Fundamenta Informaticae 118 (2012) 35–63
DOI 10.3233/FI-2012-705
IOS Press

$\mathcal{M}$-Adhesive Transformation Systems with Nested Application Conditions. Part 2: Embedding, Critical Pairs and Local Confluence

Hartmut Ehrig
Technische Universität Berlin, Germany
ehrig@cs.tu-berlin.de

Ulrike Golas
Konrad-Zuse-Zentrum für Informationstechnik Berlin, Germany
golas@zib.de

Annegret Habel
Carl v. Ossietzky Universität Oldenburg, Germany
habel@informatik.uni-oldenburg.de

Leen Lambers[*,†]
Hasso Plattner Institut, Universität Potsdam, Germany
Leen.Lambers@hpi.uni-potsdam.de

Fernando Orejas[‡]
Universitat Politècnica de Catalunya, Spain
orejas@lsi.upc.edu

Abstract. Graph transformation systems have been studied extensively and applied to several areas of computer science like formal language theory, the modeling of databases, concurrent or distributed systems, and visual, logical, and functional programming. In most kinds of applications it is necessary to have the possibility of restricting the applicability of rules. This is usually done by means of application conditions. In this paper, we continue the work of extending the fundamental theory of graph transformation to the case where rules may use arbitrary (nested) application conditions. More precisely, we generalize the Embedding theorem, and we study how local confluence can be checked in this context. In particular, we define a new notion of critical pair which allows us to formulate and prove a Local Confluence Theorem for the general case of rules with nested application conditions. All our results are presented, not for a specific class of graphs, but for any arbitrary $\mathcal{M}$-adhesive category, which means that our results apply to most kinds of graphical structures. We demonstrate our theory on the modeling of an elevator control by a typed graph transformation system with positive and negative application conditions.

[*]The work of this author was partially funded by the Deutsche Forschungsgemeinschaft in the course of the project - Correct Model Transformations - see `http://www.hpi.uni-potsdam.de/giese/projekte/kormoran.html?L=1`.
[†]Address for correspondence: Fachgebiet Systemanalyse und Modellierung, Hasso-Plattner-Institut für Softwaresystemtechnik GmbH, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam
[‡]The work of this author was partially supported by the MEC project FORMALISM (ref. TIN2007-66523) and by the AGAUR grant to the research group ALBCOM (ref. 00516).

Keywords: $\mathcal{M}$-adhesive transformation systems, $\mathcal{M}$-adhesive categories, graph replacement categories, nested application conditions, embedding, critical pairs, local confluence

1. Introduction

There is a group of general results which are considered to be the fundamental theory of graph transformation, and which are the basis for more specific results in different application areas of graph transformation. In [29] we can find these results for the case of plain (i.e. without application conditions) graph transformation systems, and in [5] for the case of adhesive transformation systems. In [7], we present a first part of this fundamental theory, for the general case of rules with nested application conditions. In particular, in [7], we present the Parallelism, Concurrency and Amalgamation Theorems. In this sense, in this paper we present the second part, including the Embedding, Critical Pairs and Local Confluence theorems. The overall aim of this work is to put together important results which are scattered over some conference and workshop papers. In particular, a short version of the results concerning critical pairs and local confluence in this paper has been presented without proofs in the proceedings of ICGT 2010 [10].

The Embedding Theorem is the result that states under which conditions a given transformation can be applied in a larger context. In the case of rules without application conditions, this means ensuring that the larger context satisfies the gluing conditions of all the rules applied in the given transformation. When rules may include application conditions this may be not enough, since the larger context may violate some condition of a rule application. In order to see if this may happen, given a transformation involving the application of a number of rules, we show how we can collect the application conditions of all these rule applications into a single *derived application condition*, so that the given larger context would violate any of the conditions of the rules if and only if it would violate the derived application condition.

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. [6]). Unfortunately, confluence of rewriting systems is undecidable in general. 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 [18] 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. [26, 27, 6]). 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 [27]. The notion of local confluence is closely related to the *embedding* of transformations, since joinability of critical pairs in a minimal context needs to be replicated in a larger context by embedding the corresponding transformations.

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, in view of several applications, we may want to limit the applicability of rules. This leads to the notion of *application conditions (ACs)*, 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 [14], 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 GTSs with ACs are quite recent and are restricted to the case of NACs [21]. 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 [17] a very general kind of conditions, called *nested application conditions*, is studied in detail. In particular, in the case of graphs, finite nested application conditions have the expressive power of the first-order fragment of Courcelle's logic of graphs [3]. Following that work, in this paper we study embedding and 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 [21]. 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 notions of graphs and introduce our running example on an elevator control, which is used throughout the paper to illustrate the main concepts and results. In Section 3, we introduce the concepts of nested application conditions and rules. In Section 4, we prove the first main results of this paper, the Embedding and Extension Theorems for rules with nested application conditions, allowing to extend a transformation to a larger context. In Section 5, we review the problems related to checking the confluence of GTSs. In Section 6, we define our new notion of critical pairs and state a completeness theorem (Theorem 6.10), saying that every pair of dependent direct transformations is an extension of a critical pair. In Section 7, we state as main result the Local Confluence Theorem (Theorem 7.3) for rules with ACs. A conclusion including related work is given in Section 8. In Appendix A, we give a short overview of the categorical notion of $\mathcal{M}$-adhesive categories and their properties that are relevant for this paper.

Acknowledgement. Many thanks to the referees for their careful reading of the draft of this paper, stimulating remarks, and suggestions which led to a considerably improved exposition.

2. Graphs and High-Level Structures

In this section, we review the definitions of typed graphs and graph morphisms and introduce our running example on the modeling of an elevator control. Moreover, we give a short introduction to $\mathcal{M}$-adhesive categories to define the setting for the results in this paper.

To introduce our example, we first review our notion of typed graphs.

Definition 2.1. ((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 $G_i = (G_{V,i}, G_{E,i}, s_i, t_i)$, $(i = 1, 2)$ 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 *TG-typed graph*, or short typed graph, if the type graph is clear in the context. Consider TG-typed graphs $G_1^T = (G_1, type_1)$ and $G_2^T = (G_2, type_2)$, a *TG-typed graph morphism* $f : G_1^T \to G_2^T$ is a graph morphism $f : G_1 \to G_2$ such that $type_2 \circ f = type_1$. The category having TG-typed graphs as objects and TG-typed graph morphisms as arrows is called **Graphs**$_{TG}$.

Example 2.2. (*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. Both ellipses and rectangles represent the nodes, while arrows represent the edges. 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 the transitive closure of opposite edges of next_up), but these are not depicted because of visibility reasons. Moreover, constraints could be used to restrict these graphs that are valid elevator models. But in our example, this is not necessary, because our well-formed rules transform a valid elevator again into a valid model, i.e. as long as we start with a valid model nothing undesired may happen. Nevertheless, the typing is necessary to restrict the matches for the rules.

In Example 3.4, some graph transformation rules with ACs modeling the elevator control are presented. We continue with the running example in Section 3, where we introduce rules with ACs, and then in Section 6 and 7, 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.

The results presented in this paper do not only apply to standard graph transformation systems, even if our examples use typed graphs. Actually, our results apply to transformation systems over any $\mathcal{M}$-adhesive category [8]. The idea behind the consideration of $\mathcal{M}$-adhesive categories is to avoid similar investigations for different instantiations like e.g. Petri nets, hypergraphs, and algebraic specifications.

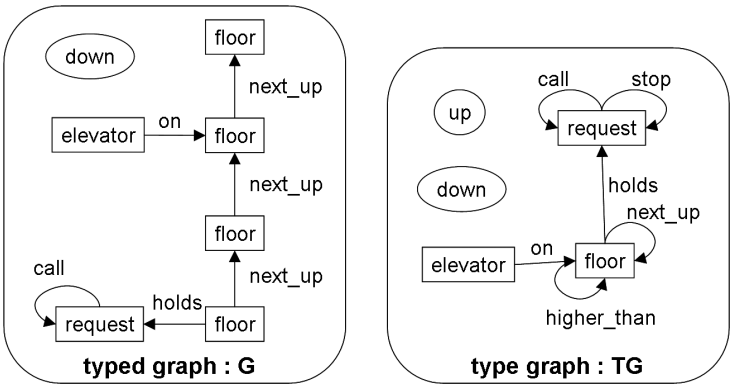


Figure 1. A typed graph of *Elevator* together with its type graph

An $\mathcal{M}$-adhesive category is a category $\mathcal{C}$ with a distinguished morphism class $\mathcal{M}$ of monomorphisms satisfying certain properties. The most important one is the van Kampen (VK) property stating a certain kind of compatibility of pushouts and pullbacks along $\mathcal{M}$-morphisms. Moreover, additional properties are needed in our context: initial pushouts, which describe the existence of a special "smallest" pushout over a morphism, and $\mathcal{E}'$-$\mathcal{M}$ pair factorizations, which extend the classical epi-mono factorization to a pair of morphisms with the same codomain. The precise definitions of $\mathcal{M}$-adhesive categories, initial pushouts, and $\mathcal{E}'$-$\mathcal{M}$ pair factorizations can be found in Appendix A. The following results in this paper all require such an $\mathcal{M}$-adhesive category where the additional properties hold.

Assumption 2.3. We assume that $\langle \mathcal{C}, \mathcal{M} \rangle$ is an $\mathcal{M}$-adhesive category with a unique $\mathcal{E}'$-$\mathcal{M}$ pair factorization (needed for Lemma 3.5, Theorem 6.10) and initial pushouts over $\mathcal{M}$-morphisms (used in Theorem 4.4 and 7.3).

Remark 2.4. ($\langle \mathrm{Graphs}_{TG}, \mathcal{M} \rangle$, $\langle \mathrm{PTNets}, \mathcal{M} \rangle$, $\langle \mathrm{Spec}, \mathcal{M}_{strict} \rangle$ are $\mathcal{M}$-adhesive [6, 8])
The category $\langle \mathrm{Graphs}_{TG}, \mathcal{M} \rangle$ with the class $\mathcal{M}$ of all injective typed graph morphisms is an $\mathcal{M}$-adhesive category. Moreover, typed attributed graphs with a class $\mathcal{M}$ of monomorphisms injective on the graph part and isomorphic on the data type part form an $\mathcal{M}$-adhesive category. The category $\langle \mathrm{PTNets}, \mathcal{M} \rangle$ of place/transition nets with the class $\mathcal{M}$ of all injective net morphisms and the category $\langle \mathrm{Spec}, \mathcal{M}_{strict} \rangle$ of algebraic specifications with the class $\mathcal{M}_{strict}$ of all strict injective specification morphisms are $\mathcal{M}$-adhesive, but not adhesive.

Remark 2.5. ($\langle \mathrm{Graphs}_{TG}, \mathcal{M} \rangle$,$\langle \mathrm{PTNets}, \mathcal{M} \rangle$, $\langle \mathrm{Spec}, \mathcal{M}_{strict} \rangle$ satisfy additional properties [6, 8])
The category $\langle \mathrm{Graphs}_{TG}, \mathcal{M} \rangle$ satisfies all additional requirements. In particular, it has a unique $\mathcal{E}'$-$\mathcal{M}$ pair factorization where $\mathcal{E}'$ is the class of jointly surjective typed graph morphism pairs (i.e., the morphism pairs (e_1, e_2) such that for each $x \in K$ there is a preimage $a_1 \in A_1$ with $e_1(a_1) = x$ or $a_2 \in A_2$ with $e_2(a_2) = x$). For typed attributed graphs [6], the additional properties are formally satisfied, but the theory is not fully satisfactory because we would like to map some parts of the data non-injectively in the pair factorization. Instead, we could use an additional class $\mathcal{M}'$ with suitable properties concerning the compatibility with $\mathcal{M}$ as given in [6, 21]. The category $\langle \mathrm{PTNets}, \mathcal{M} \rangle$ of place/transition nets with the class $\mathcal{M}$ of all injective net morphisms and the category $\langle \mathrm{Spec}, \mathcal{M}_{strict} \rangle$ of

algebraic specifications with the class $\mathcal{M}_{strict}$ of all strict injective specification morphisms satisfy the additional properties, where $\mathcal{E}'$ is the class of all jointly epimorphic morphism pairs.

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 rule-based transformations. Moreover, we introduce typed graph transformation rules with nested application conditions for our running example on the modeling of an elevator control.

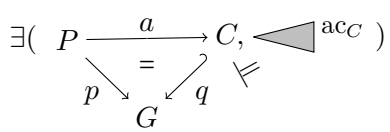
Nested application conditions were defined in [17]. They generalize the corresponding notions in [14, 19, 4], where a negative (positive) application condition, short NAC (PAC), over a graph P, denoted $\neg\exists a$ ($\exists a$) is defined in terms of a morphism $a : P \to C$. Informally, a morphism $m : P \to G$ satisfies $\neg\exists a$ ($\exists a$) if there does not exist a morphism $q : C \to G$ extending m (if there exists q extending 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 3.1. (application condition and satisfaction)
An *application condition* ac_P over an object 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.

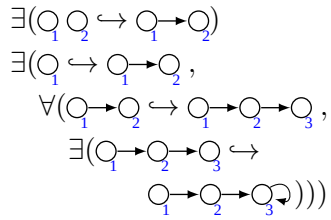
We define inductively when a morphism *satisfies* an application condition:

- 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 $\mathcal{M}$-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 ($i \in I$).



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

Example 3.2. Examples of ACs are given below, where the first one is a standard PAC considered already in [14], 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.

ACs are used to restrict the application of rules to a given object. The idea is to equip the left-hand side of rules with an application condition[1]. Then we can only apply a given rule to an object G if the corresponding match morphism satisfies the AC of the rule. However, for technical reasons[2], we also introduce the application of rules *disregarding* the associated ACs.

Definition 3.3. (rules and transformations with ACs)
A *rule* $\rho = \langle p, \mathrm{ac}_L \rangle$ consists of a *plain* rule $p = \langle L \hookleftarrow I \rightarrow R \rangle$ with $I \hookrightarrow L$ and $I \hookrightarrow R$ morphisms in $\mathcal{M}$ and an application condition ac_L over L.

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

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

Example 3.4. (typed graph rules of *Elevator*)
We show three rules (only their left and right-hand sides[3]) modeling part of the elevator control as given in Example 2.2. In Fig. 2, first, we have the rule move_down with combined AC on L, consisting of three PACs ($\exists \mathrm{pos}_i \colon L \rightarrow P_i, i = 1, \ldots, 3$) and a NAC ($\nexists \mathrm{neg} \colon L \rightarrow N$), 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)[4], 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 as shown in the following lemma.

[1] We could have also allowed to equip the right-hand side of rules with an additional AC, but Lemma 3.7 shows that this case can be reduced to rules with left ACs only.
[2] For example, a critical pair with ACs (see Def. 6.6) consists of transformations that do not need to satisfy the associated ACs.
[3] The intermediate graph I and the span $\mathcal{M}$-morphisms can be derived as follows: the graph I consists of all nodes and edges occurring in L and R that are labeled by the same number. The span morphisms 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.
[4] 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.

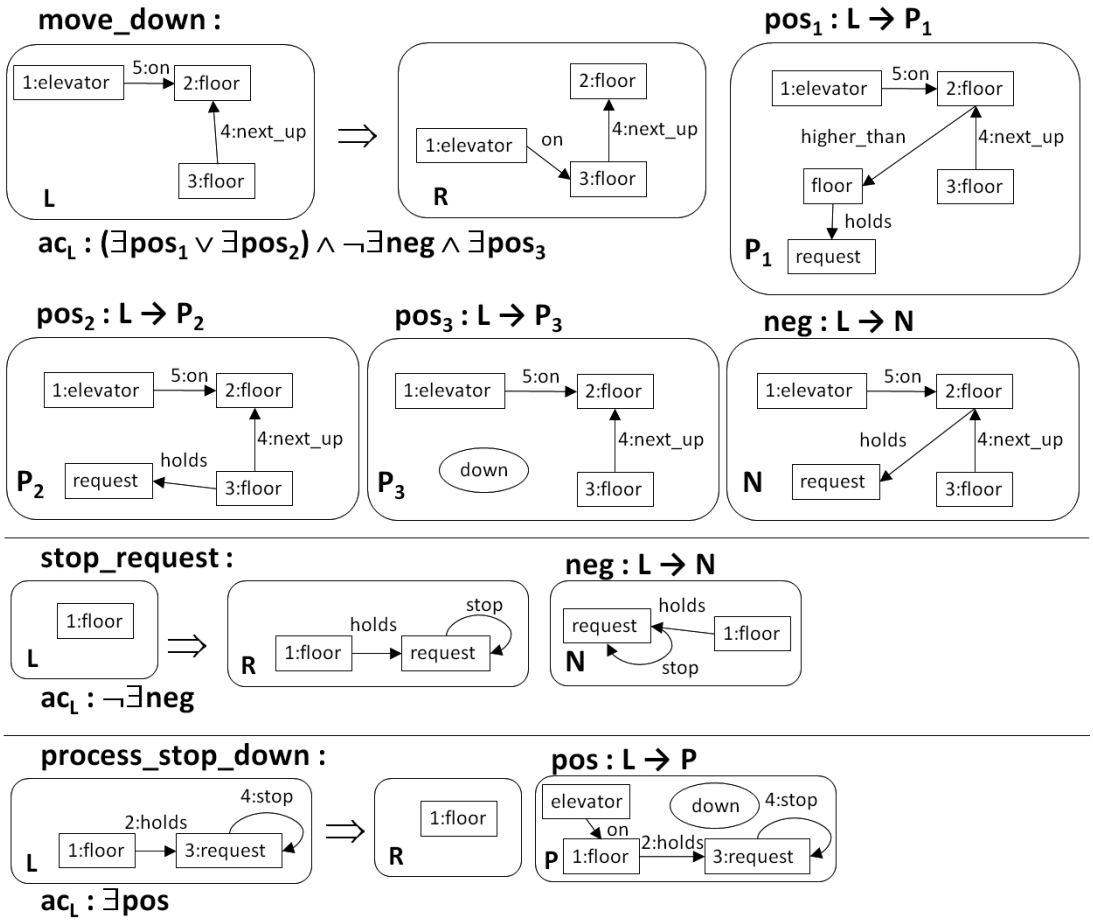


Figure 2. Rules for *Elevator*

Lemma 3.5. (shift of ACs over morphisms [7])
There is a transformation Shift from morphisms and application conditions to application conditions such that for each application condition ac_P and for each morphism $b\colon P \to P'$, 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 1. The transformation Shift is inductively defined as follows:

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

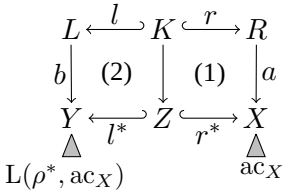
$\mathrm{Shift}(b, \mathrm{true}) = \mathrm{true}$.
$\mathrm{Shift}(b, \exists(a, \mathrm{ac}_C)) = \bigvee_{(a',b')\in\mathcal{F}} \exists(a', \mathrm{Shift}(b', \mathrm{ac}_C))$
if $\mathcal{F} = \{(a', b') \in \mathcal{E}' \mid b' \in \mathcal{M}$ and (1) commutes$\} \neq \emptyset$
$\mathrm{Shift}(b,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

Remark 3.6. Note, that the index set $\mathcal{F}$ used in the formula for Shift may be infinite, which is allowed according to Def. 3.1. However, for practical examples with finite graphs C and P', and $\mathcal{E}'$ consisting of jointly surjective morphisms, the set $\mathcal{F}$ is w.l.o.g. finite.

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 3.7. (shift of ACs over rules [17, 7])
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 $m \models \mathrm{L}(\rho, \mathrm{ac}_R) \Leftrightarrow m^* \models \mathrm{ac}_R$.

Construction 2. The transformation L is inductively defined as follows:

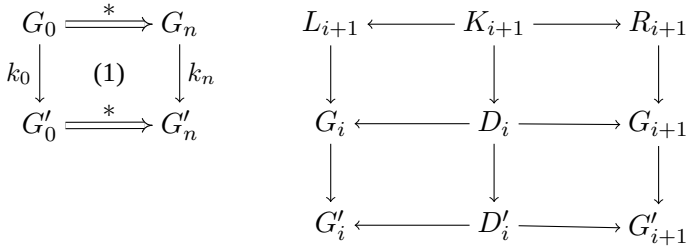


$\mathrm{L}(\rho, \mathrm{true}) = \mathrm{true}$
$\mathrm{L}(\rho, \exists(a, \mathrm{ac}_X)) = \exists(b, \mathrm{L}(\rho^*, \mathrm{ac}_X))$ if $\langle r, a\rangle$ has a pushout complement (1) and $\rho^* = \langle Y \hookleftarrow Z \hookrightarrow X\rangle$ is the derived rule by constructing 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. Embedding and Extension

In this section, we present Embedding and Extension Theorems for rules with ACs, which allow us to extend a transformation via rules with ACs to a larger context. These results are the basis to prove many other results in graph transformation. In particular they are needed to prove the Local Confluence Theorem for rules with ACs in Section 7.

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 on the left. For each rule application and transformation step, we have two double pushout diagrams as shown on the right, where the rule ρ_{i+1} is applied to both G_i and G'_i.



For the notion of consistency of an extension morphism as given in Def. 4.2 as well as for AC-compatibility in Def. 7.1 in Section 7, we introduce the notion of derived rule and derived application condition of a transformation $t : G_0 \Rightarrow^* G_n$. A derived rule $\rho(t)$ describes the combined changes of the transformation by condensing the transformation into a single rule from G_0 to G_n. A derived application condition $\mathrm{ac}(t)$ for a transformation t summarizes all ACs in t into an application condition on G_0. To this extent we use the shift construction in Lemma 3.5 to translate the AC, ac_{L_i}, on the rule ρ_i into an equivalent condition $\mathrm{Shift}(m_i, \mathrm{ac}_{L_i})$ on G_{i-1}. On the other hand, we also know that, applying the

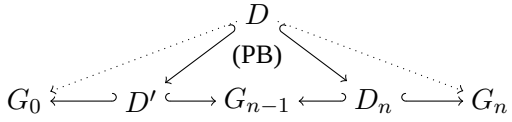
construction in Lemma 3.7, we can transform every condition $\mathrm{Shift}(m_i, \mathrm{ac}_{L_i})$ on G_{i-1} into an equivalent condition ac'_i on G_0. Then, $\mathrm{ac}(t)$ is just the conjunction of all these conditions, i.e. $\mathrm{ac}(t) = \bigwedge_i \mathrm{ac}'_i$.

Definition 4.1. (derived application condition, derived rule)
For a transformation $t\colon G_0 \Rightarrow^* G_n$, the plain derived rule $p(t)$ is defined by the span $G_0 \leftarrow D_0 \rightarrow G_1$ for $n = 1$ and by iterated pullback construction leading to the span $G_0 \leftarrow D \rightarrow G_n$ for $n \geq 2$ as shown below.

Given an AC-disregarding transformation $t\colon G_0 \Rightarrow^* G_n$, the *derived application condition* $\mathrm{ac}(t)$ over G_0 is inductively defined 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 $p^* = \langle G_0 \hookleftarrow D \rightarrow G_{n-1} \rangle$ is the plain derived rule og $G_0 \Rightarrow^* G_{n-1}$.

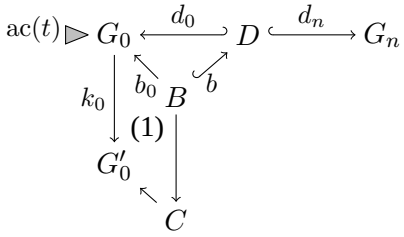
The *derived rule* $\rho(t) = \langle p(t), \mathrm{ac}(t) \rangle$ of an AC-disregarding transformation $t\colon G_0 \Rightarrow^* G_n$ consists of the plain derived rule $p(t)$ and the derived application condition $\mathrm{ac}(t)$ of t.



It may be noticed that, in the above definition, we have defined the notions of derived application condition and derived rule for AC-disregarding transformations, and not for transformations satisfying the ACs. The same happens in the definition and results presented below. The reason is that the constructions used are independent of whether the given transformation satisfies the ACs or not. Moreover, it may happen that the extension of an AC-disregarding transformation does satisfy the given ACs, which is used in some results presented in the sections below.

Definition 4.2. (Boundary consistency, AC-consistency, consistency)
Given an AC-disregarding transformation $t\colon G_0 \Rightarrow^* G_n$ with derived rule $\rho(t) = \langle p(t) = (G_0 \leftarrow D \rightarrow G_n), \mathrm{ac}(t) \rangle$ and a morphism $k_0\colon G_0 \rightarrow G'_0$ then:

1. 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 $d_0 \circ b = b_0$.
2. k_0 is *AC-consistent* with respect to t, if $k_0 \models \mathrm{ac}(t)$.
3. k_0 is *consistent* with respect to t, if k_0 is boundary and AC-consistent with respect to t.



Theorem 4.3. (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.

$$\begin{array}{ccc} G_0 & \overset{*}{\underset{t}{\Longrightarrow}} & G_n \\ {\scriptstyle k_0}\downarrow & (1) & \downarrow{\scriptstyle k_n} \\ G_0' & \overset{*}{\underset{t'}{\Longrightarrow}} & G_n' \end{array}$$

Proof:
Let $t\colon G_0 \Rightarrow^* G_n$ be an AC-disregarding transformation via rules with application conditions and $k_0\colon G_0 \to G_0'$ be consistent with respect to t. Then k_0 is boundary consistent with respect to the underlying transformation t_0 via rules without application conditions and, by the Embedding Theorem for rules without application conditions [6], there is a plain extension diagram t_0' over t_0 and k_0. Moreover, by the Extension Theorem for rules without application conditions [6] the following DPO diagram exists:

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

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, $k_0 \models \mathrm{ac}(t) = \mathrm{Shift}(m_1, \mathrm{ac}_{L_1})$ if and only if $k_0 \circ m_1 \models \mathrm{ac}_{L_1}$.
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:

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

Then

$$\begin{array}{lll} & k_0 \models \mathrm{ac}(G_0 \Rightarrow^* G_{i+1}) & \\ \Leftrightarrow & k_0 \models \mathrm{ac}(G_0 \Rightarrow^* G_i) \wedge \mathrm{L}(p(G_0 \Rightarrow^* G_i), \mathrm{ac}(G_i \Rightarrow G_{i+1})) & \text{Definition of ac} \\ \Leftrightarrow & k_0 \models \mathrm{ac}(G_0 \Rightarrow^* G_i) \wedge k_i \models \mathrm{ac}(G_i \Rightarrow G_{i+1}) & \text{Lemma 3.7} \\ \Leftrightarrow & k_0 \models \mathrm{ac}(G_0 \Rightarrow^* G_i) \wedge k_i \models \mathrm{Shift}(m_{i+1}, \mathrm{ac}_{L_{i+1}}) & \text{Definition of ac} \\ \Leftrightarrow & k_0 \models \mathrm{ac}(G_0 \Rightarrow^* G_i) \wedge k_i \circ m_{i+1} \models \mathrm{ac}_{L_{i+1}} & \text{Lemma 3.5} \\ \Leftrightarrow & k_{j-1} \circ m_j \models \mathrm{ac}_{L_j} \text{ for } j = 1, \ldots, i+1 & \text{Ind hypothesis} \end{array}$$

□

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 4.4. (Extension Theorem with ACs)
Given an AC-disregarding transformation $t \colon G_0 \Rightarrow^* G_n$ with derived rule $\rho(t) = \langle p(t) = (G_0 \leftarrow D \rightarrow G_n), \mathrm{ac}(t) \rangle$ and an extension diagram (1), where $t' \colon G'_0 \Rightarrow G'_n$ satisfies the corresponding ACs,

$$\begin{array}{ccccc} B & \xrightarrow{b_0} & G_0 & \overset{*}{\underset{t}{\Longrightarrow}} & G_n \\ \downarrow & (2) & \downarrow{\scriptstyle k_0} & (1) & \downarrow{\scriptstyle k_n} \\ C & \longrightarrow & G'_0 & \overset{*}{\underset{t'}{\Longrightarrow}} & G'_n \end{array}$$

with an initial pushout (2) over $k_0 \in \mathcal{M}$, then we have the following, shown in the diagrams 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 $G'_0 \Rightarrow G'_n$ 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 $B \to C$.

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

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 [6], k_0 is boundary consistent with respect to $t \colon G_0 \Rightarrow^* G_n$, with the morphism $b \colon B \to D$ and items 2. and 3. are valid for plain rules. 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 $k_{i-1} \circ m_i \models \mathrm{ac}_{L_i}$ 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 $b \colon B \to D$. □

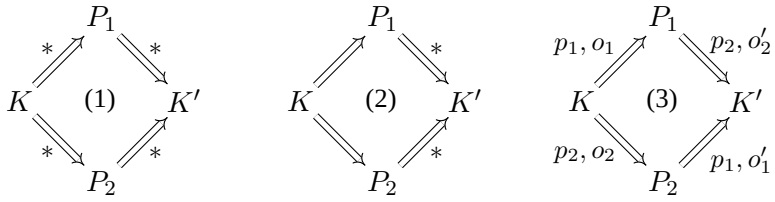
Remark 4.5. The Embedding and Extension Theorems are valid for t satisfying ACs and for t disregarding ACs. In both cases t' satisfies the corresponding ACs.

5. Local Confluence in Graph Transformation

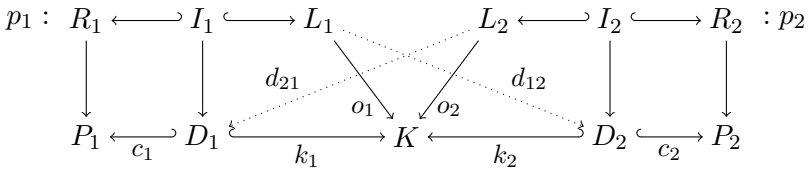
In this section, we review some results about the local confluence of graph transformation systems without application conditions. In this sense, we assume that the reader has a reasonably good knowledge of

the double-pushout approach to graph transformation. For more detail, the interested reader may consult e.g. [6].

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 [23].



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., [6]). In particular, the applications of two rules p_1 and p_2 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, this is the case 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 [26, 27], 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

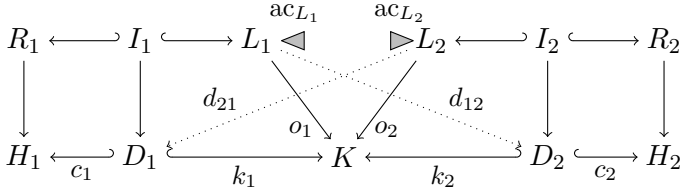
confluence of the critical pair would imply the confluence of $G_1 \Leftarrow G \Rightarrow G_2$. However, this is not true in general. For instance, if we try to apply in a larger context the rules in the derivations $P_1 \overset{*}{\Rightarrow} K' \overset{*}{\Leftarrow} P_2$ the gluing conditions 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 [27, 6] it is shown for plain rules that plain strict confluence of all critical pairs implies the local confluence of a graph transformation system. In the following sections we show how to extend these results to rules with application conditions.

6. Critical Pairs and Completeness

The use of application conditions in transformation rules complicates considerably the analysis of the confluence of a 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 6.3 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 6.1. (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 $c_1 \circ d_{21} \models \mathrm{ac}_{L_2}$.



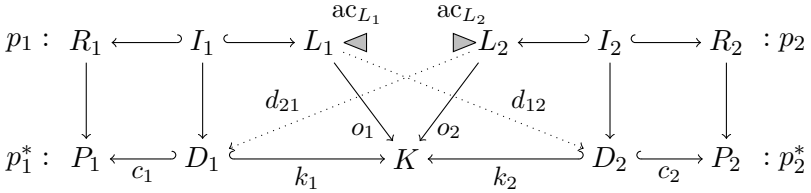
The intuition of the notion of *weak critical pair* is also quite simple. We know that all pairs of rule applications are potentially non confluent, even if they are parallel independent when disregarding the

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

Definition 6.2. (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 the pair (o_1, o_2) is in $\mathcal{E}'$.[5] Every weak critical pair induces 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 6.3. (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 Fig. 5 at the end of Section 7. 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 $o_1 = id_L$. The extension condition ac_K of this weak critical pair is shown in Fig. 4.

[5] In the category of graphs, $\mathcal{E}'$ is the class of pairs of jointly surjective morphisms.

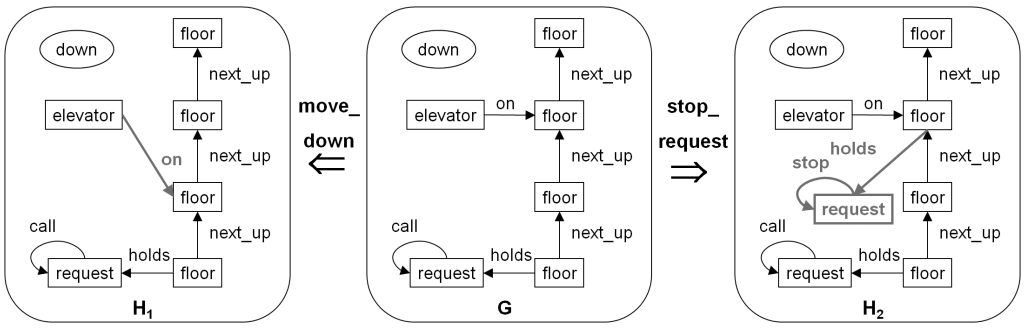


Figure 3. Parallel dependent pair of direct transformations with ACs of *Elevator*

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 the morphism o_2 (see Lemma 3.5). 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^*.

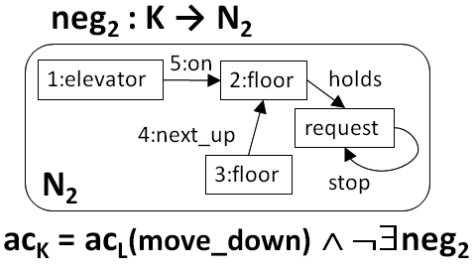


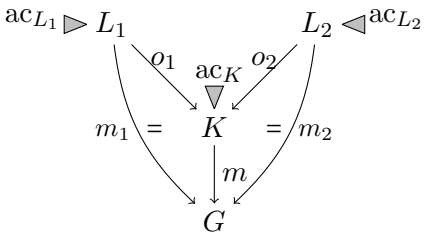
Figure 4. ac_K for critical pair of *Elevator*

We can prove that for each pair of parallel dependent transformations with ACs, there exists a weak critical pair.

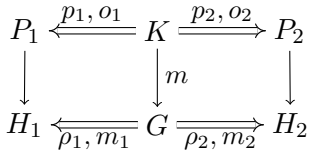
Lemma 6.4. (completeness of weak critical pairs with ACs)
For each pair of parallel dependent direct transformations $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ with ACs, there is a weak critical pair $P_1 \Leftarrow_{\rho_1,o_1} K \Rightarrow_{\rho_2,o_2} P_2$ with induced application conditions ac_K and ac_K^* and extension diagrams as shown below with $m \in \mathcal{M}$ and $m \models \mathrm{ac}_K \wedge \mathrm{ac}_K^*$.

$$\begin{array}{ccccc} P_1 & \overset{\rho_1,o_1}{\Longleftarrow} & K & \overset{\rho_2,o_2}{\Longrightarrow} & P_2 \\ \downarrow & & \downarrow m & & \downarrow \\ H_1 & \underset{\rho_1,m_1}{\Longleftarrow} & G & \underset{\rho_2,m_2}{\Longrightarrow} & H_2 \end{array}$$

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 $m_2 = m \circ o_2$.



Now we apply the Restriction Theorem without ACs [6], which states that a transformation $G \Rightarrow_{p_i,m_i} H_i$ can be restricted to a transformation $K \Rightarrow_{p_i,o_i} P_i$ via an $\mathcal{M}$-morphism $m : K \to G$ if there is a match $o_i : L_i \to K$ with $m \circ o_i = m_i$. This means that 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:



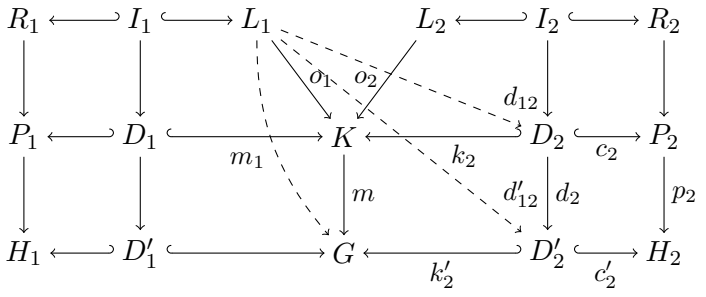
By assumption, $m_i \models \mathrm{ac}_{L_i}$ for $i = 1, 2$. By Lemma 3.5, we have

$$m_i = m \circ o_i \models \mathrm{ac}_{L_i} \Leftrightarrow m \models \mathrm{Shift}(o_i, \mathrm{ac}_{L_i})$$

for $i = 1, 2$. 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, $\mathrm{ac}^*_{K,d_{12}} = \mathrm{false}$, 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 Lemmas 3.5 and 3.7, $c'_2 \circ d'_{12} = p_2 \circ c_2 \circ d_{12} \models \mathrm{ac}_{L_1} \Leftrightarrow p_2 \models \mathrm{Shift}(c_2 \circ d_{12}, \mathrm{ac}_{L_1}) \Leftrightarrow m \models \mathrm{L}(p^*_2, \mathrm{Shift}(c_2 \circ d_{12}, \mathrm{ac}_{L_1})) = \mathrm{ac}^*_{K,d_{12}}$. 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, $m \models \mathrm{ac}^*_K$. □

It can be proven that the conflict-inducing AC ac^*_K is characteristic for parallel dependency with ACs.

Lemma 6.5. (characterization of parallel dependency with ACs)
Given a pair of transformations with ACs $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ 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:

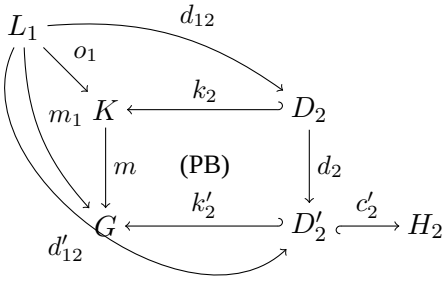
$$H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2 \text{ is parallel dependent} \iff m \models \mathrm{ac}^*_K.$$

Proof:
"⇒". This follows from the 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. Similarly, 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. □

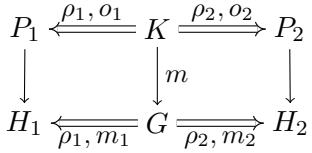
From the definition of ac_K and ac^*_K it follows that weak critical pairs without an extension m satisfying ac_K and ac^*_K are useless for checking local confluence, because no extension of parallel dependent transformation with ACs would exist anyway. Our notion of critical pair makes use of this 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 6.6. (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 a morphism $m\colon K \to G \in \mathcal{M}$ 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, i.e. m_i has a pushout complement with respect to p_i.

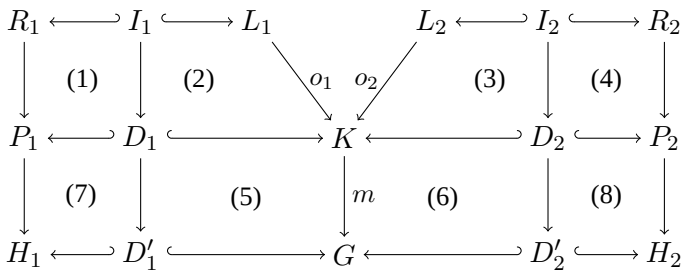
Remark 6.7. The condition "m_i has a pushout complement with respect to p_i" in the critical pair definition (Def. 6.6) is equivalent to the condition "m has a pushout complement with respect to the rule $p^*_i = \langle K \hookleftarrow D_i \to P_i\rangle$" where p^*_i is the derived rule of $K \Rightarrow_{p_i,o_i} P_i$ ($i = 1, 2$).

We can prove that a weak critical pair is also a critical pair with ACs if and only if it can be extended to a pair of parallel dependent transformations with ACs.

Lemma 6.8. (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 if and only if there is a parallel dependent pair $H_1 \Leftarrow_{\rho_1,m_1} G \Rightarrow_{\rho_2,m_2} H_2$ and morphism $m\colon K \to G \in \mathcal{M}$ with extension diagram



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 $m\colon K \to G \in \mathcal{M}$ with $m \models \mathrm{ac}_K \wedge \mathrm{ac}^*_K$ such that m has a pushout complement with respect to the rules p^*_i (see Remark 6.7) 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 Lemma 3.5, $m_i = m \circ o_i \models \mathrm{ac}_{L_i} \Leftrightarrow m \models \mathrm{Shift}(o_i, \mathrm{ac}_{L_i})$ $(i = 1, 2)$. 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 6.5, this sequence is parallel dependent using $m \models \mathrm{ac}^*_K$.
"$\Leftarrow$": Given condition 2, the weak critical pair is a critical pair, because we have $m\colon K \to G \in \mathcal{M}$ 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 6.5 using parallel dependence of the given pair. □

Remark 6.9. In Lemma 6.8 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.

Note that this new critical pair notion is different from the one for rules with NACs [21]. 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 [21]. 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 [21] 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.

Theorem 6.10. (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 $m \in \mathcal{M}$ and $m \models \mathrm{ac}_K \wedge \mathrm{ac}^*_K$.

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

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

Proof:
The proof is based on the completeness of critical pairs without ACs in [6]. 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 3.5) 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 3.5 and 3.7). This proof is shown in Lemma 6.4. 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 6.8) , where the proof of Lemma 6.8 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 6.5).

Conversely, it follows from the critical pair definition, Lemma 6.8 and Lemma 6.5 that each critical pair can be extended to a pair of parallel dependent transformations with ACs. □

Example 6.11. (critical pair)
Because of Theorem 6.10, the weak critical pair described in Example 6.3 is a critical pair. In particular, the parallel dependent transformations depicted in Fig. 3 satisfy ac_K and ac^*_K.

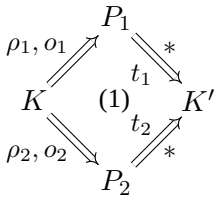
Example 6.12. (critical pair with non-trivial ac^*_K)
As noticed already in Example 6.3, the conflict-inducing condition ac^*_K is equivalent to true for each monomorphic extension morphism of the critical pair, leading to conflicting transformations for each extension of the critical pair. Now we illustrate by a toy example that, in general, ac^*_K may also be non-trivial. 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 $m : K \rightarrow G$ 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 $m : K \rightarrow G$ 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.

7. 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 5, even in the case of plain graph transformation rules this is not sufficient to show local confluence [26, 27]. 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 $m : K \rightarrow G$ 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 $\overline{t} : K \overset{*}{\Rightarrow} K'$ we may compute (see Def. 4.1) a derived application condition $\mathrm{ac}(\overline{t})$ which is equivalent to all the AC's in the transformation, in the sense that for every extension $m : K \rightarrow G$ we have that m satisfies $\mathrm{ac}(\overline{t})$ 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 $\mathrm{ac}(\overline{t})$ and $\mathrm{ac}(\overline{t'})$, where $\overline{t} : K \Rightarrow P_1 \overset{*}{\Rightarrow} K' \overset{*}{\Leftarrow} P_2 \Leftarrow K : \overline{t'}$, 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 $m : K \rightarrow G$ satisfies ac_K and ac^*_K. AC-compatibility plus plain strict confluence is called *strict AC-confluence*.

Definition 7.1. (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 the following holds:
The pair is *plain strictly confluent*, i.e. strictly confluent in the sense of [6] with AC-disregarding transformations t_1 and t_2, such that the extended AC-disregarding transformations $\overline{t}_i = K \Rightarrow_{p_i,o_i} P_i \Rightarrow^*_{t_i} K'$

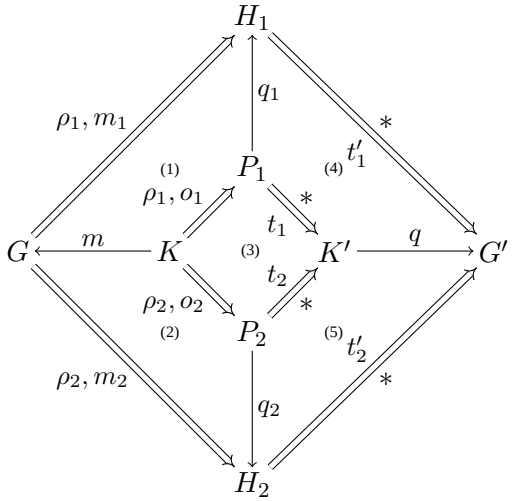
($i = 1, 2$) with derived ACs $\mathrm{ac}(\overline{t}_i)$ on K are *AC-compatible*, i.e. $\mathrm{ac}_K \wedge \mathrm{ac}^*_K \Rightarrow \mathrm{ac}(\overline{t}_1) \wedge \mathrm{ac}(\overline{t}_2)$.



Remark 7.2. 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 $\mathrm{ac}(\overline{t}_1) \wedge \mathrm{ac}(\overline{t}_2)$ and in the Local Confluence Theorem to show that the extended transformations $\overline{t}'_1$ and $\overline{t}'_2$ of $\overline{t}_1$ and $\overline{t}_2$ along $m : K \to G$ satisfy the ACs.

Theorem 7.3. (Local Confluence with ACs)
A transformation system with ACs 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 [7, **?**]. By the Local Confluence Theorem without ACs [6] and the Completeness Theorem 6.10 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 morphism $m \models (\mathrm{ac}_K \wedge \mathrm{ac}^*_K) \in \mathcal{M}$. 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 $\overline{t}'_i \colon G \Rightarrow_{\rho_i,m_i} H_i \Rightarrow^*_{t'_i} G'$ for $i = 1, 2$ satisfies the corresponding ACs. For this purpose, we have to extend the Embedding Theorem without ACs [6] to the case with ACs. This is shown as Theorem 4.3 based on Lemma 3.5 and 3.7. 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. 4.1). In our context, AC-consistency means that we have to show $m \models \mathrm{ac}(\overline{t}_1)$ and $m \models \mathrm{ac}(\overline{t}_2)$, where $\overline{t}_i : K \Rightarrow_{\rho_i,o_i} P_i \Rightarrow^*_{t_i} K'$, $(i = 1, 2)$. Note that AC-satisfaction of $\overline{t}_1$ and $\overline{t}_2$ is not required, but that of $\overline{t}'_1$ and $\overline{t}'_2$ is a consequence. Now, by AC-confluence, especially, AC-compatibility, we have

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

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

Remark 7.4. As proven in [17], 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 [24, 25], techniques are presented to tackle the satisfiability and the deduction problems in practice. Obviously, this kind of techniques would be important in our context to compute critical pairs. Nevertheless, it must be taken into account that, as shown in [27], checking local confluence for terminating graph transformation systems is undecidable, even in the case of rules without ACs.

Example 7.5. (Local Confluence with ACs)
In order to apply Theorem 7.3 we show that the critical pair in Example 6.3 and 6.11 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 $\overline{t}_1 : K \Rightarrow P_1 = K'$ and $\overline{t}_2 : K \Rightarrow P_2 \Rightarrow P_3 \Rightarrow K'$. At first, we can conclude that $\mathrm{ac}_K \Rightarrow \mathrm{ac}(\overline{t}_2)$ (see ac_K in Fig. 4), because $\mathrm{ac}(\overline{t}_2)$ is, in particular, equivalent to ac_K. Therefore, also $\mathrm{ac}_K \wedge \mathrm{ac}^*_K \Rightarrow \mathrm{ac}(\overline{t}_1) \wedge \mathrm{ac}(\overline{t}_2)$, since $\mathrm{ac}(\overline{t}_1) = \mathrm{Shift}(o_1, \mathrm{ac}_{L_1})$, 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 6.3 ac^*_K is equivalent to true. By Theorem 7.3 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.

8. Conclusion, Related and Future Work

In this paper, together with the results in part 1 of our paper [7] on parallelism, concurrency, and amalgamation we have extended the main results for graph transformation and adhesive HLR systems in [6] to the case of rules with nested application conditions. In particular, we have presented a new method for proving local confluence for transformation systems with nested application conditions, short ACs, which requires to generalize also the embedding theorem to the case with ACs. Nested application conditions provide a considerable increase of expressive power with respect to the NACs that were used up to now. Moreover, all the results presented apply not only to the categories of graphs or typed graphs, but to arbitrary $\mathcal{M}$-adhesive 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

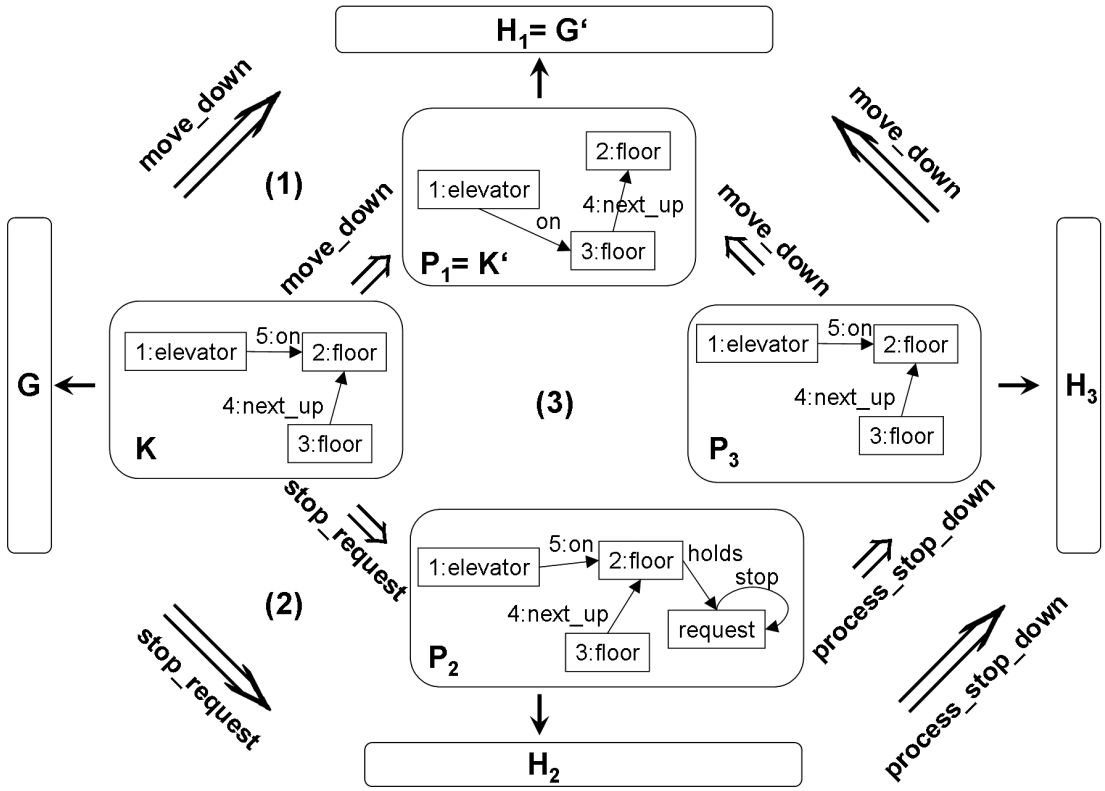


Figure 5. Strictly AC-confluent critical pair of *Elevator*

not just a generalization of the critical pair method introduced in [21] to prove local confluence of GTSs with NACs. There, 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 definition of critical pairs had to be found. As main results we have shown an Embedding and Extension, Completeness and Local Confluence Theorem for the case with ACs.

Nested graph conditions [16] were influenced by the work of [28]. In [17], nested graph conditions are shown to be expressively equivalent to first-order graph formulas [3] where one part of the proof is similar to the translation between first-order logic and predicates on edge-labeled graphs with single edges [28].

The use of critical pairs to check confluence in GTSs was introduced in [26]. On the other hand, graph transformation with the important special kind of negative application conditions was introduced in [14], 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 [4]. 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 [21]. In [13], the results of this paper

on local confluence for rules with nested application conditions are used to prove confluence of a model transformation from statecharts to Petri nets.

In this paper, we have considered DPO rules with left and right morphism in $\mathcal{M}$. More generally, we could consider rules with arbitrary right morphism [15, 11, 2]. But up to now, there is no theory for DPO rules with arbitrary right morphism and nested application conditions. Beside the DPO approach, one may also consider the single-pushout (SPO) approach introduced in [22], which is a generalization of the DPO framework where only one morphism defines the rule, which may be partial to allow deletion. In [14], SPO rules with negative application conditions are considered and the Local Confluence and Parallelism Theorems are shown. As far as we know, a theory on SPO rules with nested application conditions is missing. Both extensions of the DPO approach have their advantages, but also their drawbacks. While it is possible to define more powerful rules which may be helpful in practice [12], less theory is available which weakens the expressiveness of the results. Since most theory has been developed for the DPO approach and we rely heavily on this theory, we have chosen the DPO approach in this paper. Developing an analogous theory for the SPO approach or non-injective rule morphisms is future work.

In the AGG system [30], it is possible to specify nested application conditions. 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 [24, 25] 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] Baldan, P., Gadducci, F., Sobocinski, P.: Adhesivity Is Not Enough: Local Church-Rosser Revisited, *Mathematical Foundations of Computer Science (MFCS 2011)*, 6907, 2011, 48–59.

[3] 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*, vol. 1, World Scientific, 1997, 313–400.

[4] 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.

[5] Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamental Theory of Typed Attributed Graph Transformation based on Adhesive HLR-Categories, *Fundamenta Informaticae*, **74(1)**, 2006, 31–61.

[6] Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: *Fundamentals of Algebraic Graph Transformation,* EATCS Monographs of Theoretical Computer Science, Springer, 2006.

[7] Ehrig, H., Golas, U., Habel, A., Lambers, L., Orejas; F.: $\mathcal{M}$-Adhesive Transformation Systems with Nested Application Conditions.Part 1: Parallelism, Concurrency and Amalgamation, Mathematical Structures in Computer Science, 2012. To appear.

[8] Ehrig, H., Golas, U., Hermann, F.: Categorical Frameworks for Graph Transformation and HLR Systems based on the DPO Approach, *Bulletin of the EATCS*, **102**, 2010, 111–121.

[9] 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.

[10] Ehrig, H., Habel, A., Lambers, L., Orejas, F., Golas, U.: Local Confluence for Rules with Nested Application Conditions, *Graph Transformations - 5th International Conference, ICGT 2010*, 6372, Springer-Verlag, 2010, 330–345.

[11] Ehrig, H., Habel, A., Parisi-Presicce, F.: Basic Results for Two Types of High-Level Replacement Systems, *GETGRATS*, 51, 2002.

[12] Geiß, R., Batz, G. V., Grund, D., Hack, S., Szalkowski, A.: GrGen: A fast SPO-based graph rewriting tool, *Graph Transformations (ICGT 2006)*, 4178, Springer, 2006, 383–397.

[13] Golas, U.: *Analysis and Correctness of Algebraic Graph and Model Transformations*, Ph.D. Thesis, Technische Universität Berlin, Vieweg + Teubner, 2011.

[14] Habel, A., Heckel, R., Taentzer, G.: Graph Grammars with Negative Application Conditions, *Fundamenta Informaticae*, **26**, 1996, 287–313.

[15] Habel, A., Müller, J., Plump, D.: Double-Pushout Graph Transformation Revisited, *Mathematical Structures in Computer Science*, **11**(5), 2001, 637–688.

[16] Habel, A., Pennemann, K.-H.: Nested Constraints and Application Conditions for High-Level Structures, *Formal Methods in Software and System Modeling*, 3393, Springer, 2005, 293–308.

[17] Habel, A., Pennemann, K.-H.: Correctness of High-Level Transformation Systems Relative to Nested Conditions, *Mathematical Structures in Computer Science*, **19**, 2009, 245–296.

[18] Knuth, N. E., Bendix, P. B.: Simple Word Problems in Universal Algebra, *In J. Leech, editor*, Computational Problems in Abstract Algebra, 1970, 263–297.

[19] 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.

[20] Lack, S., Sobociński, P.: Adhesive and Quasiadhesive Categories, *Theoretical Informatics and Application*, **39(2)**, 2005, 511–546.

[21] Lambers, L.: *Certifying Rule-Based Models using Graph Transformation*, Ph.D. Thesis, Technische Universität Berlin, 2010.

[22] Löwe, M.: Algebraic Approach to Single-Pushout Graph Transformation, *Theoretical Computer Science*, **109**, 1993, 181–224.

[23] Newman, M. H. A.: On theories with a combinatorial definition of "equivalence", *Annals of Mathematics*, **43**(2), 1942, 223–243.

[24] Orejas, F., Ehrig, H., Prange, U.: A Logic of Graph Constraints, *Fundamental Approaches to Software Engineering (FASE 2008)*, 4961, 2008, 179–198.

[25] Pennemann, K.-H.: *Development of Correct Graph Transformation Systems*, Ph.D. Thesis, Universität Oldenburg, 2009.

[26] Plump, D.: Hypergraph Rewriting: Critical Pairs and Undecidability of Confluence, in: *Term Graph Rewriting: Theory and Practice*, John Wiley, 1993, 201–213.

[27] Plump, D.: Confluence of Graph Transformation Revisited, *Processes, Terms and Cycles: Steps on the Road to Infinity: Essays Dedicated to Jan Willem Klop on the Occasion of His 60th Birthday*, 3838, Springer, 2005, 280–308.

[28] Rensink, A.: Representing first-order logic by graphs, *Graph Transformations (ICGT'04)*, 3256, Springer, 2004, 319–335.

[29] Rozenberg, G., Ed.: *Handbook of Graph Grammars and Computing by Graph Transformation*, vol. 1: Foundations, World Scientific, 1997.

[30] TFS-group, TU Berlin: *AGG, http://tfs.cs.tu-berlin.de/agg*.

A. $\mathcal{M}$-adhesive Categories

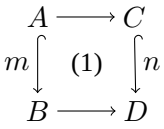
Adhesive categories have been introduced in [20] and generalized to weak adhesive HLR categories in [6] as a categorical framework for various kinds of graphs and net transformation systems. Here, we use $\mathcal{M}$-adhesive categories [8], a slight variation of weak adhesive HLR categories.

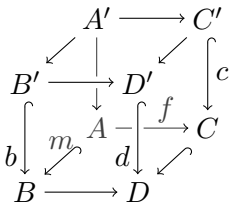
Definition A.1. ($\mathcal{M}$-adhesive category)
An *$\mathcal{M}$-adhesive category* $(\mathcal{C}, \mathcal{M})$ consists of a category $\mathcal{C}$ and a class $\mathcal{M}$ of monomorphisms in $\mathcal{C}$ such that the following properties hold:

1. $\mathcal{M}$ is closed under isomorphisms, composition, and decomposition, i.e., for morphisms f isomorphism implies $f \in \mathcal{M}$; $f, g \in \mathcal{M}$ implies $g \circ f \in \mathcal{M}$; and $g \circ f \in \mathcal{M}$, $g \in \mathcal{M}$ implies $f \in \mathcal{M}$.

2. $\mathcal{C}$ has pushouts and pullbacks along $\mathcal{M}$-morphisms, i.e. pushouts and pullbacks, where at least one of the given morphisms is in $\mathcal{M}$, and $\mathcal{M}$-morphisms are closed under pushouts and pullbacks, i.e. given a pushout (1) as in the figure below, $m \in \mathcal{M}$ implies $n \in \mathcal{M}$ and, given a pullback (1), $n \in \mathcal{M}$ implies $m \in \mathcal{M}$.

3. Pushouts in $\mathcal{C}$ along $\mathcal{M}$-morphisms are vertical weak van Kampen (VK) squares, short $\mathcal{M}$-VK squares, i.e. for any commutative cube in $\mathcal{C}$ where we have a pushout with $m \in \mathcal{M}$ in the bottom, $b, c, d \in \mathcal{M}$ and the back faces are pullbacks, it holds: the top is pushout iff the front faces are pullbacks.





Remark A.2. Note that the decomposition property of $\mathcal{M}$ follows from the pullback stability and is only listed for historical reasons.

$\mathcal{M}$-adhesive categories have a number of nice properties, called HLR properties [9].

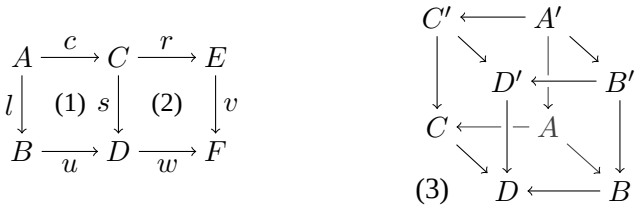
Fact A.3. (properties of $\mathcal{M}$-adhesive categories [6, 8])
For an $\mathcal{M}$-adhesive category $\langle \mathcal{C}, \mathcal{M} \rangle$, the following properties hold:

1. Pushouts along $\mathcal{M}$-morphisms are pullbacks.

2. $\mathcal{M}$ pushout-pullback decomposition: If the diagram (1)+(2) in the figure below is a pushout, (2) a pullback, $w \in \mathcal{M}$ and ($l \in \mathcal{M}$ or $c \in \mathcal{M}$), then (1) and (2) are pushouts and also pullbacks.

3. Cube pushout-pullback decomposition: Given the commutative cube (3) in the figure below, where all morphisms in the top and the bottom are in $\mathcal{M}$, 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.



4. Uniqueness of pushout complements. Given an $\mathcal{M}$-morphism $c\colon A \to C$ 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.

In order to prove the main results for $\mathcal{M}$-adhesive systems based on $\mathcal{M}$-adhesive categories we use some additional properties [6, 8].

Definition A.4. (additional properties)
Let $\langle \mathcal{C}, \mathcal{M} \rangle$ be an $\mathcal{M}$-adhesive category.

1. $\langle \mathcal{C}, \mathcal{M} \rangle$ has *initial pushouts over $\mathcal{M}$-morphisms*, if for every morphism $f\colon A \hookrightarrow A'$ with $f \in \mathcal{M}$ there exists an initial pushout over f. *Initiality* of a pushout (1) over f means that for every pushout (3) with $b' \in \mathcal{M}$ there exist unique morphisms $b^*, c^* \in \mathcal{M}$ such that $b' \circ b^* = b$, $c' \circ c^* = c$ and (2) is a pushout. In this case, $b\colon B \to A \in \mathcal{M}$ is called the *boundary* over f, B is called the *boundary* object and C the *context* with respect to f.

2. $\langle \mathcal{C}, \mathcal{M} \rangle$ has a *unique $\mathcal{E}'$-$\mathcal{M}$ pair factorization* for a given class of morphism pairs $\mathcal{E}'$ with the same codomain and a class of morphisms $\mathcal{M}$ 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 $m\colon K \hookrightarrow C$ with $(e_1, e_2) \in \mathcal{E}'$ and $m \in \mathcal{M}$ such that $m \circ e_1 = f_1$ and $m \circ e_2 = f_2$.