to G_V$, assigning to each edge $e \in G_E$ a source $s(e) \in G_V$ and target $t(e) \in G_V$. A *graph morphism* $f : G_1 \to G_2$ between two graphs $G_i = (G_{i,V}, G_{i,E}, s_i, t_i)$, $(i = 1, 2)$ is a pair $f = (f_V : G_{V,1} \to G_{V,2}, f_E : G_{E,1} \to G_{E,2})$ of mappings, such that $f_V \circ s_1 = s_2 \circ f_E$ and $f_V \circ t_1 = t_2 \circ f_E$. The category having graphs as objects and graph morphisms as arrows is called **Graphs**. A *type graph* is a distinguished graph $TG = (V_{TG}, E_{TG}, s_{TG}, t_{TG})$. V_{TG} and E_{TG} are called the node and the edge type alphabets, respectively. A tuple $(G, type)$

of a graph G together with a graph morphism $type : G \to TG$ is then called a *typed graph*. Consider typed graphs $G_1^T = (G_1, type_1)$ and $G_2^T = (G_2, type_2)$, a *typed graph morphism* $f : G_1^T \to G_2^T$ is a graph morphisms $f : G_1 \to G_2$ such that $type_2 \circ f = type_1$. The category having typed graphs as objects and typed graph morphisms as arrows is called $\mathbf{Graphs}_{TG}$.

The considerations in this paper are based on adhesive categories together with some specific extra properties. Note that the theory introduced in this paper can be generalized also to (weak) adhesive high-level replacement (HLR) categories, describing more structures which are of interest in computer science and mathematics like Petri nets, (hyper)graphs, and algebraic specifications. The idea behind the consideration of (weak) adhesive (HLR) categories is to avoid similar investigations for different instantiations. Readers interested in the category-theoretic background of these concepts may consult e.g. [2].

Definition 2 (adhesive category [11]). A category $\mathcal{C}$ is an *adhesive category*, if the following properties hold:

1. $\mathcal{C}$ has pushouts along monomorphisms (i.e. pushouts where at least one of the given morphisms is a monomorphism).
2. $\mathcal{C}$ has pullbacks.
3. Pushouts in $\mathcal{C}$ along monomorphisms are VK-squares, i.e. for any commutative cube in $\mathcal{C}$ where we have a pushout in the bottom and the back faces are pullbacks, it holds: the top is pushout iff the front faces are pullbacks.



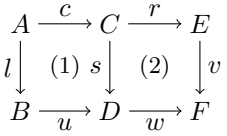
Fact 1 ($\mathbf{Graphs}_{TG}$ is adhesive [2]). The category $\mathbf{Graphs}_{TG}$ of graphs typed over a type graph TG is an adhesive category.

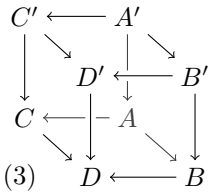
Adhesive categories have a number of nice properties, called HLR properties [12].

Fact 2 (properties of adhesive categories [11,2]). For an adhesive category $\mathcal{C}$, the following properties hold:

1. Pushouts along monomorphisms are pullbacks.
2. Pushout-pullback decomposition. If the diagram (1)+(2) is a pushout, (2) a pullback, w a monomorphism and (l or c a monomorphism), then (1) and (2) are pushouts and also pullbacks.

3. Cube pushout-pullback decomposition. Given the commutative cube (3) below, where all morphisms in the top and the bottom are monomorphisms, the top is pullback and the front faces are pushouts, then the bottom is a pullback iff the back faces of the cube are pushouts.

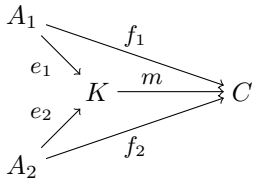




(3)

4. Uniqueness of pushout complements. Given a monomorphism $c\colon A \to C$ and morphism $s\colon C \to D$, then there is, up to isomorphism, at most one B with $l\colon A \to B$ and $u\colon B \to D$ such that diagram (1) is a pushout.

Definition 3 (unique $\mathcal{E}$-$\mathcal{M}$ pair factorization). Given a class of morphism pairs $\mathcal{E}$ with the same codomain and a class $\mathcal{M}$ of monomorphisms, an adhesive category has a unique $\mathcal{E}$-$\mathcal{M}$ *pair factorization* if, for each pair of morphisms $f_1\colon A_1 \to C$ and $f_2\colon A_2 \to C$, there exist a unique (up to isomorphism) object K and unique (up to isomorphism) morphisms $e_1\colon A_1 \to K$, $e_2\colon A_2 \to K$, and $m\colon K \to C$ with $(e_1, e_2) \in \mathcal{E}$ and $m \in \mathcal{M}$ such that $m \circ e_1 = f_1$ and $m \circ e_2 = f_2$:



3 Rules with Nested Application Conditions

In this section, we reintroduce basic concepts with respect to rules with nested application conditions. Moreover, we introduce our running example on the modeling of an elevator control by typed graphs and typed graph transformation rules. The example makes use of rules with nested application conditions.

Example 1 (*Elevator*). The elevator control is modelled by a typed graph and typed graph rules. 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 occurs, for example, in apartment buildings or multi-storey car parks [13]. 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 *up*ward or *down*ward 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.

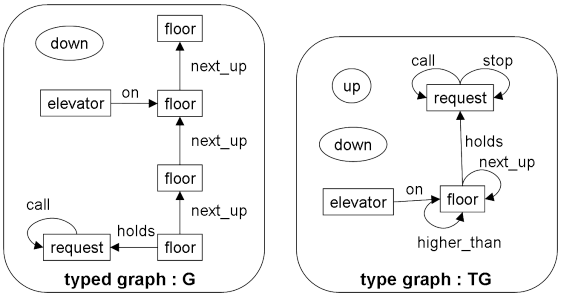


Fig. 1. A typed graph of *Elevator* together with its type graph

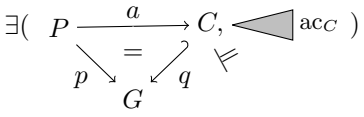
In the following, we consider adhesive categories [11,2] with some specific extra properties. The basic definitions and some specific extra properties are collected in the appendix. The reader unfamiliar with adhesive categories can safely take the standard category of typed graphs as considered in our running example.

General Assumption. In the following, we assume that $\mathcal{C}$ is an adhesive category with an epi-$\mathcal{M}$-factorization (used in Lemma 1) and a unique $\mathcal{E}$-$\mathcal{M}$ pair factorization (used in Thm 3), where $\mathcal{M}$ is the class of all monomorphisms and with initial pushouts over $\mathcal{M}$-morphisms (used in Thm 4).

The category $\mathbf{Graphs}_{TG}$ of graphs and morphisms typed over TG is adhesive and holds the properties in the assumption [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.

Application conditions are defined as in [14,9]. Syntactically, application conditions may be seen as a tree of morphisms equipped with certain logical symbols such as quantifiers and connectives.

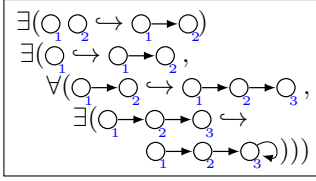
Definition 4 (nested application conditions). A *(nested) application condition*, short *condition* or AC, ac_P over an object P is of the form true or $\exists(a, \mathrm{ac}_C)$, where $a\colon P \to C$ is a morphism and ac_C is an application condition over C. Moreover, Boolean formulas over application conditions over P are application conditions over P. $\exists a$ abbreviates $\exists(a, \mathrm{true})$, $\forall(a, \mathrm{ac}_C)$ abbreviates $\neg\exists(a, \neg\mathrm{ac}_C)$. Every morphism *satisfies* true. A morphism $p\colon P \to G$ *satisfies* an application condition $\exists(a, \mathrm{ac}_C)$ if there exists a morphism q in $\mathcal{M}$ such that $q \circ a = p$ and $q \models \mathrm{ac}_C$.



The satisfaction of application conditions over P by morphisms with domain P is extended to Boolean formulas over application conditions in the usual way. We write $p \models \mathrm{ac}_P$ to denote that the morphism p satisfies ac_P. A condition ac_L *implicates* ac'_L, written $\mathrm{ac}_L \Rightarrow \mathrm{ac}'_L$, if for all morphisms $m\colon L \to G$ with $m \models \mathrm{ac}_L$ it follows that $m \models \mathrm{ac}'_L$. Two conditions ac_L and ac'_L over L are *equivalent*, denoted by $\mathrm{ac}_L \equiv \mathrm{ac}'_L$, if for all morphisms $m\colon L \to G$, $m \models \mathrm{ac}_L$ iff $m \models \mathrm{ac}'_L$.

Remark 1. The definition of application conditions generalizes those in [7,15,16]. Negative (positive) application conditions [7,8], short NACs (PACs), correspond to nested application conditions of the form $\nexists a$ ($\exists a$).

Example 2. Examples of ACs are given below, where the first one is of standard type considered already in [7], while the second one is properly nested.



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.

Application conditions can be shifted over morphisms into corresponding ACs over the codomain of the morphism.

Lemma 1 (shift of ACs over morphisms [17]). There is a transformation Shift such that, for all application conditions ac_P over P and all morphisms $b\colon P \to P'$, $n\colon P' \to H$, we have $n \circ b \models \mathrm{ac}_P \Leftrightarrow n \models \mathrm{Shift}(b, \mathrm{ac}_P)$.

Rules are defined as in [16,9]. They are specified by a span of monomorphisms with an application condition on the left-hand side of the rule[5]. We consider the classical semantics based on the double-pushout (DPO) construction [2,18]. Thereby, we distinguish between transformations via rules with ACs, where application conditions may have to be fulfilled or may also be disregarded.

Definition 5 (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 $I \hookrightarrow R$ monomorphisms and an application condition ac_L over L. L and R are called the left- and the right-hand side (LHS and RHS) of p and I the interface; ac_L is the application condition of p.

$$\begin{array}{ccccc} \mathrm{ac}_L & & L & \hookleftarrow & I & \hookrightarrow & R \\ & \not\models & \downarrow m & (1) & \downarrow & (2) & \downarrow \\ & & G & \hookleftarrow & D & \hookrightarrow & H \end{array}$$

A *direct transformation* consists of two pushouts (1) and (2), called DPO, such that $m \models \mathrm{ac}_L$. We write $G \Rightarrow_{\rho,m,m^*} H$ and say that $m\colon L \to G$ is the match of ρ in G and $m^*\colon R \to H$ is the comatch of ρ in H. We also write $G \Rightarrow_{\rho,m} H$ or $G \Rightarrow_{\rho} H$ to express that there is an m^* or there are m and m^*, respectively, such that $G \Rightarrow_{\rho,m,m^*} H$. A *plain direct transformation* via a plain rule p consists of DPO (1) and (2). We write $G \Rightarrow_{p,m,m^*} H$. An *AC-disregarding direct transformation* $G \Rightarrow_{\rho,m,m^*} H$ consists of DPO (1) and (2), where m (resp. m^*) does not necessarily need to fulfill ac_L (resp. ac_R). A *sequence* of (plain resp. AC-disregarding) direct transformations $G_0 \Rightarrow G_1 \Rightarrow G_2 \cdots G_{n-1} \Rightarrow G_n$ is called a (plain resp. AC-disregarding) transformation and is denoted by $G_0 \Rightarrow^* G_n$. For $n = 0$, we have the identical transformation $G_0 \Rightarrow^0 G_0'$ for $G_0 \cong G_0'$, because PO's and hence also transformations are only unique up to isomorphism.

Example 3 (typed graph rules of *Elevator*). Exemplarily, we show three rules modeling part of the elevator control as given in Example 1. First, we have rule *move_down* in Fig. 2 with combined AC on L, consisting of three PACs ($pos_i : L \to P_i, i = 1, \ldots, 3$) and a NAC ($neg : L \to N$). This rule describes that the elevator car moves down one floor under the condition that the elevator car is in downward mode ($\exists pos_3$), that no request is present on the elevator floor ($\nexists neg$), and that some request is present on the next lower floor ($\exists pos_2$) or some other lower floor ($\exists pos_1$). Note that only L and R of the rules are depicted. 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. As a second rule, we

[5] Note that in [16,9] also application conditions on the right-hand side of a rule are allowed, but because of Lemma 2 this case can be reduced to rules with left application conditions only.

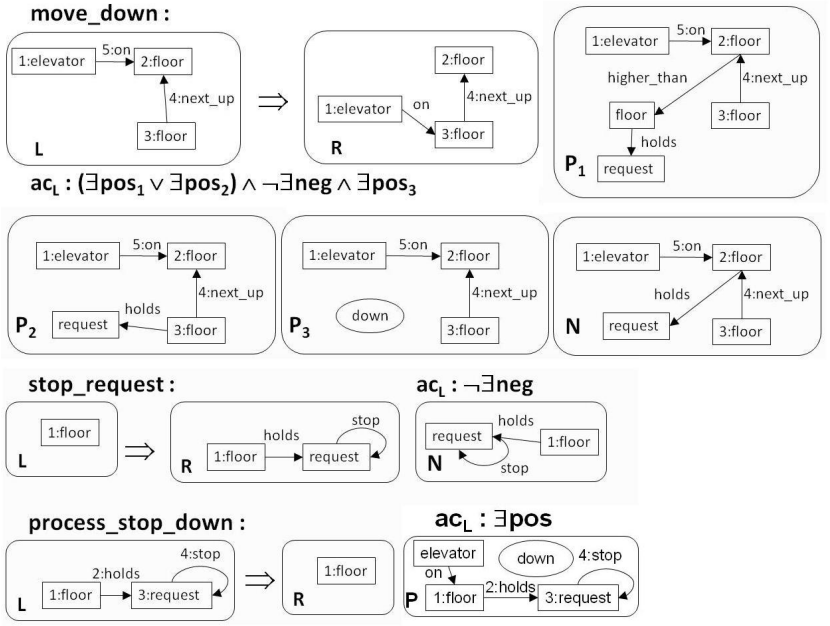


Fig. 2. Rules for *Elevator*

consider rule *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 of a rule can be shifted from right to left and vice versa.

Lemma 2 (shift of ACs over rules [9]). There are transformations L and R such that, for every application condition ac_R on R and every application condition ac_L on L of a rule ρ and every direct transformation $G \Rightarrow_{\rho,m,m^*} H$, we have $m \models \mathrm{L}(\rho, \mathrm{ac}_R) \Leftrightarrow m^* \models \mathrm{ac}_R$ and $m \models \mathrm{ac}_L \Leftrightarrow m^* \models \mathrm{R}(\rho, \mathrm{ac}_L)$.

4 Embedding and Extension Theorems

In this section, we present Embedding and Extension Theorems, which allow us to extend a transformation to a larger context.

An extension diagram describes how a transformation $t\colon G_0 \Rightarrow^* G_n$ can be extended to a transformation $t'\colon G'_0 \Rightarrow^* G'_n$ via the same rules and an extension

morphism $k_0 \colon G_0 \to G'_0$ that maps G_0 to G'_0 as shown in the following diagram.

$$\begin{array}{ccc} G_0 & \overset{*}{\Longrightarrow} & G_n \\ {\scriptstyle k_0}\downarrow & (1) & \downarrow{\scriptstyle k_n} \\ G'_0 & \overset{*}{\Longrightarrow} & G'_n \end{array}$$

We first introduce the notion of a derived application condition (resp. derived rule) of a transformation $t : G_0 \Rightarrow^* G_n$, which is needed for the notion of consistency of an extension morphism as given in Def. 7 as well as for AC-compatibility in Def. 11 in Section 6.

Definition 6 (derived application condition, derived rule). Given an (AC-disregarding) transformation $t \colon G_0 \Rightarrow^* G_n$, the *derived application condition* $\mathrm{ac}(t)$ over G_0 is defined inductively as follows: For t of length 0 with $G_0 \cong G'_0$, let $\mathrm{ac}(t) = \mathrm{true}$. For $t \colon G_0 \Rightarrow_{\rho_1, m_1} G_1$, let $\mathrm{ac}(t) = \mathrm{Shift}(m_1, \mathrm{ac}_{L_1})$. For $t \colon G_0 \Rightarrow^* G_{n-1} \Rightarrow G_n$ with $n \geq 2$, let $\mathrm{ac}(t) = \mathrm{ac}(G_0 \Rightarrow^* G_{n-1}) \wedge \mathrm{L}(p^*, \mathrm{ac}(G_{n-1} \Rightarrow G_n))$, where $p^* = \langle G_0 \hookleftarrow D \hookrightarrow G_{n-1} \rangle$ is the plain derived rule [2] of $G_0 \Rightarrow^* G_{n-1}$ defined by the DPO for $n = 2$ and by iterated pullback construction for $n > 2$. The *derived rule* $\rho(t) = \langle p(t), \mathrm{ac}(t) \rangle$ of an (AC-disregarding) transformation $t \colon G_0 \Rightarrow^* G_n$ consists of the plain derived rule $p(t)$ and the derived application condition $\mathrm{ac}(t)$ of t.

$$\begin{array}{ccccccccc} & & & & D & & & & \\ & & & \swarrow & = & \searrow & & & \\ G_0 & \hookleftarrow & D' & \hookrightarrow & \text{(PB)} & \hookleftarrow & D_n & \hookrightarrow & G_n \end{array}$$

Definition 7 (AC-consistency, consistency). Given an (AC-disregarding) transformation $t \colon G_0 \Rightarrow^* G_n$ with derived rule $\rho(t) = \langle p(t), \mathrm{ac}(t) \rangle$ and a morphism $k_0 \colon G_0 \to G'_0$. k_0 is *boundary consistent* with respect to t, if there exist an initial pushout (1) over k_0 and a morphism $b \in \mathcal{M}$ with $d_0 \circ b = b_0$. k_0 is *AC-consistent* with respect to t, if $k_0 \models \mathrm{ac}(t)$. k_0 is *consistent* with respect to t, if k_0 is boundary and AC-consistent with respect to t.

$$\begin{array}{ccccccc} \mathrm{ac}(t) \triangleright & G_0 & \overset{d_0}{\hookleftarrow} & D & \overset{d_n}{\hookrightarrow} & G_n & \\ & {\scriptstyle k_0}\downarrow & \nwarrow^{b_0} & \nearrow_{b} & & & \\ & G'_0 & & B & & & \\ & & \searrow & \downarrow & & & \\ & & & C & & & \end{array}$$

Theorem 1 (Embedding Theorem with ACs). Given an (AC-disregarding) transformation $t \colon G_0 \Rightarrow^* G_n$ and an extension morphism $k_0 \colon G_0 \to G'_0$ consistent with respect to t, then there is an extension diagram (1) over t and k_0 as shown

below, where $t'\colon G'_0 \Rightarrow^* G'_n$ is a transformation via the same rules satisfying the corresponding ACs.

$$\begin{array}{ccc} G_0 & \overset{*}{\Longrightarrow} & G_n \\ {\scriptstyle k_0}\downarrow & (1) & \downarrow{\scriptstyle k_n} \\ G'_0 & \overset{*}{\Longrightarrow} & G'_n \end{array}$$

Proof. Let $t\colon G_0 \Rightarrow^* G_n$ be a (AC-disregarding) transformation via rules with application conditions and $k_0\colon G_0 \to G'_0$ be consistent with respect to t. Then the underlying transformation t_0 via rules without application conditions is boundary consistent with respect to t_0 and, by the Embedding Theorem for rules without application conditions [2], there is a plain extension diagram t'_0 over t_0 and k_0. Moreover, by the Extension Theorem for rules without application conditions [2] the following DPO diagram exists:

$$\begin{array}{ccccc} G_0 & \longrightarrow & D & \longrightarrow & G_n \\ {\scriptstyle k_0}\downarrow & (1) & \downarrow & (2) & \downarrow{\scriptstyle k_n} \\ G'_0 & \longrightarrow & D' & \longrightarrow & G'_n \end{array}$$

By assumption, $k_0 \models \mathrm{ac}(t)$. It remains to show that the transformation via rules with application conditions is an extension diagram, i.e., $k_{i-1} \circ m_i \models \mathrm{ac}_{L_i}$ for $i = 1, \ldots, n$. This is proved by induction over the number of direct transformation steps n.

Basis. For a transformation $t\colon G_0 \Rightarrow^0 G'_0$ of length 0, $k_0 \models \mathrm{ac}(t) = \mathrm{true}$. For a transformation $t\colon G_0 \Rightarrow_{\rho_1,m_1} G_1$ of length 1, $k_0 \models \mathrm{ac}(t) = \mathrm{Shift}(m_1, \mathrm{ac}_{L_1}) \Leftrightarrow k_0 \circ m_1 \models \mathrm{ac}_{L_1}$.

Induction hypothesis. For a transformation $t\colon G_0 \Rightarrow^* G_i$ of length $i \geq 1$, $k_0 \models \mathrm{ac}(G_0 \Rightarrow^* G_i) \Leftrightarrow k_{j-1} \circ m_j \models \mathrm{ac}_{L_j}$ for $j = 1, \ldots, i$.

Induction step. Consider now the transformation $t\colon G_0 \Rightarrow^* G_i \Rightarrow G_{i+1}$ and the following DPO diagram:

$$\begin{array}{ccccc} G_0 & \longrightarrow & D & \longrightarrow & G_i \\ {\scriptstyle k_0}\downarrow & (1) & \downarrow & (2) & \downarrow{\scriptstyle k_i} \\ G'_0 & \longrightarrow & D' & \longrightarrow & G'_i \end{array}$$

Then

$$\begin{array}{lll} & k_0 \models \mathrm{ac}(G_0 \Rightarrow^* G_{i+1}) & \\ \Leftrightarrow & k_0 \models \mathrm{ac}(G_0 \Rightarrow^* G_i) \wedge \mathrm{L}(p(G_0 \Rightarrow^* G_i), \mathrm{ac}(G_i \Rightarrow G_{i+1})) & \text{Definition of ac} \\ \Leftrightarrow & k_0 \models \mathrm{ac}(G_0 \Rightarrow^* G_i) \wedge k_i \models \mathrm{ac}(G_i \Rightarrow G_{i+1}) & \text{Shift-Lemma 2} \\ \Leftrightarrow & k_0 \models \mathrm{ac}(G_0 \Rightarrow^* G_i) \wedge k_i \models \mathrm{Shift}(m_{i+1}, \mathrm{ac}_{L_{i+1}}) & \text{Definition of ac} \\ \Leftrightarrow & k_0 \models \mathrm{ac}(G_0 \Rightarrow^* G_i) \wedge k_i \circ m_{i+1} \models \mathrm{ac}_{L_{i+1}} & \text{Shift-Lemma1} \\ \Leftrightarrow & k_{j-1} \circ m_j \models \mathrm{ac}_{L_j} \text{ for } j = 1, \ldots, i+1 & \text{Ind hypothesis} \end{array}$$

□

Remark 2. Note that the Embedding Theorem is valid for t satisfying ACs and for t disregarding ACs.

The consistency condition is also necessary for the construction of extension diagrams, provided that we have initial pushouts over $\mathcal{M}'$-morphisms. Moreover, we are able to give a direct construction of G'_n in the extension diagram (1) below. This avoids the need to give an explicit construction of $t'\colon G'_0 \Rightarrow^* G'_n$.

Theorem 2 (Extension Theorem). Given a (AC-disregarding) transformation $t\colon G_0 \Rightarrow^* G_n$ with derived rule $\rho(t)$ and an extension diagram (1),

$$\begin{array}{ccccc} B & \xrightarrow{b_0} & G_0 & \overset{*}{\Longrightarrow} & G_n \\ \downarrow & (2) & \downarrow{\scriptstyle k_0} & (1) & \downarrow{\scriptstyle k_n} \\ C & \longrightarrow & G'_0 & \overset{*}{\Longrightarrow} & G'_n \end{array}$$

with an initial pushout (2) over $k_0 \in \mathcal{M}'$ for some class $\mathcal{M}'$, closed under pushouts and pullbacks along $\mathcal{M}$-morphisms and with initial pushouts over $\mathcal{M}'$-morphisms, then we have the following, shown in the diagram below:

1. k_0 is consistent with respect to $t\colon G_0 \Rightarrow^* G_n$ with $b\colon B \to D$.
2. There is a direct transformation $t'\colon G'_0 \Rightarrow G'_n$ via $\rho(t)$ and k_0 given by the pushouts (3) and (4) with $k, k_n \in \mathcal{M}'$.
3. There are initial pushouts (5) over $k \in \mathcal{M}'$ and (6) over $k_n \in \mathcal{M}'$, respectively, with same boundary-context morphism $B \to C$.

$$\begin{array}{ccccc} {\scriptstyle \mathrm{ac}(t)}\triangleright\, G_0 & \hookleftarrow & D & \hookrightarrow & G_n \\ {\scriptstyle k_0}\downarrow & (3) & {\scriptstyle k}\downarrow & (4) & \downarrow{\scriptstyle k_n} \\ G'_0 & \hookleftarrow & D' & \hookrightarrow & G'_n \end{array} \qquad \begin{array}{ccccc} D & \hookleftarrow & B & \hookrightarrow & G_n \\ {\scriptstyle k}\downarrow & (5) & \downarrow & (6) & \downarrow{\scriptstyle k_n} \\ D' & \hookleftarrow & C & \hookrightarrow & G'_n \end{array}$$

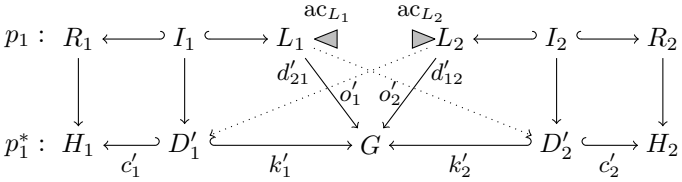
Proof. Let $t\colon G_0 \Rightarrow^* G_n$ be a transformation via rules with application conditions with derived rule $\rho(t)$ and an extension diagram (1), with an initial pushout (2) over $k_0 \in \mathcal{M}'$. By the Extension Theorem for rules without application conditions [2], k_0 is boundary consistent with respect to $t\colon G_0 \Rightarrow^* G_n$, with the morphism $b\colon B \to D$ and properties (2) and (3) are satisfied. It remains to show that k_0 is consistent with respect to $t\colon G_0 \Rightarrow^* G_n$, with the morphism $b\colon B \to D$. By assumption, (1) is an extension diagram, i.e., $t'\colon G'_0 \Rightarrow^* G'_n$ is a transformation via $(\rho_1, \ldots, \rho_n)$, and $k_i \circ m_i \models \mathrm{ac}_{L_i}$ for $i = 1, \ldots, n$. Then $k_0 \models \mathrm{ac}(t)$, i.e. k_0 is AC-consistent and hence consistent with respect to $t\colon G_0 \Rightarrow^* G_n$ with the morphism $b\colon B \to D$. □

5 Critical Pairs and Completeness

We now present the new concept of critical pairs, which will finally lead to the Local Confluence Theorem. First, we recall from [17] that a pair $H_1 \Leftarrow_{\rho_1, m_1}$

$G \Rightarrow_{\rho_2,m_2} H_2$ is called parallel independent if it is plain parallel independent in the sense of [2] and that the induced matches satisfy the application conditions of ρ_1 and ρ_2, respectively. Note that given a parallel dependent (i.e. not parallel independent) pair $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ via (ρ_1, ρ_2) with ACs, the standard construction in [2] for the corresponding critical pair via $(\mathcal{E}, \mathcal{M})$ pair factorization leads to a weak critical pair $P_1 \Leftarrow_{\rho_1,o_1} K \Rightarrow_{\rho_2,o_2} P_2$, which is AC-disregarding and not necessarily plain parallel dependent, but minimal. It will later be used to define a new notion of critical pairs, which is used in the Local Confluence Theorem in Section 6 for the case of nested application conditions.

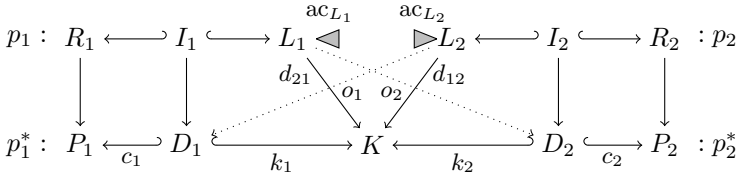
Definition 8 (parallel independence with ACs). A pair of direct transformations $H_1 \Leftarrow_{\rho_1,o_1'} G \Rightarrow_{\rho_2,o_2'} H_2$ with ACs is parallel independent if there exists a morphism $d_{12}' \colon L_1 \to D_2'$ such that $k_2' \circ d_{12}' = o_1'$ and $c_2' \circ d_{12}' \models \mathrm{ac}_{L_1}$ and there exists a morphism $d_{21}' \colon L_2 \to D_1'$ such that $k_1' \circ d_{21}' = o_2'$ and $c_1' \circ d_{21}' \models \mathrm{ac}_{L_2}$.



Definition 9 (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 $(o_1, o_2) \in \mathcal{E}$. Every weak critical pair induces the following application conditions ac_K and ac_{K^*} on K:
$\mathrm{ac}_K = \mathrm{Shift}(o_1, \mathrm{ac}_{L_1}) \wedge \mathrm{Shift}(o_2, \mathrm{ac}_{L_2})$, called *extension AC*, and
$\mathrm{ac}_K^* = \neg(\mathrm{ac}_{K,d_{12}}^* \wedge \mathrm{ac}_{K,d_{21}}^*)$, called *conflict-inducing AC*, with

if ($\exists\, d_{12}$ with $k_2 \circ d_{12} = o_1$) then $\mathrm{ac}_{K,d_{12}}^* = \mathrm{L}(p_2^*, \mathrm{Shift}(c_2 \circ d_{12}, \mathrm{ac}_{L_1}))$ else false
if ($\exists\, d_{21}$ with $k_1 \circ d_{21} = o_2$) then $\mathrm{ac}_{K,d_{21}}^* = \mathrm{L}(p_1^*, \mathrm{Shift}(c_1 \circ d_{21}, \mathrm{ac}_{L_2}))$ else false

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 POs.



Note that, given direct transformations $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ with ACs, there is already a weak critical pair as above with a monomorphism $m : K \to G$ and $m \models \mathrm{ac}_K$. Parallel dependency of the given pair implies in addition $m \models \mathrm{ac}_K^*$ (see Lemma 4). For this reason ac_K and ac_K^* are called extension and conflict-inducing AC, respectively.

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

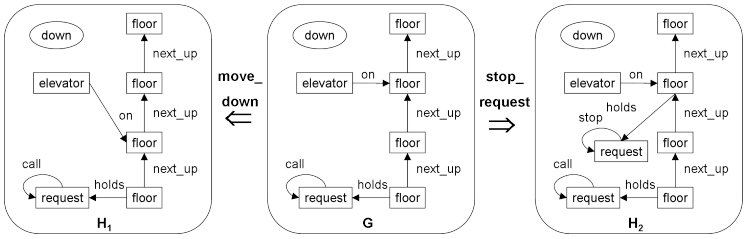


Fig. 3. Parallel dependent pair of direct transformations with ACs of *Elevator*

these transformations are plain parallel independent. In particular, they are parallel dependent because rule *move_down* can not be applied to H_2 since rule *stop_request* (see Fig. 2) adds a stop request to the elevator floor, which is forbidden by the AC of rule *move_down*. We construct the weak critical pair with ACs $P_1 \Leftarrow_{\rho_1,o_1} K \Rightarrow_{\rho_2,o_2} P_2$ for the rules $\rho_1 =$*move_down* and $\rho_2 =$*stop_request* by zooming in to the overlapping of the corresponding LHSs of these rules leading to K, being identical to the LHS of *move_down*. It is depicted on the left in Fig. 5 at the end of Section 4. 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 $\exists pos_3$ is not satisfied by $o_1 = id_L$. 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 $\nexists neg_2$, 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 $\exists id_K$.

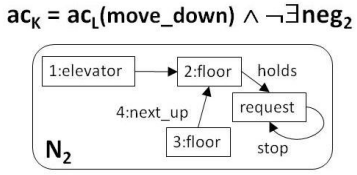


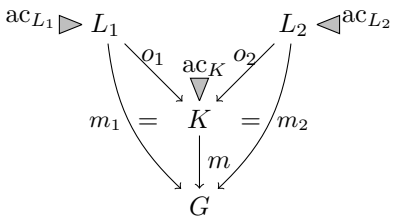
Fig. 4. ac_K for critical pair of *Elevator*

Lemma 3 (completeness of weak critical pairs with ACs). 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 weak 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 $m \in \mathcal{M}$ and $m \models \mathrm{ac}_K \wedge \mathrm{ac}_K^*$.

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

Proof. First, we show that there is an extension diagram with $m \in \mathcal{M}$ and $m \models \mathrm{ac}_K$: Let $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ be a pair of parallel dependent direct transformations and $H_1 \Leftarrow_{p_1,m_1} G \Rightarrow_{p_2,m_2} H_2$ be the underlying pair without ACs. From the $\mathcal{E}$-$\mathcal{M}$ pair factorization for m_1 and m_2 there exists an object K and morphisms $m \in \mathcal{M}$, $o_1 \colon L_1 \to K$, $o_2 \colon L_2 \to K$ with $(o_1, o_2) \in \mathcal{E}$ such that $m_1 = m \circ o_1$ and $m_2 = m \circ o_2$.



By the Restriction Theorem without ACs [2], there is a pair of direct transformations $P_1 \Leftarrow_{p_1,o_1} K \Rightarrow_{p_2,o_2} P_2$ leading to the following extension diagram:

$$\begin{array}{ccccc} P_1 & \xleftarrow{p_1, o_1} & K & \xrightarrow{p_2, o_2} & P_2 \\ \downarrow & & \downarrow m & & \downarrow \\ H_1 & \xleftarrow[\rho_1, m_1]{} & G & \xrightarrow[\rho_2, m_2]{} & H_2 \end{array}$$

By assumption, $m_1 \models \mathrm{ac}_{L_1}$ and $m_2 \models \mathrm{ac}_{L_2}$. By the Shift-Lemma 1, we have

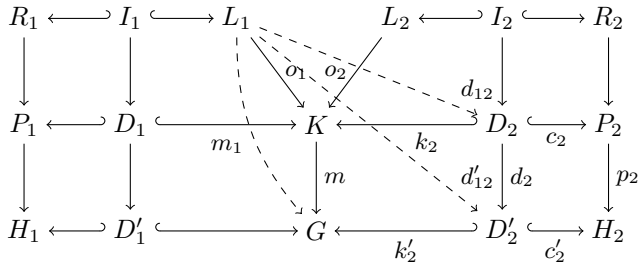
$$\begin{aligned} m_1 = m \circ o_1 \models \mathrm{ac}_{L_1} &\Leftrightarrow m \models \mathrm{Shift}(o_1, \mathrm{ac}_{L_1}) \\ m_2 = m \circ o_2 \models \mathrm{ac}_{L_2} &\Leftrightarrow m \models \mathrm{Shift}(o_2, \mathrm{ac}_{L_2}) \end{aligned}$$

Consequently, $m \models \mathrm{Shift}(o_1, \mathrm{ac}_{L_1}) \wedge \mathrm{Shift}(o_2, \mathrm{ac}_{L_2}) = \mathrm{ac}_K$. It remains to show that $m \models \mathrm{ac}_K^*$. According to parallel dependence of $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$, we have four cases.

1. $\nexists\, d'_{12} \colon k'_2 \circ d'_{12} = m_1$.
2. $\exists\, d'_{12} \colon k'_2 \circ d'_{12} = m_1$,but $c'_2 \circ d'_{12} \not\models \mathrm{ac}_{L_1}$.

3.+4. Symmetric cases for d'_{21}.

By definition of ac^*_K, we have $m \models \mathrm{ac}^*_K \Leftrightarrow m \not\models \mathrm{ac}^*_{K,d_{12}}$ or $m \not\models \mathrm{ac}^*_{K,d_{21}}$.



Case 1. $\nexists\, d'_{12}\colon k'_2 \circ d'_{12} = m_1$. Then $\nexists d_{12}\colon k_2 \circ d_{12} = o_1$. By definition, $\mathrm{ac}^*_{K,d_{12}} =$ false, i.e. $m \not\models \mathrm{ac}^*_{K,d_{12}}$. Thus, $m \models \mathrm{ac}^*_K$.
Case 2. $\exists\, d'_{12}\colon k'_2 \circ d'_{12} = m_1$, but $c'_2 \circ d'_{12} \not\models \mathrm{ac}_{L_1}$.
Case 2.1. $\nexists\, d_{12}\colon k_2 \circ d_{12} = o_1$. Then $m \models \mathrm{ac}^*_K$ as in Case 1.
Case 2.2. $\exists\, d_{12}\colon k_2 \circ d_{12} = o_1$. By definition, $\mathrm{ac}^*_{K,d_{12}} = \mathrm{L}(p^*_2, \mathrm{Shift}(c_2 \circ d_{12}, \mathrm{ac}_{L_1}))$ and $d_2 \circ d_{12} = d'_{12}$ because $k'_2 \circ d_2 \circ d_{12} = m \circ k_2 \circ d_{12} = m \circ o_1 = k'_2 \circ d'_{12}$ and k'_2 is in $\mathcal{M}$. By the Shift-Lemmas 1 and 2, $c'_2 \circ d'_{12} = p_2 \circ c_2 \circ d_{12} \models \mathrm{ac}_{L_1} \Leftrightarrow p_2 \models \mathrm{Shift}(c_2 \circ d_{12}, \mathrm{ac}_{L_1}) \Leftrightarrow m \models \mathrm{L}(p^*_2, \mathrm{Shift}(c_2 \circ d_{12}, \mathrm{ac}_{L_1})) = \mathrm{ac}^*_{K,d_{12}}$. By Case 2, $c'_2 \circ d'_{12} \not\models \mathrm{ac}_{L_1}$ and therefore, $m \not\models \mathrm{ac}^*_{K,d_{12}}$. Thus, $m \models \mathrm{ac}^*_K$.
Case 3 and 4 are symmetric using $m \not\models \mathrm{ac}^*_{K,d_{21}}$. Thus, $m \models \mathrm{ac}^*_K$. □

Lemma 4 (characterization of parallel dependency with ACs). Given a general pair $H_1 \Leftarrow_{\rho_1,m_1} K \Rightarrow_{\rho_2,m_2} H_2$ with weak critical pair $P_1 \Leftarrow_{\rho_1,o_1} K \Rightarrow_{\rho_2,o_2} P_2$, ac_K, ac^*_K, and extension diagram with $m \in \mathcal{M}$ and $m \models \mathrm{ac}_K$, then we have:

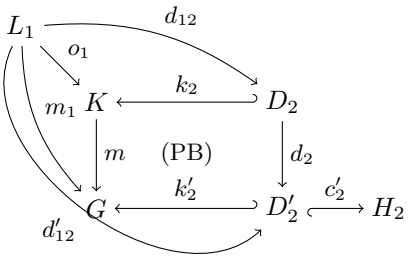
$$H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2 \text{ is parallel dependent} \iff m \models \mathrm{ac}^*_K.$$

Proof. "⇒". By completeness of weak critical pairs.
"⇐". If the general pair is plain parallel dependent, then it is also parallel dependent and we are done. Otherwise, the general pair is plain parallel independent and we have both properties 1. and 2. below:

(1) $\exists\, d'_{12}\colon k'_2 \circ d'_{12} = m_1$.

(2) $\exists\, d'_{21}\colon k'_1 \circ d'_{21} = m_2$.

From property (1) and the PB-property of D_2, we obtain a unique $d_{12}\colon L_1 \to D_2$ with $k_2 \circ d_{12} = o_1$ and $d_2 \circ d_{12} = d'_{12}$ using $k'_2 \circ d'_{12} = m_1 = m \circ o_1$.



In part 2 of the proof of the completeness theorem, we have shown for this case with $d'_{12} = d_2 \circ d_{12}$,

(3) $c'_2 \circ d'_{12} \models \mathrm{ac}_{L_1} \Leftrightarrow m \models \mathrm{ac}^*_{K,d_{12}}$.

Similarly, property (2) implies

(4) $c'_1 \circ d'_{21} \models \mathrm{ac}_{L_2} \Leftrightarrow m \models \mathrm{ac}^*_{K,d_{21}}$.

By assumption we have $m \models ac^*_K$ and, by definition of ac^*_K,

(5) $m \models \mathrm{ac}^*_K \Leftrightarrow m \not\models \mathrm{ac}^*_{K,d_{12}}$ or $m \not\models \mathrm{ac}^*_{K,d_{21}}$.

In case $m \not\models \mathrm{ac}^*_{K,d_{12}}$, we have by (3) $c'_2 \circ d'_{12} \not\models \mathrm{ac}_{L_1}$, which implies parallel dependence. Similar, in case $m \not\models \mathrm{ac}^*_{K,d_{21}}$, we have by (4) $c'_1 \circ d'_{21} \not\models \mathrm{ac}_{L_2}$, which also implies parallel dependence. □

A critical pair is a weak critical pair with an additional condition, ensuring that there is at least one way to extend the critical pair such that in a larger context G conflicting transformations arise.

Definition 10 (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, provided that there exists a monomorphism $m\colon K \to G$ such that $m \models \mathrm{ac}_K \wedge \mathrm{ac}_{K^*}$ and the morphism $m_i = m \circ o_i$ has a pushout complement with respect to p_i $(i = 1, 2)$, i.e. the matches m_i satisfy the gluing condition.

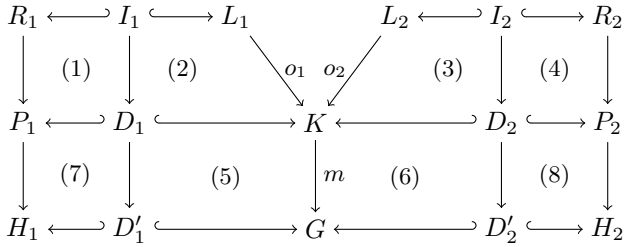
Remark 3. As proven in [9], nested application conditions are expressively equivalent to first order graph formulas and therefore, the satisfiability problem for nested application conditions is undecidable. However in [19,20], techniques are presented to tackle the satisfiability problem in praxis, which is necessary to be able to construct the set of critical pairs for rules with ACs effectively.

Note that this new critical pair notion is different from the one for rules with NACs [8], since here the critical pair is AC-disregarding. Moreover, note that there exists an extension to a parallel dependent pair, but for other extensions we may obtain parallel independent pairs. In particular, this is the case if the conflict-inducing application condition ac_{K^*} of the critical pair is not fulfilled by the extension morphism m into the larger context G.

Lemma 5 (characterization of critical pairs with ACs). Given a weak critical pair $P_1 \Leftarrow_{\rho_1,o_1} K \Rightarrow_{\rho_2,o_2} P_2$ for (ρ_1, ρ_2), we have that the weak critical pair is a critical pair $\iff$ there is a parallel dependent pair $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ and monomorphism $m\colon K \to G$ with extension diagram

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

Proof. "⇒": Given a weak critical pair $P_1 \Leftarrow_{\rho_1,o_1} K \Rightarrow_{\rho_2,o_2} P_2$ with $\mathrm{ac}_K, \mathrm{ac}^*_K$ and $m\colon K \to G$ a monomorphism with $m \models \mathrm{ac}_K \wedge \mathrm{ac}^*_K$ and m has a pushout complement with respect to the rules p^*_i (see Remark 4) for $i = 1, 2$, we have the pushouts (1)–(6) by assumption and can construct pushouts (7) and (8).



By definition of ac_K, $m \models \mathrm{ac}_K \Leftrightarrow m \models \mathrm{Shift}(o_1, \mathrm{ac}_{L_1}) \wedge m \models \mathrm{Shift}(o_2, \mathrm{ac}_{L_2})$ and, by the Shift-Lemma 1, $m_i = m \circ o_i \models \mathrm{ac}_{L_i} \Leftrightarrow m \models \mathrm{Shift}(o_i, \mathrm{ac}_{L_i})$ $(i = 1, 2)$. Hence $m \models \mathrm{ac}_K$ implies $m_i \models \mathrm{ac}_{L_i}$ $(i = 1, 2)$ such that $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ defined by pushouts (1)–(8) is AC-consistent. By Lemma 4, this sequence is parallel dependent using $m \models \mathrm{ac}^*_K$.
"⇐": Given condition 2, the weak critical pair is a critical pair, because we have $m\colon K \to G$ a monomorphism such that m_i has a pushout complement with respect to p_i by pushouts (2)+(5) and (3)+(6). Moreover $m \models \mathrm{ac}_K$, because $m_i \models \mathrm{ac}_{L_i}$ $(i = 1, 2)$ by assumption on AC-consistency of $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ and the equivalence shown above. Finally, $m \models \mathrm{ac}^*_K$ by Lemma 4 using parallel dependence of the given pair. □

Remark 4. The condition "m_i has a pushout complement with respect to p_i" in the critical pair definition (Def. 10) is equivalent to the condition "m has a pushout complement with respect to the rule $p^*_i = \langle K \hookleftarrow D_i \hookrightarrow P_i \rangle$" where p^*_i is the derived rule of $K \Rightarrow_{p_i,o_i} P_i$ $(i = 1, 2)$.

Remark 5. Note that in Lemma 5 the pair $P_1 \Leftarrow_{\rho_1,o_1} K \Rightarrow_{\rho_2,o_2} P_2$ is AC-disregarding, while $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ satisfies ac_{L_1} of ρ_1 and ac_{L_2} of ρ_2.

Theorem 3 (completeness of critical pairs with ACs). 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 a monomorphism m and $m \models \mathrm{ac}_K \wedge \mathrm{ac}^*_K$.

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

Proof. The proof is based on the completeness of critical pairs without ACs in [2]. In a first step, we obtain the completeness of weak critical pairs with ACs by showing $m \models \mathrm{ac}_K$ for $\mathrm{ac}_K = \mathrm{Shift}(o_1, \mathrm{ac}_{L_1}) \wedge \mathrm{Shift}(o_2, \mathrm{ac}_{L_2})$ by shift of ACs over morphisms (Lemma 1) and $m \models \mathrm{ac}^*_K$ for $\mathrm{ac}^*_K = \neg(\mathrm{ac}^*_{K,d_{12}} \wedge \mathrm{ac}^*_{K,d_{21}})$ by shift of ACs over morphisms and rules (Lemma 1 and 2). This proof is shown in Lemma 3. In a second step, we show that a weak critical pair is a critical pair iff it can be extended to a parallel dependent pair (Lemma 5), where the proof of Lemma 5 is based on the fact that $m \models \mathrm{ac}^*_K$ iff $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ is parallel dependent (Lemma 4). □

Example 5 (critical pair). Because of Theorem 3, the weak critical pair as described in Example 4 is, in particular, a critical pair. Therefore, the extension morphism $m : K \to G$ as shown in Fig. 5 fulfills the conjunction of the extension condition ac_K and the conflict-inducing condition ac^*_K.

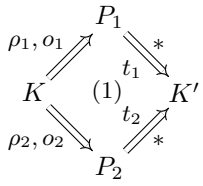
Example 6 (critical pair with non-trivial ac^*_K). As noticed already in Example 4, the conflict-inducing condition ac^*_K is equivalent to true for each monomorphic extension morphism of the critical pair, leading to conflicting transformations for each extension of the critical pair. Now we illustrate 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 $\nexists neg : \bullet \rightarrow \bullet\bullet\bullet$, producing a node only if there do not exist already two other nodes. In this case, $\mathrm{ac}_K = \nexists neg : \bullet \rightarrow \bullet\bullet\bullet$ and $\mathrm{ac}^*_K = \exists pos_1 : \bullet \rightarrow \bullet\bullet \vee \exists pos_2 : \bullet \rightarrow \bullet\bullet\bullet$ with $K = \bullet$. The extension condition ac_K expresses that each extension morphism $m : K \to G$ into a graph 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.

6 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 as introduced in the previous section. In order to show local confluence, we have, roughly speaking, to require that all critical pairs are confluent. Unfortunately, confluence of critical pairs is not sufficient to show local confluence. As shown in [4,5] and discussed in [2], strict confluence of critical pairs is needed. To handle rules with application conditions, AC-compatibility is needed in addition.

Definition 11 (strict AC-confluence). A critical pair $P_1 \Leftarrow_{\rho_1,o_1} K \Rightarrow_{\rho_2,o_2} P_2$ for $\langle \rho_1, \rho_2 \rangle$ with induced ac_K and ac_K^* on K is called *strictly AC-confluent*, if

1. The pair is *plain strictly confluent*, i.e. strictly confluent in the sense of [2] with AC-disregarding transformations t_1 and t_2.



2. The extended AC-disregarding transformations $\bar{t}_i = K \Rightarrow_{p_i,o_i} P_i \Rightarrow_{t_i}^* K'$ $(i = 1, 2)$ with derived ACs $\mathrm{ac}(\bar{t}_i)$ on K are *AC-compatible*, i.e. $\mathrm{ac}_K \wedge \mathrm{ac}_K^* \Rightarrow \mathrm{ac}(\bar{t}_1) \wedge \mathrm{ac}(\bar{t}_2)$.

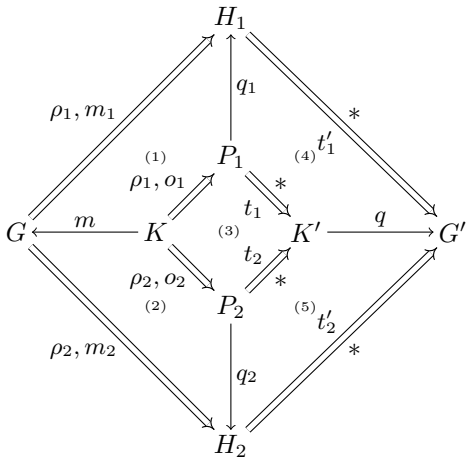
Remark 6. It is important to note that the strict confluence diagram (1) includes AC-disregarding transformations t_1 and t_2. The ACs occurring in t_1 and t_2 are used in the construction of $\mathrm{ac}(\bar{t}_1) \wedge \mathrm{ac}(\bar{t}_2)$ and in the Local Confluence Theorem to show that the extended transformations $\bar{t}_1'$ and $\bar{t}_2'$ of $\bar{t}_1$ and $\bar{t}_2$ along $m : K \to G$ satisfy ACs.

Now we define adhesive rewriting systems with ACs and prove a Local Confluence Theorem, also called *Critical Pair Lemma*, for this case, which generalizes the Local Confluence Theorem without ACs [2]. It is based on the new notion of critical pairs and on strict AC-confluence as defined above.

Definition 12 (adhesive rewriting system with ACs). An *adhesive rewriting system with nested application conditions* consists of a set of rules with ACs, where rewriting is defined by DPO transformations as given in Def. 5.

Theorem 4 (Local Confluence with ACs). An adhesive rewriting system with nested application conditions is locally confluent, if all critical pairs are strictly AC-confluent.

Proof. For parallel independent pairs with ACs, we have local confluence using the Local Church-Rosser Theorem in [17]. By the Local Confluence Theorem without ACs [2] and the Completeness Theorem 3 of critical pairs, we have for each parallel dependent pair $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ with ACs a critical pair $P_1 \Leftarrow_{p_1,o_1} K \Rightarrow_{p_2,o_2} P_2$ and embedding diagrams (1) and (2) with a monomorphism $m \models (\mathrm{ac}_K \wedge \mathrm{ac}_K^*)$. By plain strict confluence, we have diagram (3) and, by Local Confluence without ACs, we have corresponding extensions t_1' of t_1 and t_2' of t_2 in diagrams (4) and (5).



It remains to show that $\bar{t}_i' \colon G \Rightarrow_{\rho_i,m_i} H_i \Rightarrow_{t_i'}^* G'$ for $i = 1, 2$ satisfies the corresponding ACs. For this purpose, we have to extend the Embedding Theorem without ACs [2] to the case with ACs. This is shown as Theorem 1 of the appendix based on Lemma 1 and 2. By this Embedding Theorem with ACs an (AC-disregarding) transformation $t \colon G_0 \Rightarrow^* G_n$ can be extended by a morphism $k_0 \colon G_0 \to G_0'$ to a transformation $t' \colon G_0' \Rightarrow^* G_n'$ regarding the ACs if k_0 is consistent with respect to t. Consistency means boundary consistency (as in the case without ACs) and AC-consistency, i.e. $k_0 \models \mathrm{ac}(t)$, where $\mathrm{ac}(t)$ is the derived application condition of t (see Def. 6). In our context, AC-consistency means that we have to show $m \models \mathrm{ac}(\bar{t}_1)$ and $m \models \mathrm{ac}(\bar{t}_2)$, where $\bar{t}_i : K \Rightarrow_{\rho_i,o_i} P_i \Rightarrow_{t_i}^* K'$, $(i = 1, 2)$. Note that AC-satisfaction of $\bar{t}_1$ and $\bar{t}_2$ is not required, but that of $\bar{t}_1'$ and $\bar{t}_2'$ is a consequence. Now, by AC-confluence, especially, AC-compatibility, we have

$$\mathrm{ac}_K \wedge \mathrm{ac}_K^* \Rightarrow \mathrm{ac}(\bar{t}_1) \wedge \mathrm{ac}(\bar{t}_2).$$

Since we have already $m \models (\mathrm{ac}_K \wedge \mathrm{ac}_K^*)$, this implies $m \models \mathrm{ac}(\bar{t}_1) \wedge \mathrm{ac}(\bar{t}_2)$ and we are done. □

Remark 7. Typed graph transformation systems are a valid instantiation of adhesive rewriting systems [11,2]. Therefore, the above Local Confluence Theorem for rules with ACs, in particular, holds for typed graph rules with ACs.

Remark 8. For showing that a strictly confluent critical pair is also AC-confluent, AC-compatibility of the strict solution needs to be shown. For this purpose, the following implication problem $\mathrm{ac}_K \wedge \mathrm{ac}_K^* \Rightarrow \mathrm{ac}(\bar{t}_1) \wedge \mathrm{ac}(\bar{t}_2)$ has to be solved (see Def. 11 of strict AC-confluence). As proven in [9], the implication problem for nested application conditions is undecidable in general. However in [21], techniques are shown to tackle the implication problem in praxis. As noted in the introduction already, showing confluence for GTS without ACs is undecidable in general [4]. It is noted as well that at least the sufficient condition for local confluence – showing that all critical pairs are strictly confluent – is semi-decidable. ACs complicate this sufficient condition, since on the one hand, in general, the set of critical pairs for rules with ACs can not be computed statically anymore (see Remark 3), and on the other hand, showing AC-compatibility on top of strict confluence is, in general, undecidable. Concluding, the sufficient condition of strict AC-confluence of all critical pairs is undecidable in general. We have seen that the complexity of confluence analysis for transformation systems with ACs is strongly related with the complexity of solving ACs, which is quite natural as a result. Therefore, the success of confluence analysis with ACs depends on the success of solving the implication and satisfiability problem for ACs [21,19,20], showing once again the importance of this topic.

Example 7 (Local Confluence with ACs). In order to apply Thm 4 note that the critical pair in Example 4 and 5 is strictly AC-confluent, since rule *process_stop_down* and then rule *move_down* can be applied to P_2 leading to $K' = P_1$ (see Fig. 5, where $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ is shown explicitly in Fig. 3). This is a strict solution, since neither some floor, nor the elevator is deleted by the solution, the structure which is commonly preserved by the critical pair. Moreover, this solution is AC-compatible, because of the following argumentation. 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 $\mathrm{ac}_K = \mathrm{Shift}(o_1, \mathrm{ac}_{L_1}) \wedge \mathrm{Shift}(o_2, \mathrm{ac}_{L_2})$. Conclude that the pair $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ is locally confluent as shown in the outer diagram of Fig. 5 by Theorem 4. 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.

7 Conclusion, Related and Future Work

In this paper we have presented a new method for checking local confluence for graph transformation systems (and, in general, adhesive rewrite systems) with nested application conditions. This kind of ACs provide a considerable increase of expressive power with respect to the NACs that were used up to now. 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 [22,8]to prove

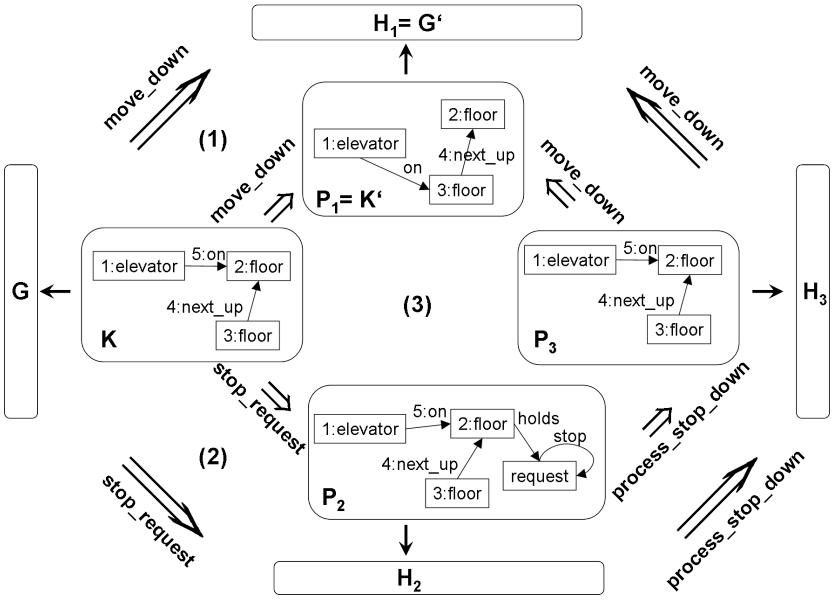


Fig. 5. strictly AC-confluent critical pair of *Elevator*

local confluence of GTS with NACs. In particular, in those papers the critical pairs were 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.

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 [7], 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 [16] 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 [22,8].

Graph conditions with a similar expressive power as nested conditions were introduced in [23], while nested conditions were introduced in [24]. A detailed account of many results on nested conditions, including their expressive power can be found in [9]. Finally, some results, in particular the Parallelism and Concurrency Theorems about graph transformation with nested conditions, which can be considered a previous step to the results presented in this paper, are presented in [17].

Since critical pairs with NACs are 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 [21,19,20] for nested application conditions.

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, Berlin (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, New York (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. Dershowitz, N., Plaisted, D.A.: Rewriting. In A. Robinson and A. Voronkov, editors, *Hanbook of Automated Reasoning, Volume I* (2001) 535–610
7. Habel, A., Heckel, R., Taentzer, G.: Graph grammars with negative application conditions. Fundamenta Informaticae **26** (1996) 287–313
8. 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
9. Habel, A., Pennemann, K.H.: Correctness of high-level transformation systems relative to nested conditions. Mathematical Structures in Computer Science **19** (2009) 245–296
10. 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
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. Ehrig, H., Habel, A., Kreowski, H.J., Parisi-Presicce, F.: Parallelism and concurrency in high level replacement systems. Mathematical Structures in Computer Science **1** (1991) 361–404
13. Baunetz Wissen: Aufzüge und Fahrtreppen: Einknopf-Sammelsteuerung. http://www.baunetzwissen.de/standardartikel/Aufzuege-und-Fahrtreppen_Einknopf-Sammelsteuerung_149062.html in German.
14. Habel, A., Pennemann, K.H.: Nested constraints and application conditions for high-level structures. In: Formal Methods in Software and System Modeling. Volume 3393 of LNCS., Springer (2005) 293–308
15. 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

16. 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
17. Ehrig, H., Habel, A., Lambers, L.: Parallelism and concurrency theorems for rules with nested application conditions. In: Manipulation of Graphs, Algebras and Pictures. Essays Dedicated to Hans-Jörg Kreowski on the Occasion of His 60th Birthday. (2009)
18. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Löwe, M.: Algebraic approaches to graph transformation. Part I: Basic concepts and double pushout approach. In: Handbook of Graph Grammars and Computing by Graph Transformation. Volume 1. World Scientific (1997) 163–245
19. 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.
20. Orejas, F.: Attributed Graph Constraints. In: Graph Transformations (ICGT'08). Volume 5214 of Lecture Notes in Computer Science., Springer-Verlag (2008) 274–288
21. Pennemann, K.H.: Resolution-like theorem proving for high-level conditions. In: Graph Transformations (ICGT 2008). Volume 5214 of LNCS., Springer (2008) 289–304
22. 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
23. Rensink, A.: Representing first-order logic by graphs. In: Graph Transformations (ICGT'04). Volume 3256 of LNCS., Springer (2004) 319–335
24. Habel, A., Pennemann, K.H.: Satisfiability of high-level conditions. In: Graph Transformations (ICGT 2006). Volume 4178 of LNCS., Springer (2006) 430–444