Under consideration for publication in Math. Struct. in Comp. Science

$\mathcal{M}$-Adhesive Transformation Systems with Nested Application Conditions. Part 1: Parallelism, Concurrency and Amalgamation

HARTMUT EHRIG[1], ULRIKE GOLAS[2], ANNEGRET HABEL[3], LEEN LAMBERS[4†], and FERNANDO OREJAS[4]

[1] *Technische Universität Berlin, Germany, ehrig@cs.tu-berlin.de.*
[2] *Konrad-Zuse-Zentrum für Informationstechnik Berlin, Germany, golas@zib.de.*
[3] *Universität Oldenburg, Oldenburg, Germany, habel@informatik.uni-oldenburg.de.*
[4] *Hasso-Plattner-Institut, Universität Potsdam, Germany, leen.lambers@hpi.uni-potsdam.de.*
[5] *Universitat Politècnica de Catalunya, Spain, orejas@lsi.upc.edu.*

Received September 2011

Nested application conditions are important for several application domains and they generalize the well-known negative application conditions. We present Local Church-Rosser, Parallelism, Concurrency, and Amalgamation Theorems for rules with nested application conditions in the framework of $\mathcal{M}$-adhesive categories, where $\mathcal{M}$-adhesive categories are slightly more general than weak adhesive high-level replacement categories. Most of the proofs are based on the corresponding statements for rules without application conditions and two Shift-Lemmas, saying that nested application conditions can be shifted over morphisms and rules.

Keywords: $\mathcal{M}$-adhesive categories, nested application conditions, local Chuch-Rosser, parallelism, concurrency, amalgamation.

Contents

† The work of this author was 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`.

1. Introduction

Standard graph transformation systems have been studied extensively and applied to several areas of computer science (Rozenberg 1997; Ehrig et al. 1999a; Ehrig et al. 1999b). To cope with the different varieties of graphical structures, they were, first, generalized to high-level replacement (HLR) systems (Ehrig et al. 1991) and then, based on the notion of adhesive categories (Lack and Sobocinski 2005), to weak adhesive HLR systems in (Ehrig et al. 2006a; Ehrig et al. 2006b) and recently to $\mathcal{M}$-adhesive systems (Ehrig et al. 2010a)[†]. There is a proper hierarchy of categories: graph ⇒ high-level ⇒ weak adh(esive) HLR ⇒ $\mathcal{M}$-adhesive; categories that show that the implications are proper are given in (Ehrig et al. 2006b; Ehrig et al. 2010a). Examples of $\mathcal{M}$-adhesive categories are given in Figure 1.

	category	class $\mathcal{M}$
graph	graphs	injective
high-level	hypergraphs	injective
	algebraic specifications	injective & strict
weak adh HLR	typed attributed graphs	injective with isomorphism on the data part
	place/transition nets	monomorphisms
$\mathcal{M}$-adhesive	list sets	monomorphisms

Figure 1. $\mathcal{M}$-adhesive categories

Originally, application conditions (ACs), as defined in (Ehrig and Habel 1986), were very simple. They were restricted for specifying that a certain graph should not include the match of the rule. For this reason, they were called Negative Application Conditions (Habel et al. 1996). This kind of conditions are useful in many cases, but they are also too restrictive for some other cases. As a consequence, they were generalized to nested application conditions in (Habel and Pennemann 2009). Nested application conditions are shown to be expressively equivalent to first-order graph formulas (Courcelle 1997), where one part of the proof is similar to the translation between first-order logic and predicates on edge-labeled graphs with single edges in (Rensink 2004). There is a proper hierarchy of types of application conditions: no ACs ⇒ negative ACs ⇒ nested ACs. Examples of application domains where nested application conditions are used are given in Figure 2. This means that even if nested application conditions do not add any difficulty (undecidability), these results show that, in principle, the expressive power of nested application conditions is not smaller than the expressive power of conditions in term rewriting. However, given their different nature, they are difficult to really compare.

For (graph) transformation systems, there are a number of results in the literature, known as Local Church-Rosser, Parallelism, Concurrency, and Amalgamation Theorems. Informal descriptions of the results with some application areas are given in Figure 3.

[†] An *$\mathcal{M}$-adhesive system* consists of an $\mathcal{M}$-adhesive category and a set of rules over the category.

	applications with ACs
no	library system (Ehrig and Kreowski 1980) telephone system (Reibeiro 1996) client-server system (Corradini et al. 1997)
negative	library system (Ehrig et al. 2010b) elevator control (Habel et al. 1996; Lambers 2010) railroad control (Pennemann 2009)
nested	access control, mobile communication, carplatoon maneuver protocol (Pennemann 2009) model transformation (Golas 2011)

Figure 2. Applications domains where nested application conditions are used

theorem	informal description	application areas
Local Church-Rosser	*Parallel independent* transformations applied to the same object can be applied in arbitrary order, leading to the same result.	data base systems (Ehrig and Kreowski 1980), process control (Mahr and Wilharm 1982), algebraic specifications (Parisi-Presicce 1989) distributed information systems (Heckel et al. 2002)
Parallelism	*Parallel independent* transformations can be parallelized to a single transformation via the *parallel rule*. *Sequentially independent* transformations can be reduced to a single transformation via the *parallel rule*.	data base systems (Ehrig and Kreowski 1980), algebraic specifications (Parisi-Presicce 1989)
Concurrency	Allows *sequentially dependent transformations*: *Related* transformations can be reduced to a single transformation via the *concurrent rule*.	logic programming (Corradini et al. 1991)
Amalgamation	Allows *parallel dependent* transformations: *Amalgamable* transformations can be amalgamated to a single transformation via the *amalgamated rule*.	distributed systems (Castellani and Montanari 1983; Boehm et al. 1987; Degano and Montanari 1987; Taentzer et al. 1999; Golas 2011), model transformation (Biermann et al. 2010)

Figure 3. Informal descriptions of the results with some application areas

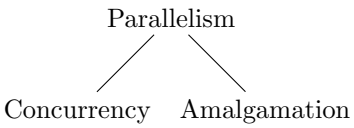
The Local Church-Rosser, Parallelism, and Concurrency Theorems were first presented for graph transformation systems on rules without application conditions in (Rosen 1975; Kreowski 1977a; Ehrig 1979; Ehrig and Rosen 1980; Ehrig et al. 1986) and were generalized to high-level replacement systems (Ehrig et al. 1991) and rules with negative application conditions (Lambers 2010). The Amalgamation Theorem is presented for graph transformation systems on rules without application conditions (Boehm et al. 1987; Corradini et al. 1997).

The aim of this paper is to show that results in the literature based on rules without (Ehrig et al. 2006b) or negative application conditions (Lambers 2010) can be generalized to nested application conditions (Habel and Pennemann 2009) in the framework of $\mathcal{M}$-adhesive transformation systems. In order to increase the expressive power of graph transformation systems it is important for several applications to consider not only negative application conditions, but also nested ones. The presentation of the main results in the categorical framework of $\mathcal{M}$-adhesive categories is also highly relevant: In this way the results are not only valid for classical graph transformation systems, but also for transformation systems based on typed and attributed graphs, hypergraphs, and different kinds of low and high-level Petri nets (Ehrig et al. 2006b).

We state the Local Church-Rosser, Parallelism, Concurrency and Amalgamation Theorems for $\mathcal{M}$-adhesive systems on rules with nested application conditions. The proofs of the Local Church-Rosser, Parallelism, and Concurrency Theorems are based on the corresponding theorems for $\mathcal{M}$-adhesive systems on rules without application conditions in (Ehrig et al. 2006b) and two Shift-Lemmata for nested application conditions (ACs), extending the ones in (Habel and Pennemann 2009), saying that application conditions can be shifted over morphisms and rules.

Theorem + Shift-Lemmata for ACs $\Rightarrow$ Theorem for rules with ACs

The Amalgamation Theorem for $\mathcal{M}$-adhesive systems on rules with nested application conditions can be considered a special case of a recent construction, called multi-amalgamation, studied in (Golas et al. 2012). The Concurrency and Amalgamation Theorems may be seen as two different generalizations of the Parallelism Theorem: in the first case, sequential independence and, in the second case, parallel independence is dropped.



The paper is organized as follows: In Section 2, we review the definition of an $\mathcal{M}$-adhesive category. In Section 3, we introduce rules with nested application conditions. In Section 4, we state and prove the Local Church-Rosser, Parallelism, and Concurrency Theorems. In Section 5, we define amalgamated rules and state the Amalgamation Theorem. Finally, in Section 7, an overview of the results of $\mathcal{M}$-adhesive transformation systems with nested application conditions is given. The concepts are illustrated by examples in the category of directed, labeled graphs with the class of all injective graph morphisms. To motivate the interest of rules with nested conditions, and to ease the un-

derstanding of the main concepts, an example describing a mutual exclusion algorithm closely following Dijkstra's work is presented.

The paper is a long version of the paper (Ehrig et al. 2010b). It contains a new section on amalgamation as well as a new illustrating example.

2. Graphs and High-level Structures

We recall the basic notions of directed, labeled graphs (Ehrig 1979; Corradini et al. 1997) and generalize them to high-level structures (Ehrig et al. 1991). The idea behind the consideration of high-level structures is to avoid similar investigations for similar structures such as Petri-nets and hypergraphs. We assume that the reader is familiar with the basic notions of graph transformation systems and the basic concepts of category theory; standard references are (Ehrig 1979; Arbib and Manes 1975; Adamek et al. 1990).

Directed, labeled graphs and graph morphisms are defined as follows.

Definition 1 (graphs and graph morphisms). Let $\mathrm{L} = (\mathrm{L_V}, \mathrm{L_E})$ be a fixed, finite label alphabet. A *graph* over L is a system $G = (\mathrm{V}_G, \mathrm{E}_G, \mathrm{s}_G, \mathrm{t}_G, \mathrm{l}_G, \mathrm{m}_G)$ consisting of two finite sets V_G and E_G of *nodes* (or *vertices*) and *edges*, *source* and *target functions* $\mathrm{s}_G, \mathrm{t}_G \colon \mathrm{E}_G \to \mathrm{V}_G$, and two *labeling functions* $\mathrm{l}_G \colon \mathrm{V}_G \to \mathrm{L_V}$ and $\mathrm{m}_G \colon \mathrm{E}_G \to \mathrm{L_E}$. A graph with an empty set of nodes is *empty* and denoted by $\emptyset$. A *graph morphism* $g \colon G \to H$ consists of two functions $g_\mathrm{V} \colon \mathrm{V}_G \to \mathrm{V}_H$ and $g_\mathrm{E} \colon \mathrm{E}_G \to \mathrm{E}_H$ that preserve sources, targets, and labels, that is, $\mathrm{s}_H \circ g_\mathrm{E} = g_\mathrm{V} \circ \mathrm{s}_G$, $\mathrm{t}_H \circ g_\mathrm{E} = g_\mathrm{V} \circ \mathrm{t}_G$, $\mathrm{l}_H \circ g_\mathrm{V} = \mathrm{l}_G$, and $\mathrm{m}_H \circ g_\mathrm{E} = \mathrm{m}_G$. A morphism g is *injective* (*surjective*) if g_V and g_E are injective (surjective), and an *isomorphism* if it is both injective and surjective. In the latter case G and H are *isomorphic*, which is denoted by $G \cong H$. The *composition* $h \circ g$ of g with a morphism $h \colon H \to M$ consists of the composed functions $h_\mathrm{V} \circ g_\mathrm{V}$ and $h_\mathrm{E} \circ g_\mathrm{E}$. The category having graphs as objects and graph morphisms as arrows is called Graphs.

Example 1. To illustrate our definitions and results in the following sections, we introduce an example describing a mutual exclusion algorithm closely following Dijkstra's work (Dijkstra 1965). First, we introduce the labels and underlying system models. In our system, we have an arbitrary number of processes `P` and resources `R`. To each resource, a turn variable `T` may be connected assigning this resource to a process. Each process may be `idle` or `active` and has a flag with possible values $0, 1, 2$, initially set to 0, which is graphically described by no flag at all at this process. Moreover, a label `crit` marks a process which has entered its critical section actually using the resource. Thus, the label alphabet used for our example is $\mathrm{L} = (\mathrm{L_V}, \mathrm{L_E})$ with $\mathrm{L_V} = \{\mathtt{P}, \mathtt{T}, \mathtt{R}, \mathtt{F1}, \mathtt{F2}\}$ and $\mathrm{L_E} = \{\mathtt{active}, \mathtt{idle}, \mathtt{crit}, \lambda\}$. In the left of Fig. 4, a system S is modeled containing a resource and two processes, one of them being `active` and one of them `idle`, where the active one is connected via an `F1`-flag and the other one via the turn variable to the resources. There is an injective graph morphism $g \colon S \to G$ extending S by another active process with a flag to the resource and an additional resource that has no turn variable and is thus disabled.

In drawings of graphs, nodes are drawn by circles and edges by arrows pointing from the source to the target node. Arbitrary graph morphisms are drawn by usual arrows

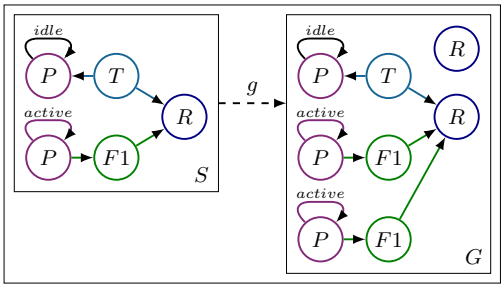


Figure 4. Example graph and graph morphism.

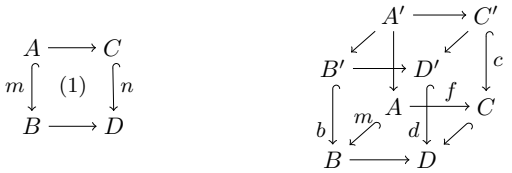
"$\to$"; the use of "$\hookrightarrow$" indicates an injective graph morphism but is only used if this should be pointed out explicitly. The actual mapping of the elements can be concluded by positions or is conveyed by indices, if necessary.

While the original double-pushout approach was defined on directed, labeled graphs (Ehrig et al. 1973; Ehrig 1979), it was later lifted to a categorical setting using a distinguished morphism class $\mathcal{M}$, with various instantiations. Especially adhesive and weak adhesive HLR categories are a suitable concept providing many of the required properties. Throughout the literature, various versions of adhesive (Lack and Sobocinski 2004), quasiadhesive (Lack and Sobocinski 2005), weak adhesive HLR (Ehrig et al. 2006b), partial map adhesive (Heindel 2010), and $\mathcal{M}$-adhesive (Ehrig et al. 2010a) exist. In adhesive categories, the class $\mathcal{M}$ of morphisms is fixed to all monomorphisms, while in quasiadhesive the class of all regular monomorphisms is considered. With slightly different requirements concerning the existence of pushouts and pullbacks along $\mathcal{M}$-morphisms and requirements of $\mathcal{M}$-morphisms in the van Kampen property, they are basically special weak adhesive HLR categories. In contrast, partial map adhesive categories are based on hereditary pushouts, which are pushouts that have to be preserved by the inclusion functor from the category $\mathcal{C}$ into the category of partial maps over $\mathcal{C}$. As shown in (Ehrig et al. 2010a), partial map adhesive categories are also $\mathcal{M}$-adhesive ones. Since all the main properties are valid in $\mathcal{M}$-adhesive categories, we have chosen to work with these in this paper.

Definition 2 ($\mathcal{M}$-adhesive category). A category $\mathcal{C}$ with a morphism class $\mathcal{M}$ is an *$\mathcal{M}$-adhesive category* if the following properties hold:

1 $\mathcal{M}$ is a class of monomorphisms closed under isomorphisms, composition, and decomposition, i.e., for morphisms f and g, 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 squares, short $\mathcal{M}$-VK squares, i.e. for any commutative cube in $\mathcal{C}$ where we have the 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.



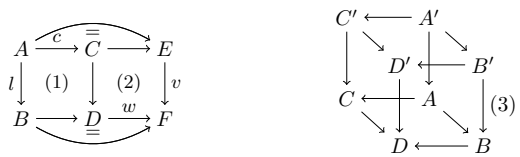
Remark 1. In contrast to a vertical weak van Kampen square, a horizontal one requires that $f \in \mathcal{M}$ instead of $b, c, d \in \mathcal{M}$. Both properties combined represent the weak van Kampen property as used in weak adhesive HLR categories (Ehrig et al. 2006b). Adhesive categories (Lack and Sobocinski 2005), which are a special case of $\mathcal{M}$-adhesive categories, are special cases of weak adhesive HLR categories in (Ehrig et al. 2006b), where, in addition, the class $\mathcal{M}$ is the class of all monomorphisms.

Fact 1 (Ehrig et al. 2006b). The category $\langle \text{Graphs}, \mathcal{M} \rangle$ with the class $\mathcal{M}$ of all injective graph morphisms is $\mathcal{M}$-adhesive. Moreover, several variants of graphs like typed and typed attributed graphs with a corresponding class $\mathcal{M}$ of injective morphisms form $\mathcal{M}$-adhesive categories. The category $\langle \text{PTNets}, \mathcal{M} \rangle$ of place/transition nets with the class $\mathcal{M}$ of all injective net morphisms and the category $\langle \text{Spec}, \mathcal{M}_{strict} \rangle$ of algebraic specifications with the class $\mathcal{M}_{strict}$ of all strict injective specification morphisms are $\mathcal{M}$-adhesive, but not adhesive.

$\mathcal{M}$-adhesive categories have a number of nice properties, known as HLR-properties (Ehrig et al. 1991).

Lemma 1 (HLR-properties). 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.

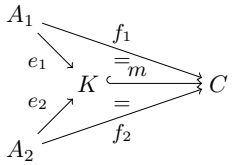


4 Uniqueness of pushout complements. Given morphisms $A \hookrightarrow C$ in $\mathcal{M}$ and $C \to D$, then there is, up to isomorphism, at most one B with $A \to B$ and $B \to D$ such that diagram (1) is a pushout.

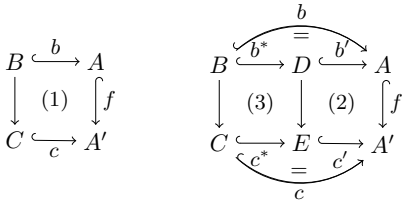
Proof. The proofs can be found in (Lack and Sobocinski 2005; Ehrig et al. 2006b). In (Ehrig et al. 2006c) the proofs of the HLR-properties are given for weak adhesive HLR-categories. But they are also valid for $\mathcal{M}$-adhesive categories, because horizontal weak VK-squares are not used in the proof. □

In order to prove the main results for $\mathcal{M}$-adhesive systems some additional HLR-requirements have to be required: unique $\mathcal{E}'$-$\mathcal{M}$ pair factorization, binary coproducts, and initial pushouts over $\mathcal{M}$-morphisms. $\mathcal{E}'$-$\mathcal{M}$ pair factorization is needed for the proof of all main results, but can also be obtained using classical $\mathcal{E}$-$\mathcal{M}$-factorization and binary coproducts. The latter ones are necessary and sufficient to define parallel rules in the Parallelism Theorem. Initial pushouts are needed for the proof of the Amalgamation Theorem. However, we cannot exclude that weaker versions of some of these HLR-requirements may be sufficient to show our main results or suitable variants of them.

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



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



Fact 2 (Ehrig et al. 2006b). The category $\langle \mathrm{Graphs}, \mathcal{M} \rangle$ has a unique $\mathcal{E}'$-$\mathcal{M}$ pair factorization (where $\mathcal{E}'$ is the class of pairs of jointly surjective graph morphisms), binary coproducts, and initial pushouts over $\mathcal{M}$-morphisms. Moreover, all examples in Fact 1 satisfy these requirements.

3. Rules with Application Conditions

We use the framework of $\mathcal{M}$-adhesive categories, introduce rules with application conditions for high-level structures such as graphs, Petri nets, (hyper)graphs, and algebraic specifications, and show how application conditions can be shifted over morphisms and rules.

General Assumption. We assume that $\langle \mathcal{C}, \mathcal{M} \rangle$ is an $\mathcal{M}$-adhesive category with $\mathcal{E}'$-$\mathcal{M}$ pair factorization (used in Shift-Lemma 2), binary coproducts (used in Definition 7), and initial pushouts over $\mathcal{M}$-morphisms (used in Theorem 4).

Remark 2. The category $\langle \text{Graphs}, \mathcal{M} \rangle$ satisfies the General Assumption. For simplicity, one may substitute *object* by *graph*, *morphism* by *graph morphism*, and $\mathcal{M}$-*adhesive* category $\langle \mathcal{C}, \mathcal{M} \rangle$ by category $\langle \text{Graphs}, \mathcal{M} \rangle$.

object	—	graph
morphism	—	graph morphism
$\langle \mathcal{C}, \mathcal{M} \rangle$	—	$\langle \text{Graphs}, \mathcal{M} \rangle$

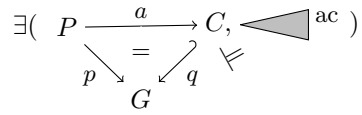
Application conditions, more concretely, nested application conditions, may be represented as a tree of morphisms equipped with logical symbols such as quantifiers and connectives.

Definition 4 (application conditions). An *application condition*, also called *nested application condition*, is inductively defined as follows:

1 For every object P, true is an application condition over P.
2 For every morphism $a\colon P \to C$ and every application condition ac over C, $\exists(a, \text{ac})$ is an application condition over P.
3 For application conditions ac, ac_i over P with $i \in I$ (for all index sets I), $\neg\text{ac}$ and $\wedge_{i \in I} \text{ac}_i$ are application conditions over P.

Satisfiability of application conditions is inductively defined as follows:

1 Every morphism satisfies true.
2 A morphism $p\colon P \to G$ satisfies $\exists(a, \text{ac})$ over P with $a\colon P \to C$ if there exists a morphism $q\colon C \to G$ in $\mathcal{M}$ such that $q \circ a = p$ and q satisfies ac.



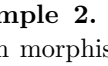
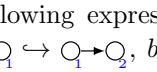
3 A morphism $p\colon P \to G$ satisfies $\neg\text{ac}$ over P if p does not satisfy ac, and p satisfies $\wedge_{i \in I} \text{ac}_i$ over P if p satisfies each ac_i $(i \in I)$.

We write $p \models \text{ac}$ to denote that the morphism p satisfies ac.

Two application conditions ac and ac$'$ over P are *equivalent*, denoted by $\text{ac} \equiv \text{ac}'$, if for all morphisms $p\colon P \to G$, $p \models \text{ac}$ if, and only if, $p \models \text{ac}'$.

Notation. $\exists a$ abbreviates $\exists(a, \text{true})$ and $\forall(a, \text{ac})$ abbreviates $\neg\exists(a, \neg\text{ac})$.

Remark 3. The concept of application conditions is introduced in (Ehrig and Habel 1986). Positive and negative application conditions, introduced in (Habel et al. 1996), correspond to nested application conditions of the form $\exists a$ and $\neg\exists a$, respectively. Negative application conditions are intensively investigated, e.g., in (Lambers 2010). Nested application conditions are introduced and intensively studied in (Habel and Pennemann 2009; Pennemann 2009). They generalize the corresponding notions in (Heckel and Wagner 1995; Koch et al. 2005; Ehrig et al. 2006a).

Example 2. The following expressions are application conditions based on injective graph morphisms a:  , b: , c:  , d: , and e: .

$\exists a$	There exists a proper edge outgoing from the image of 1.
$\neg\exists a$	There does not exist a proper edge outgoing from the image of 1.
$\exists(a, \neg\exists b)$	There exists a proper edge outgoing from the image of 1 without an edge in converse direction.
$\forall(a, \exists c)$	For every proper edge outgoing from the image of 1, the target has a loop.
$\exists(a, \forall(d, \exists e))$	There exists a proper edge outgoing from the image of 1 such that, for all edges outgoing from the target, the target has a loop.

The first application condition is positive, the second one negative, and the later ones are properly nested.

Rules are specified by a pair of $\mathcal{M}$-morphisms. For restricting the applicability of rules, the rules are equipped with a left and a right application condition. Such a rule is applicable with respect to a "match" morphism from the left-hand side of the rule to an object, if, and only if, the underlying plain rule is applicable, the match morphism satisfies the left application condition and the comatch morphism satisfies the right application condition.

Definition 5 (rules and transformations). A *plain rule* $p = \langle L \hookleftarrow K \hookrightarrow R \rangle$ consists of two $\mathcal{M}$-morphisms $K \hookrightarrow L$ and $K \hookrightarrow R$. A *rule* $\varrho = \langle p, \mathrm{ac}_L, \mathrm{ac}_R \rangle$ consists of a plain rule p and two application conditions ac_L and ac_R over L and R, respectively. L and R are called *left-* and *right-hand side* of p, respectively; ac_L and ac_R are called *left* and *right application condition* of ϱ, respectively.

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

A *direct transformation* consists of two pushouts (1) and (2) such that $m \models \mathrm{ac}_L$ and $m^* \models \mathrm{ac}_R$. We write $G \Rightarrow_{\varrho,m,m^*} H$ and say that $m\colon L \to G$ is the *match* of ϱ in G and $m^*\colon R \to H$ is the *comatch* of ϱ in H. We also write $G \Rightarrow_{\varrho,m} H$, $G \Rightarrow_{\varrho} H$, or $G \Rightarrow H$ to express that there are an m^*, m, m^*, or ϱ, m, m^*, respectively, such that $G \Rightarrow_{\varrho,m,m^*} H$.

A transformation $G \overset{*}{\Rightarrow} H$ means $G \cong H$ or a sequence of direct transformations $G = G_0 \Rightarrow G_1 \Rightarrow \ldots \Rightarrow G_n = H$.

Fact 3. In $\langle \text{Graphs}, \mathcal{M} \rangle$, the application of a rule $\varrho = \langle p, \text{ac}_L, \text{ac}_R \rangle$ to a graph G amounts to the following steps:

(1) Find a match $m\colon L \to G$ satisfying ac_L as well as the *gluing condition*:
Dangling condition: No edge in $G - m(L)$ is incident to a node in $m(L) - m(K)$.
Identification condition: For all distinct items $x, y \in L$, $m(x) = m(y)$ only if $x, y \in K$.
(This condition is understood to hold separately for nodes and edges.)
(2) Remove $m(L - K)$ from G, yielding a graph D and add $R - K$, yielding a graph H.
(3) Check whether the comatch $m^*\colon R \to H$ satisfies ac_R.

Example 3. Now we introduce the rules for the mutual exclusion algorithm. Its main aim is to ensure that at each time at most one process is using each resource. A different variant of this algorithm implemented by graph transformation can be found in (Ehrig et al. 2006b), where the lack of application conditions induces a much more complex model including more types or labels and also additional rules for handling a single resource. Using application conditions we can simplify the models and do not need additional edges representing the next executable step of the system while extending the context to an arbitrary number of resources.

Initially, each process is idle and for each resource the turn variable is connected to an arbitrary process, meaning that this process has the turn to use that resource. If a process P_1 wants to use some resource R it becomes active and points the flag F1 to R. If, in addition, it has the turn for R, it may proceed to use it, which is described by an F2-flag to the resource and a crit loop at the process. Otherwise, if the turn for R belongs to another process P', P must wait until P' is not flagging R. At this point the process may get the turn for R and start using it. When P has finished using R, the flag and the crit are removed, and the process is again idle. As an extension of this normal behaviour, a resource may be disabled, denoted by eliminating its turn variable, if there is no flag present for it. Moreover, a resource may be enabled again if all other resources have at least two requests waiting

The rules `setFlag`, `setTurn`, `enter`, and `exit` in Fig. 5 describe the standard behavior of the system. With `setFlag`, a process becomes active and sets its `F1`-flag to a resource. Note that this rule has a positive application condition requiring that the resource has a turn variable noting it as enabled. If a process has set an `F1`-flag to a resource and the turn variable of this resource points to another process, which has no flag to this resource, the turn variable can be assigned to the first process via `setTurn`. Here, the application condition forbids that the process that has the turn of the resource is already flagging that resource. With the rule enter, if a process has the turn of a resource R and it points to R with an F1-flag then the flag is replaced by an F2-flag and a loop `crit` is added to the process. When the process is finished, the rule `exit` is executed deleting the loop and the flag, and the process becomes idle again. Moreover, with the rules `disableR` and `enableR`, a resource can be disabled or enabled if the corresponding application conditions are fulfilled. In the figures, the application condition true is not drawn, while application

conditions $\mathrm{Q}(a, \mathrm{ac})$ with $\mathrm{Q} \in \{\exists, \neg\exists, \forall\}$ are drawn by the morphism $a\colon P \to C$, marked by $\mathrm{Q}a$, combined with a drawing of ac, and conjunctions of application conditions are marked by $\wedge$ between "the outgoing morphisms".

Note that we could easily have a rule `setFlag` without any application condition. In particular it is enough to include in the left-hand side of the rule the turn variable pointing to the resource R. However, the application condition $\forall(b_6, \exists c_6)$ depicted on the right of the rule `enableR` cannot be removed, even if it is also a positive application condition. In particular, this condition is nested twice, which is needed to specify that every other enabled resource has two waiting processes.

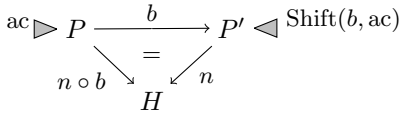
Consider the rule `setTurn` with the match m_1 depicted in the left of Fig. 6. Note that m_1 matches the two processes of the rule `setTurn` to the upper two processes in G such that m_1 satisfies the gluing condition as well as the application condition $\neg\exists a_2 \wedge \neg\exists b_2$ and leads to the direct transformation $G \Rightarrow_{\texttt{setTurn},m_1} H_1$ redirecting the turn variable from the idle process to one of the active ones as shown in Fig. 6. The graph H_1 is obtained from G by removing $m_1(L_1 - K_1)$ and adding $R_1 - K_1$. For the graph H_1, there is no direct transformation $H_1 \Rightarrow_{\texttt{setTurn},m'} H_2$ because any match $m'\colon L_1 \to H_1$ does not satisfy the application condition $\neg\exists a_2$.

It may be noticed that rules are completely symmetric, which means that a rule can be reversed obtaining its inverse rule.

Fact 4 (inverse rule). For every rule $\varrho = \langle p, \mathrm{ac}_L, \mathrm{ac}_R\rangle$ with $p = \langle L \hookleftarrow K \hookrightarrow R\rangle$, the rule $\varrho^{-1} = \langle p^{-1}, \mathrm{ac}_R, \mathrm{ac}_L\rangle$ with $p^{-1} = \langle R \hookleftarrow K \hookrightarrow L\rangle$ is the *inverse rule* of ϱ. For every direct transformation $G \Rightarrow_{\varrho,m,m^*} H$, there is a direct transformation $H \Rightarrow_{\varrho^{-1},m^*,m} G$ via the inverse rule.

Next we show two important technical results that are the key to prove the main results of this paper. The first one shows that application conditions can be shifted over morphisms. While the construction in (Habel and Pennemann 2009) allows a shift over a monomorphism and uses pushouts along $\mathcal{M}$-morphisms, the construction in this paper allows a shift over an arbitrary morphism and uses $\mathcal{E}'$-$\mathcal{M}$ pair factorizations.

Lemma 2 (shift of application conditions over morphisms). There is a Shift-construction such that, for each application condition ac over P and for each morphism $b\colon P \to P'$, Shift transforms ac via b into an application condition $\mathrm{Shift}(b, \mathrm{ac})$ over P' such that, for each morphism $n\colon P' \to H$, $n \circ b \models \mathrm{ac} \iff n \models \mathrm{Shift}(b, \mathrm{ac})$.



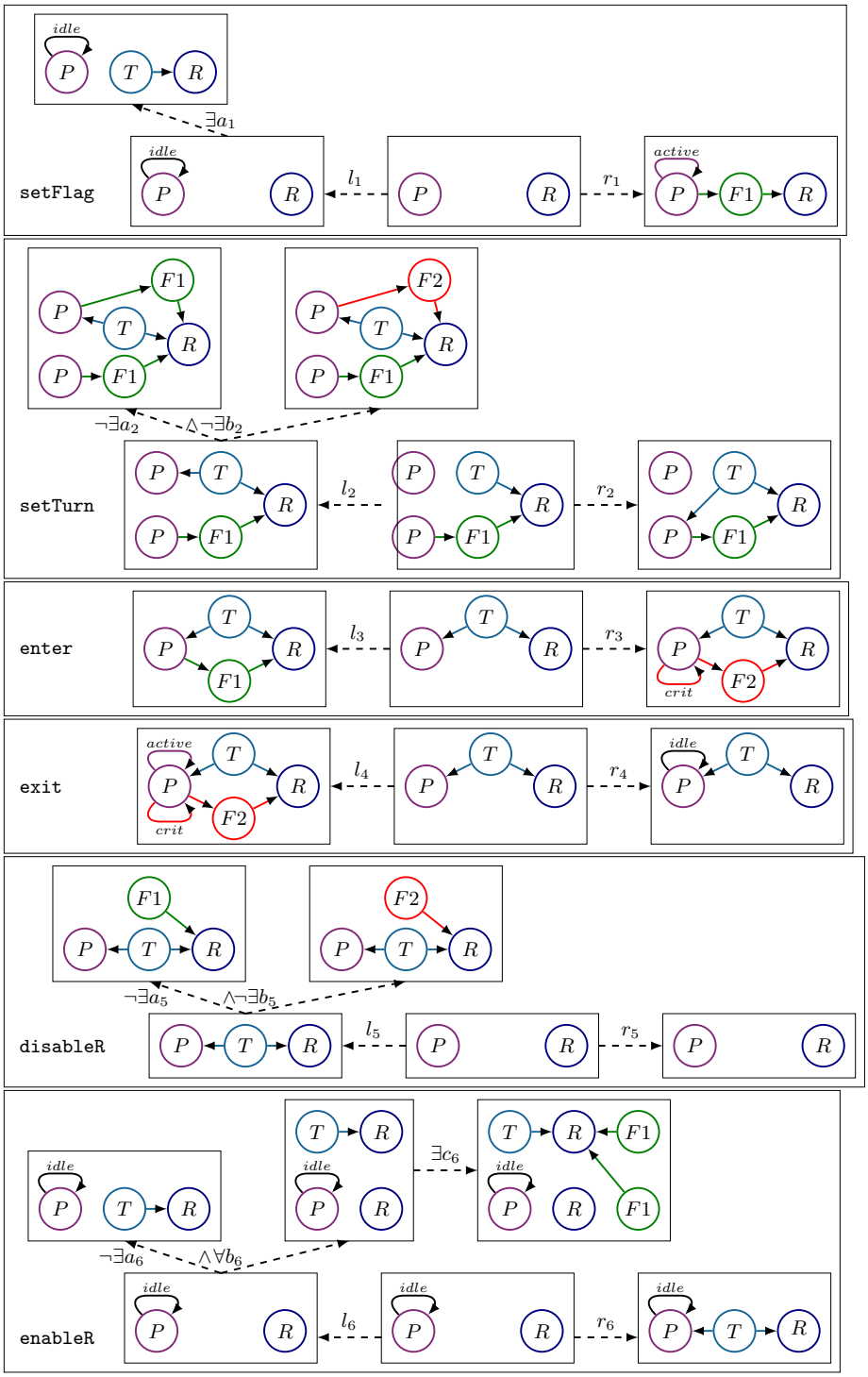


Figure 5. The rules for the mutual exclusion algorithm

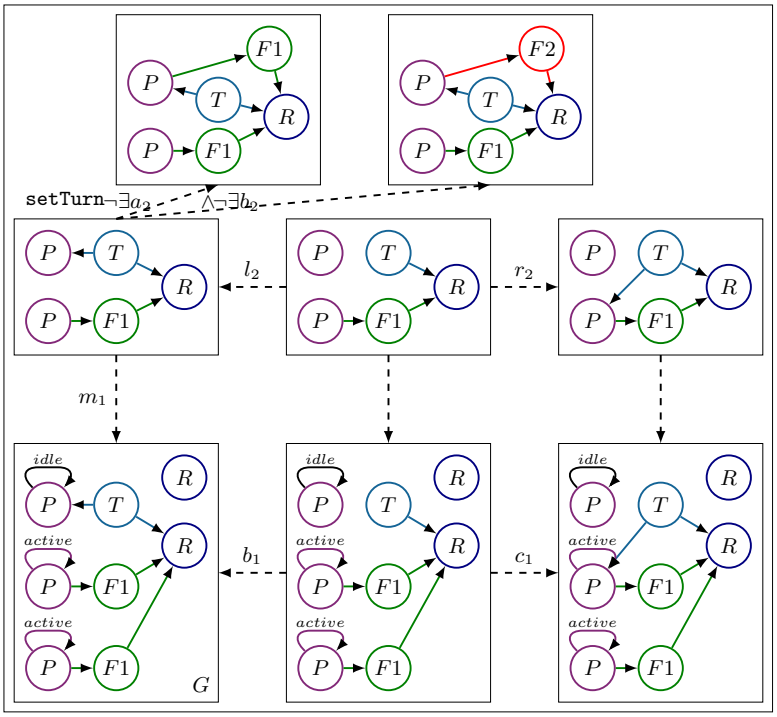


Figure 6. Rule application

Construction 1. The Shift-construction is inductively defined as follows:

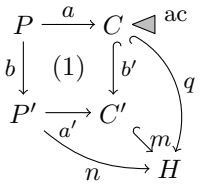
$$\begin{array}{ccc} P & \xrightarrow{b} & P' \\ {\scriptstyle a}\downarrow & (1) & \downarrow{\scriptstyle a'} \\ C & \xhookrightarrow{b'} & C' \\ \triangle & & \\ \text{ac} & & \end{array}$$

$\text{Shift}(b, \text{true}) = \text{true}$.
$\text{Shift}(b, \exists(a, \text{ac})) = \bigvee_{(a',b')\in\mathcal{F}} \exists(a', \text{Shift}(b', \text{ac}))$
if $\mathcal{F} = \{(a', b') \in \mathcal{E}' \mid b' \in \mathcal{M} \text{ and } (1) \text{ commutes}\} \neq \emptyset$ and false, otherwise.
$\text{Shift}(b, \neg\text{ac}) = \neg\text{Shift}(b, \text{ac})$ and $\text{Shift}(b, \wedge_{i\in I}\text{ac}_i) = \wedge_{i\in I}\text{Shift}(b, \text{ac}_i)$.

Proof. The statement is proved by structural induction.
Basis. For the application condition true, the equivalence holds trivially.
Inductive step. For an application condition of the form $\exists(a, \text{ac})$, we have to show

$$n \circ b \models \exists(a, \text{ac}) \iff n \models \text{Shift}(b, \exists(a, \text{ac})).$$

Only if. Let $n \circ b \models \exists(a, \text{ac})$. By definition of satisfiability, there is some $q \in \mathcal{M}$ with $q \circ a = n \circ b$ and $q \models \text{ac}$. By $\mathcal{E}'$-$\mathcal{M}$ pair factorization, there exist an object C' and morphisms $a' \colon P' \to C'$, $b' \colon C \to C'$, and $m \colon C' \hookrightarrow H$ with $(a', b') \in \mathcal{E}'$ and $m \in \mathcal{M}$ such that $m \circ a' = n$ and $m \circ b' = q$. Then $m \circ a' \circ b = n \circ b = q \circ a = m \circ b' \circ a$ and, by $m \in \mathcal{M}$, $a' \circ b = b' \circ a$, i.e., (1) commutes. Since $\mathcal{M}$ is closed under decomposition, $q, m \in \mathcal{M}$ implies $b' \in \mathcal{M}$. Thus, $(a', b') \in \mathcal{F}$. By inductive hypothesis, $q = m \circ b' \models \text{ac} \Leftrightarrow m \models \text{Shift}(b', \text{ac})$. Then $n = m \circ a' \models \exists(a', \text{Shift}(b', \text{ac}))$ and, by definition of Shift, $n \models \exists(b, \text{Shift}(a, \text{ac}))$.



If. Let $n \models \text{Shift}(b, \exists(a, \text{ac}))$. Then there is some $(a', b') \in \mathcal{F}$ such that $b' \in \mathcal{M}$, $a' \circ b = b' \circ a$, and $n \models \exists(a', \text{Shift}(b', \text{ac}))$. By definition of satisfiability, there is some $m \in \mathcal{M}$ such that $m \circ a' = n$ and $m \models \text{Shift}(b', \text{ac})$. By inductive hypothesis, $m \models \text{Shift}(b', \text{ac}) \Leftrightarrow m \circ b' \models \text{ac}$. Then there is some $q = m \circ b' \in \mathcal{M}$ such that $q \models \text{ac}$, i.e., $n \circ b = q \circ a \models \exists(a, \text{ac})$. This completes the inductive proof. □

Example 4. To illustrate the construction of shifting an application condition over morphisms, consider the application condition $\forall(b_6, \exists c_6)$ of the rule `enableR`, which is an application condition over the left-hand side of this rule. We want to shift this condition over the morphism v shown in the top of Fig. 7. The first step of the construction is

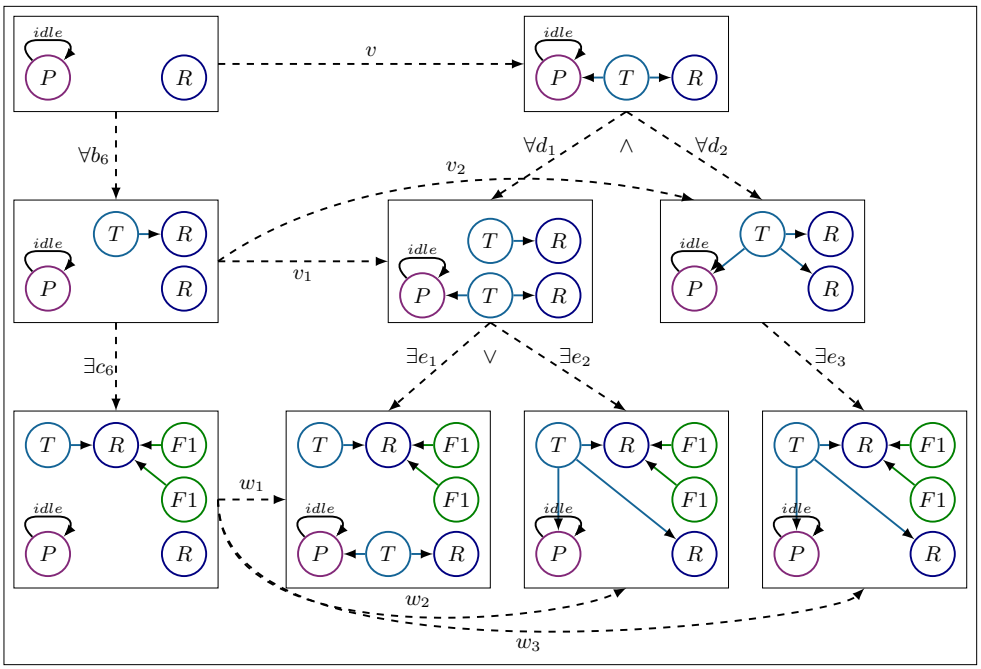


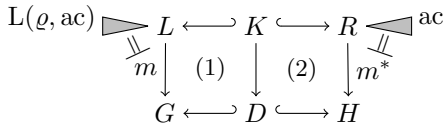
Figure 7. Shift of the application condition $\forall(b_6, \exists c_6)$ over a morphism.

shown in the upper part of Fig. 7, it results in the intermediate application condition $\text{Shift}(v, \forall(b_6, \exists c_6)) = \forall(d_1, \text{Shift}(v_1, \exists c_6)) \wedge \forall(d_2, \text{Shift}(v_2, \exists c_6))$. Since v_i has to be injective and the resulting object has to be an overlapping of the codomains of v and b_6 such that the diagram commutes, only these two solutions are possible. In a second step, the second part of the application condition has to be shifted over the two new morphisms v_1 and

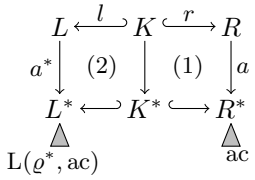
v_2. The result is shown in the lower part of Fig. 7 leading to the resulting application condition $\text{Shift}(v, \forall(b_6, \exists c_6)) = \forall(d_1, \exists e_1 \vee \exists e_2) \wedge \forall(d_2, \exists e_3)$.

The other result which is the key to prove the main results of the paper is that application conditions can be shifted along rules.

Lemma 3 (shift of application conditions over rules (Habel and Pennemann 2009)). There is a construction L such that, for each rule ϱ and each application condition ac over R, L transforms ac via ϱ into $\text{L}(\varrho, \text{ac})$ over L such that, for each direct transformation $G \Rightarrow_{\varrho,m,m^*} H$, we have $m \models \text{L}(\varrho, \text{ac}) \iff m^* \models \text{ac}$.



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



$\text{L}(\varrho, \text{true}) = \text{true}$.
$\text{L}(\varrho, \exists(a, \text{ac})) = \exists(a^*, \text{L}(\varrho^*, \text{ac}))$ if $\langle r, a\rangle$ has a pushout complement (1) and $\varrho^* = \langle L^* \hookleftarrow K^* \hookrightarrow R^*\rangle$ is the derived rule by constructing the pushout (2) and false, otherwise.
$\text{L}(\varrho, \neg\text{ac}) = \neg\text{L}(\varrho, \text{ac})$ and $\text{L}(\varrho, \wedge_{i\in I}\text{ac}_i) = \wedge_{i\in I}\text{L}(\varrho, \text{ac}_i)$.

Remark 4. The construction L transforms right application conditions via rules into left ones. The construction R with $\text{R}(\varrho, \text{ac}) = \text{L}(\varrho^{-1}, \text{ac})$ transforms left application conditions ac via the rule ϱ into right ones.

Example 5. Suppose we want to translate the application condition $\forall(b_6, \exists c_6)$ of the rule `enableR` to the right-hand side. Basically, this means to apply the rule to the first graph of the application condition, leading to a span which is applied as a rule to the second graph. The result is shown in Fig. 8, i.e. the translated application condition is $\forall(b_6^*, \exists c_6^*)$.

As a consequence of Shift-Lemma 3, every rule can be transformed into an equivalent rule where the right application condition is always true. A rule of the form $\langle p, \text{ac}_L, \text{true}\rangle$ is said to be a rule *with left application condition* and is abbreviated by $\langle p, \text{ac}_L\rangle$. This may be considered an improvement with respect to efficiency since, to check a right application condition, one must first apply the rule and then backtrack, in case the condition is not satisfied. However, left application conditions can be checked immediately after a match has been found.

Corollary 1 (rules with left application condition). There is a construction Left such that, for every rule ϱ, the rules ϱ and $\text{Left}(\varrho)$ are equivalent, where $\text{Left}(\varrho)$ is a rule with only left application condition.

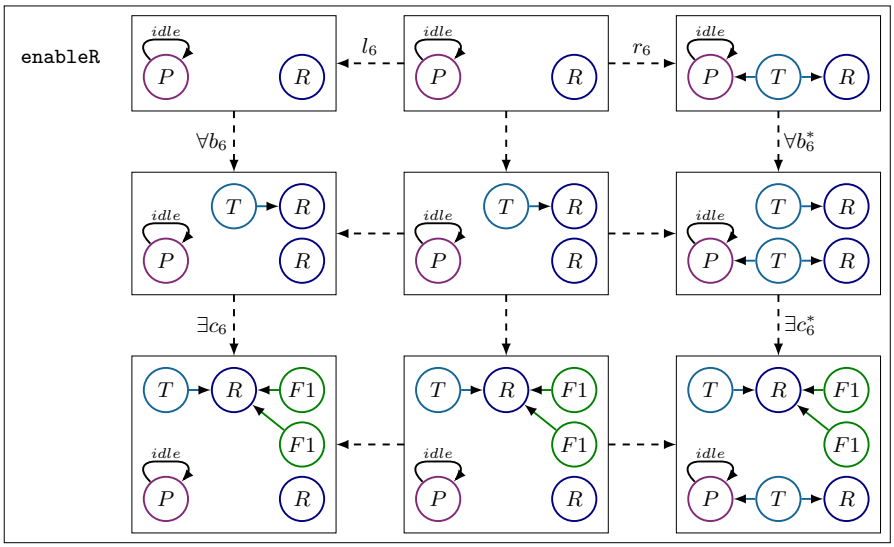


Figure 8. Shift of the application condition from left to right.

Proof. For a rule $\varrho = \langle p, \mathrm{ac}_L, \mathrm{ac}_R \rangle$, let $\mathrm{Left}(\varrho) = \langle p, \mathrm{ac}_L \wedge \mathrm{L}(\varrho, \mathrm{ac}_R) \rangle$. Then, by Definition 5 and Shift-Lemma 3, ϱ and $\mathrm{Left}(\varrho)$ are equivalent:

$$\begin{aligned} G \Rightarrow_{\varrho,m,m^*} H &\Leftrightarrow G \Rightarrow_{p,m,m^*} H \wedge m \models \mathrm{ac}_L \text{ and } m^* \models \mathrm{ac}_R \\ &\Leftrightarrow G \Rightarrow_{p,m,m^*} H \wedge m \models \mathrm{ac}_L \text{ and } m \models \mathrm{L}(\varrho, \mathrm{ac}_R) \\ &\Leftrightarrow G \Rightarrow_{p,m,m^*} H \wedge m \models \mathrm{ac}_L \wedge \mathrm{L}(\varrho, \mathrm{ac}_R) \\ &\Leftrightarrow G \Rightarrow_{\mathrm{Left}(\varrho),m,m^*} H \end{aligned}$$

□

4. Local Church-Rosser, Parallelism, and Concurrency

In this section, we present Local Church-Rosser, Parallelism, and Concurrency Theorems for rules with application conditions generalizing the well-known theorems for rules without application conditions (Ehrig et al. 2006b) and with negative application conditions (Lambers 2010). The proofs of the statements are based on the corresponding statements for rules without application conditions and Shift-Lemmata 2 and 3, saying that application conditions can be shifted over morphisms and rules. The structure of the proofs is as follows: We switch from transformations with application conditions to the corresponding transformations without application conditions, use the results for transformations without application conditions, and lift the results without application conditions to application conditions.

transformations with ACs	$\Longrightarrow$	result with ACs
$\downarrow$		$\uparrow$
transformations without ACs	$\Longrightarrow$	result without ACs

Remark 5. For every direct transformation $G \Rightarrow_{\varrho,m} H$ via a rule $\varrho = \langle p, \mathrm{ac}_L, \mathrm{ac}_R \rangle$, there is a direct transformation $G \Rightarrow_{p,m} H$ via the underlying plain rule p, called the underlying direct transformation without application conditions.

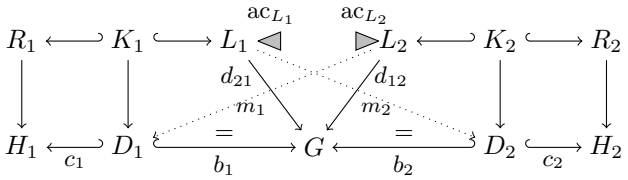
By Corollary 1, we may assume that the rules are rules with left application condition.

Assumption. In the following, for $i = 1, 2$, let $\varrho_i = \langle p_i, \mathrm{ac}_{L_i} \rangle$ be a rule with left application condition and $p_i = \langle L_i \hookleftarrow K_i \hookrightarrow R_i \rangle$ the underlying plain rule.

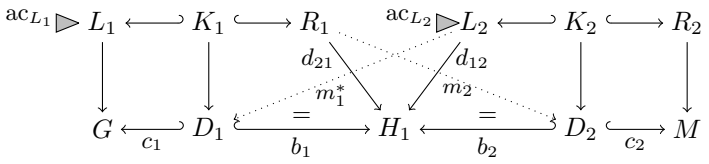
First, we consider direct transformations $H_1 \Leftarrow_{\varrho_1} G \Rightarrow_{\varrho_2} H_2$ and look for conditions under which there are direct transformations $H_1 \Rightarrow_{\varrho_2} M \Leftarrow_{\varrho_1} H_2$. In particular, the first obvious condition is that the underlying plain transformations are parallel independent. However, this is not enough, we must also require that the matches of ϱ_2 and ϱ_1 in H_1 and H_2, respectively, satisfy the application conditions of the corresponding rule. Similarly, we consider transformations $G \Rightarrow_{\varrho_1} H_1 \Rightarrow_{\varrho_2} M$ and look for conditions under which there are transformations $G \Rightarrow_{\varrho_2} H_2 \Rightarrow_{\varrho_1} M$. In this case, in addition to the sequential independence of the underlying plain rules, we must ask that the match of ϱ_2 in G satisfies its application condition and that the comatch of ϱ_1 to M satisfies the application condition $\mathrm{R}(\varrho_1, \mathrm{ac}_{L_1})$.

We formulate the notions of parallel and sequential independence and present the Local Church-Rosser Theorem.

Definition 6 (parallel and sequential independence). A pair of direct transformations $H_1 \Leftarrow_{\varrho_1, m_1} G \Rightarrow_{\varrho_2, m_2} H_2$ is *parallel independent* if there are morphisms $d_{ij} \colon L_i \to D_j$ such that $m_i = b_j \circ d_{ij}$ and $m_i' = c_j \circ d_{ij} \models \mathrm{ac}_{L_i}$ $(i, j \in \{1, 2\}$ and $i \neq j)$.



A pair of direct transformations $G \Rightarrow_{\varrho_1, m_1} H_1 \Rightarrow_{\varrho_2, m_2} M$ is *sequentially independent* if there are morphisms $d_{12} \colon R_1 \to D_2$ and $d_{21} \colon L_2 \to D_1$ such that $m_1^* = b_2 \circ d_{12}$, $m_2 = b_1 \circ d_{21}$, $m_2' = c_1 \circ d_{21} \models \mathrm{ac}_{L_2}$, and $m_1' = c_2 \circ d_{12} \models \mathrm{R}(\varrho_1, \mathrm{ac}_{L_1})$.



A pair of direct transformations that is not parallel (sequentially) independent, is called *parallel (sequentially) dependent.*

Example 6. The pair $H_1 \Leftarrow_{\texttt{setTurn}, m_1} G \Rightarrow_{\texttt{enableR}, m_2} H_2$ of direct transformations in Fig. 9 is parallel independent. The left rule application is the one already considered in Fig. 6. Obviously, m_2 matches the idle process to the uppermost process in G. The morphisms d_{12} and d_{21} exist such that $b_1 \circ d_{21} = m_2$, $b_2 \circ d_{12} = m_1$, $m_1' = c_2 \circ d_{12} \models \neg\exists a_2 \wedge \neg\exists b_2$ and $m_2' = c_1 \circ d_{21} \models \neg\exists a_6 \wedge \forall(b_6, \exists c_6)$.

The sequence $H_1 \Rightarrow_{\texttt{setFlag}, m_0} G \Rightarrow_{\texttt{enableR}, m_2} H_2$ of direct transformations in Fig. 10

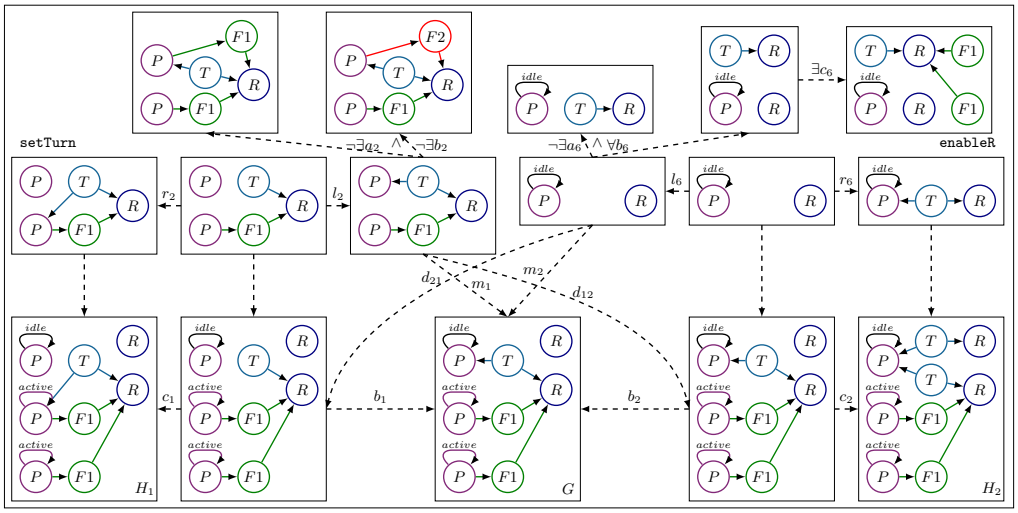


Figure 9. Parallel independent transformations.

is sequentially dependent. Note that m_0 matches the process of the rule `setFlag` to the lowermost process in H_1, while the second transformation is the one already considered in Fig. 9. The morphisms d_{12} and d_{21} exist such that $c_1 \circ d_{21} = m_2$, $c_2 \circ d_{12} = m_1^*$, $b_2 \circ d_{12} \models \mathrm{R}(\texttt{setFlag}, \exists a_1)$, but $b_1 \circ d_{21} \not\models \neg\exists a_6 \wedge \forall(b_6, \exists c_6)$. The transformations are sequentially dependent, since the rule `setFlag` adds a second flag, which is needed to fulfill the application condition $\forall(b_6, \exists c_6)$ of the rule `enableR`. Note that the transformations without application conditions would be sequentially independent.

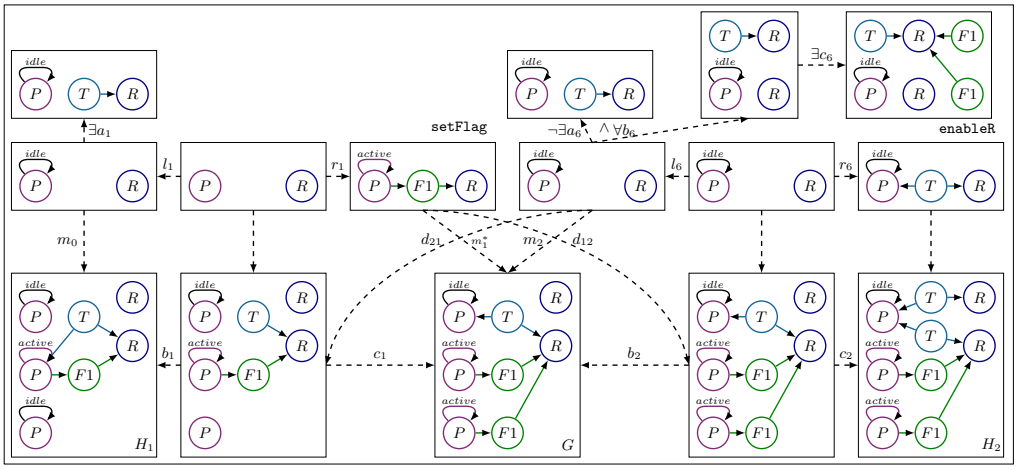


Figure 10. Sequentially dependent transformations.

By Definition 6, we immediately obtain the following fact.

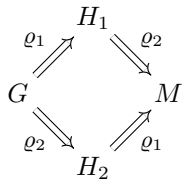
Fact 5. Direct transformations are parallel (sequentially) independent if, and only if, the underlying direct transformations without application conditions are parallel (sequentially) independent and the "induced" matches satisfy the corresponding application conditions.

By Definition 6, parallel and sequential independence are closely related.

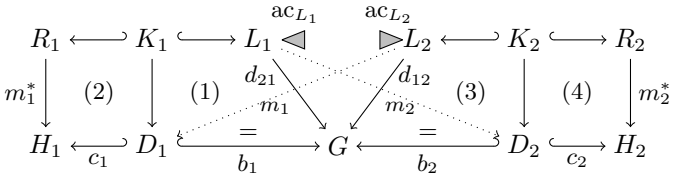
Fact 6. Two direct transformations $H_1 \Leftarrow_{\varrho_1,m_1} G \Rightarrow_{\varrho_2,m_2} H_2$ are parallel independent if, and only if, the two direct transformations $H_1 \Rightarrow_{\varrho_1^{-1},m_1^*} G \Rightarrow_{\varrho_2,m_2} H_2$ are sequentially independent, where m_1^* is the comatch of ϱ_1 in H_1.

Now we present the Local Church-Rosser Theorem.

Theorem 1 (Local Church-Rosser Theorem). Given two parallel independent direct transformations $H_1 \Leftarrow_{\varrho_1,m_1} G \Rightarrow_{\varrho_2,m_2} H_2$, there is an object M and there are direct transformations $H_1 \Rightarrow_{\varrho_2,m_2'} M \Leftarrow_{\varrho_1,m_1'} H_2$ such that the two transformations $G \Rightarrow_{\varrho_i,m_i} H_1 \Rightarrow_{\varrho_j,m_j'} M$ and $G \Rightarrow_{\varrho_2,m_2} H_2 \Rightarrow_{\varrho_1,m_1'} M$ are sequentially independent. Given two sequentially independent direct transformations $G \Rightarrow_{\varrho_1,m_1} H_1 \Rightarrow_{\varrho_2,m_2} M$, there is an object H_2 and there is a transformation $G \Rightarrow_{\varrho_2,m_2'} H_2 \Rightarrow_{\varrho_1,m_1'} M$ such that $H_1 \Leftarrow_{\varrho_1,m_1} G \Rightarrow_{\varrho_2,m_2'} H_2$ are parallel independent.

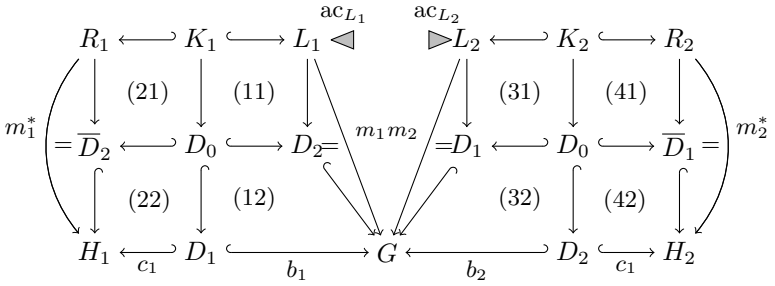


Proof. Let $H_1 \Leftarrow_{\varrho_1,m_1} G \Rightarrow_{\varrho_2,m_2} H_2$ be parallel independent. Then the underlying direct transformations $H_1 \Leftarrow_{p_1,m_1} G \Rightarrow_{p_2,m_2} H_2$ without application conditions are parallel independent. By the Local Church-Rosser Theorem without application conditions (Ehrig et al. 2006b), there are an object M and direct transformations $H_1 \Rightarrow_{p_2,m_2'} M \Leftarrow_{p_1,m_1'} H_2$ such that the transformations $G \Rightarrow_{p_1,m_1} H_1 \Rightarrow_{p_2,m_2'} M$ and $G \Rightarrow_{p_2,m_2} H_2 \Rightarrow_{p_1,m_1'} M$ are sequentially independent. By parallel independence, there are morphisms $d_{ij}\colon L_i \to D_j$ such that $m_i = b_j \circ d_{ij}$ $(i,j \in \{1,2\}$ and $i \neq j)$.

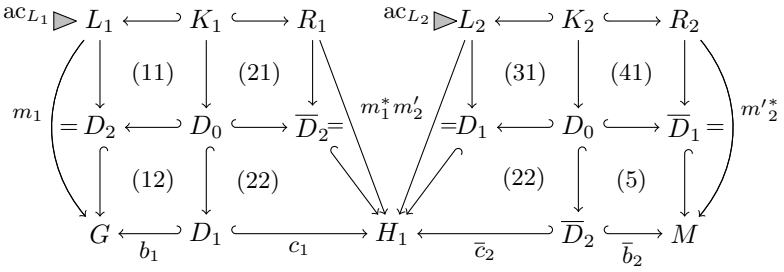


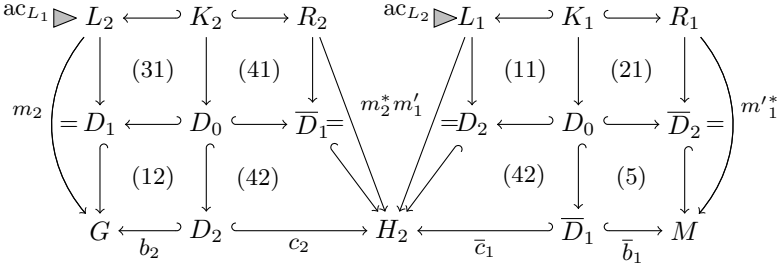
The morphisms are used for the decomposition of the pushouts (i) into pushouts (i1)

and (i2) for $i = 1, \ldots, 4$.



The pushouts can be rearranged as in the figures below. Since the composition of pushouts yields pushouts, we obtain direct transformations $H_1 \Rightarrow_{p_2,m_2'} M \Leftarrow_{p_1,m_1'} H_2$ such that, for $i \in \{1,2\}$ and $i \neq j$, $G \Rightarrow_{p_i,m_i} H_i \Rightarrow_{p_j,m_j'} M$ are sequentially independent: there are morphisms $\overline{d}_{ij} \colon R_i \to \overline{D}_j$ and $\overline{d}_{ji} \colon L_j \to \overline{D}_i$ such that $\overline{c}_j \circ \overline{d}_{ij} = m_i^*$ and $c_i \circ \overline{d}_{ji} = m_j'$.



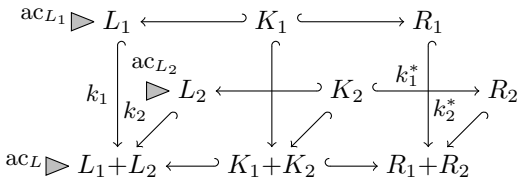


By assumption, $m_i, m_i' \models \mathrm{ac}_{L_i}$. By Shift-Lemma 3, $m_i \models \mathrm{ac}_{L_i} \Leftrightarrow m_i^* \models \mathrm{R}(\varrho_i, \mathrm{ac}_i)$. Thus, there is a transformation $G \Rightarrow_{\varrho_i,m_i} H_i \Rightarrow_{\varrho_j,m_j'} M$ that is sequentially independent. The second statement can be proved with the help of the first statement and Fact 6. □

Next, we consider parallel rules and parallel transformations. The parallel rule $\varrho_1 + \varrho_2$ of the rules ϱ_1 and ϱ_2 can be defined with help of the binary coproducts of the components of the rules, because, by the General Assumption, $\langle \mathcal{C}, \mathcal{M} \rangle$ has binary coproducts.

Definition 7 (parallel rule and transformation). The *parallel rule* of ϱ_1 and ϱ_2 is the rule $\varrho_1 + \varrho_2 = \langle p, \mathrm{ac}_L \rangle$ where $p = \langle L_1 + L_2 \hookleftarrow K_1 + K_2 \hookrightarrow R_1 + R_2 \rangle$ is the parallel rule

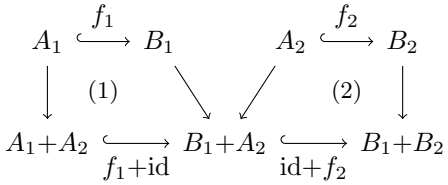
of p_1 and p_2 and $\mathrm{ac}_L = \wedge_{i=1}^{2}\mathrm{Shift}(k_i, \mathrm{ac}_{L_i}) \wedge \mathrm{L}(p_1{+}p_2, \mathrm{Shift}(k_i^*, \mathrm{R}(\varrho_i, \mathrm{ac}_{L_i})))$.



A direct transformation via a parallel rule is called *parallel* direct transformation or parallel transformation, for short.

Fact 7. The morphisms $K_1{+}K_2 \hookrightarrow L_1{+}L_2$ and $K_1{+}K_2 \hookrightarrow R_1{+}R_2$ are in $\mathcal{M}$.

Proof. Binary coproducts are compatible with $\mathcal{M}$, i.e., $f_1, f_2 \in \mathcal{M}$ implies $f_1{+}f_2 \in \mathcal{M}$. In fact, pushout (1) with $f_1 \in \mathcal{M}$ implies $(f_1{+}\mathrm{id}) \in \mathcal{M}$ and pushout (2) with $f_2 \in \mathcal{M}$ implies $(\mathrm{id}{+}f_2) \in \mathcal{M}$, but now $(f_1{+}f_2) = (\mathrm{id}{+}f_2) \circ (f_1{+}\mathrm{id}) \in \mathcal{M}$ by closure under composition.



□

Example 7. In the upper row of Fig. 11, the parallel rule `setTurn`+`enableR` is shown,

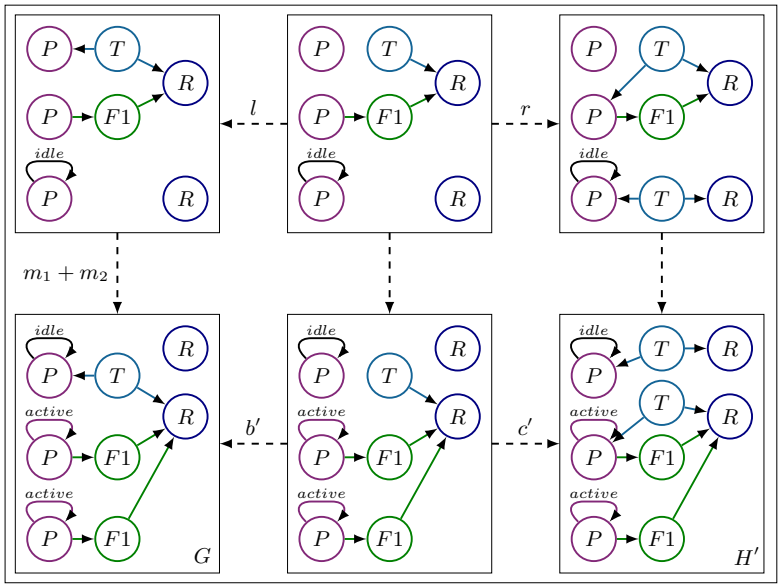


Figure 11. Parallel rule and transformation.

where we have not shown the application conditions due to the large amount of them,

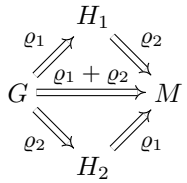
which basically describe for various overlappings of processes or resources that there is no F1- or F2-flag between the process and the resource from the rule setTurn, no turn on the resource of enableR, and all active resources have at least two F1-flags pointing to them. The application $G \Rightarrow_{\texttt{setTurn+enableR},m_1+m_2} H'$ of this parallel rule is shown in Fig. 11 and combines the effects of both rules to G leading to the graph H', where both the turn points to an active process and the previously disabled resource is now activated.

Two rules ϱ and ϱ' are isomorphic, denoted by $\varrho \cong \varrho'$, if there are isomorphisms $\mathrm{iso}_L, \mathrm{iso}_K, \mathrm{iso}_R$ between the components such that the arising diagrams commute and the application conditions are isomorphic with respect to iso_L. As an immediate consequence of the definition we obtain the following fact.

Fact 8. For all rules ϱ_1 and ϱ_2, we have $\varrho_1 + \varrho_2 \cong \varrho_2 + \varrho_1$.

The connection between sequentially independent direct transformations and parallel direct transformations using the parallel rule (Def. 7) is expressed by the Parallelism Theorem.

Theorem 2 (Parallelism). Given two sequentially independent direct transformations $G \Rightarrow_{\varrho_1,m_1} H_1 \Rightarrow_{\varrho_2,m_2'} M$, there is a parallel transformation $G \Rightarrow_{\varrho_1+\varrho_2,m} M$. Given a parallel transformation $G \Rightarrow_{\varrho_1+\varrho_2,m} M$, there are sequentially independent direct transformations $G \Rightarrow_{\varrho_1,m_1} H_1 \Rightarrow_{\varrho_2,m_2'} M$ and $G \Rightarrow_{\varrho_2,m_2} H_i \Rightarrow_{\varrho_1,m_1'} M$.



Proof. Let $G \Rightarrow_{\varrho_1,m_1} H_1 \Rightarrow_{\varrho_2,m_2'} M$ be sequentially independent. Then the underlying transformation without application conditions is sequentially independent and, by the Parallelism Theorem without application conditions (Ehrig et al. 2006b), there is a parallel transformation $G \Rightarrow_{p_1+p_2,m} M$ with $m_1 = m \circ k_1$ and $m'^*_2 = m^* \circ k_2^*$. By assumption, $m_1 \models \mathrm{ac}_{L_1}$ and $m_2' \models \mathrm{ac}_{L_2}$. By Shift-Lemmata 2 and 3 and Definition 7,

$$
\begin{aligned}
(*) \quad & m_1 \models \mathrm{ac}_{L_1} \wedge m_2' \models \mathrm{ac}_{L_2} \\
\Leftrightarrow \quad & m \models \mathrm{Shift}(k_1, \mathrm{ac}_{L_1}) \wedge m'^*_2 \models \mathrm{R}(\varrho_2, \mathrm{ac}_{L_2}) \\
\Leftrightarrow \quad & m \models \mathrm{Shift}(k_1, \mathrm{ac}_{L_1}) \wedge m^* \models \mathrm{Shift}(k_2^*, \mathrm{R}(\varrho_2, \mathrm{ac}_{L_2})) \\
\Leftrightarrow \quad & m \models \mathrm{Shift}(k_1, \mathrm{ac}_{L_1}) \wedge \mathrm{L}(p_1{+}p_2, \mathrm{Shift}(k_2^*, \mathrm{R}(\varrho_2, \mathrm{ac}_{L_2}))) = \mathrm{ac}_L
\end{aligned}
$$

Thus, $m \models \mathrm{ac}_L$, i.e., the parallel transformation satisfies the application condition.

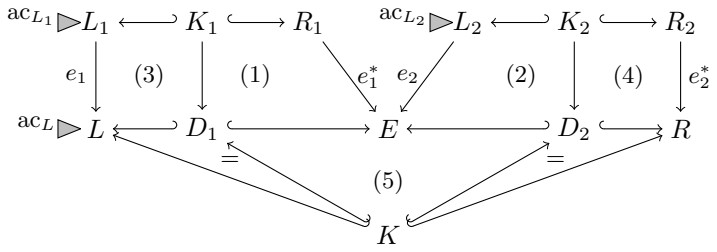
Vice versa, let $G \Rightarrow_{\varrho_1+\varrho_2,m} M$ be a parallel transformation. Then there is an underlying parallel transformation without application conditions, and, by the Parallelism Theorem without application conditions (Ehrig et al. 2006b), there is a sequentially independent direct transformation $G \Rightarrow_{p_1,m_1} H_1 \Rightarrow_{p_2,m_2'} M$ with $m_1 = m \circ k_1$ and $m'^*_2 = m^* \circ k_2^*$. By assumption, $m \models \mathrm{ac}_L$. By (*), $m_1 \models \mathrm{ac}_{L_1}$ and $m_2' \models \mathrm{ac}_{L_2}$, i.e., the sequentially independent direct transformations satisfy the application conditions. By $\varrho_1 + \varrho_2 \cong \varrho_2 +$

ϱ_1, there is also a sequentially independent direct transformation $G \Rightarrow_{p_2,m_2} H_2 \Rightarrow_{p_1,m_1'} M$ with $m_2 \models \mathrm{ac}_{L_2}$ and $m_1' \models \mathrm{ac}_{L_1}$. □

Finally, we consider transformations of the form $G \Rightarrow_{\varrho_1} H \Rightarrow_{\varrho_2} M$, but the assumption of sequential independence is dropped. This leads to the notions of an E-dependency relation, an E-concurrent rule for ϱ_1 and ϱ_2, E-concurrent transformations, and E-related transformations. The connection between E-related and E-concurrent transformations is established in the Concurrency Theorem.

The construction of an E-concurrent rule is based on an E-dependency relation which guarantees the existence of some pushout complements. It is defined with the help of pushouts and pullbacks along $\mathcal{M}$-morphisms. The application condition of the E-concurrent rule guarantees that, whenever the E-concurrent rule is applicable, the rule ϱ_1 and, afterwards, the rule ϱ_2 is applicable.

Definition 8 (E-concurrent rule). Let $\mathcal{E}'$ be a class of morphism pairs with the same codomain. Given two rules ϱ_1 and ϱ_2, an object E with morphisms $e_1^*\colon R_1 \to E$ and $e_2\colon L_2 \to E$ is an *E-dependency relation* for ϱ_1 and ϱ_2 if $(e_1^*, e_2) \in \mathcal{E}'$ and the pushout complements (1) and (2) over $K_1 \hookrightarrow R_1 \to E$ and $K_2 \hookrightarrow L_2 \to E$ exist. Given such an E-dependency relation for ϱ_1 and ϱ_2, the *E-concurrent rule* of ϱ_1 and ϱ_2 is the rule $\varrho_1 *_E \varrho_2 = \langle p, \mathrm{ac}_L \rangle$ where $p = \langle L \hookleftarrow K \hookrightarrow R \rangle$ with pushouts (3), (4) and pullback (5), $\varrho_1^* = \langle L \hookleftarrow D_1 \hookrightarrow E \rangle$ is the rule derived by ϱ_1 and k_1, and $\mathrm{ac}_L = \mathrm{Shift}(e_1, \mathrm{ac}_{L_1}) \wedge \mathrm{L}(\varrho_1^*, \mathrm{Shift}(e_2, \mathrm{ac}_{L_2}))$.



Example 8. In Fig. 12, the E-concurrent rule construction is depicted, leading to the E-related sequence $G' \Rightarrow_{\texttt{setFlag},m_0} G \Rightarrow_{\texttt{enableR},m_2} H_2$ of direct transformations already considered in Fig. 10. Note that e_1 matches the process of `setFlag` to the lowermost process and e_2 matches the process of `enableR` to the uppermost process. Moreover, $\mathrm{ac}_L = \mathrm{Shift}(e_1, \mathrm{ac}_1) \wedge \mathrm{L}(\varrho_6^*, \mathrm{Shift}(e_2, \mathrm{ac}_6))$ is not depicted explicitly, since it becomes too large. It expresses that the lowermost resource should be connected to a token ($\mathrm{Shift}(e_1, \exists a_1)$), that the uppermost resource should not already be connected to a token ($\mathrm{L}(\varrho_6^*, \mathrm{Shift}(e_2, \neg\exists a_6))$), that the lowermost resource should already be connected to a `F1`-flag, and that other enabled resources should already be connected to at least two `F1`-flags ($\mathrm{L}(\varrho_6^*, \mathrm{Shift}(e_2, \forall(b_6, \exists c_6)))$).

For rules without application conditions, the parallel rule is a special case of the E-concurrent rule with $E = R_1 + L_2$ (Ehrig et al. 2006b). For rules with application conditions, in general, this is not the case: While the application conditions for the parallel rule must guarantee the applicability of the rules in each order, the application condition

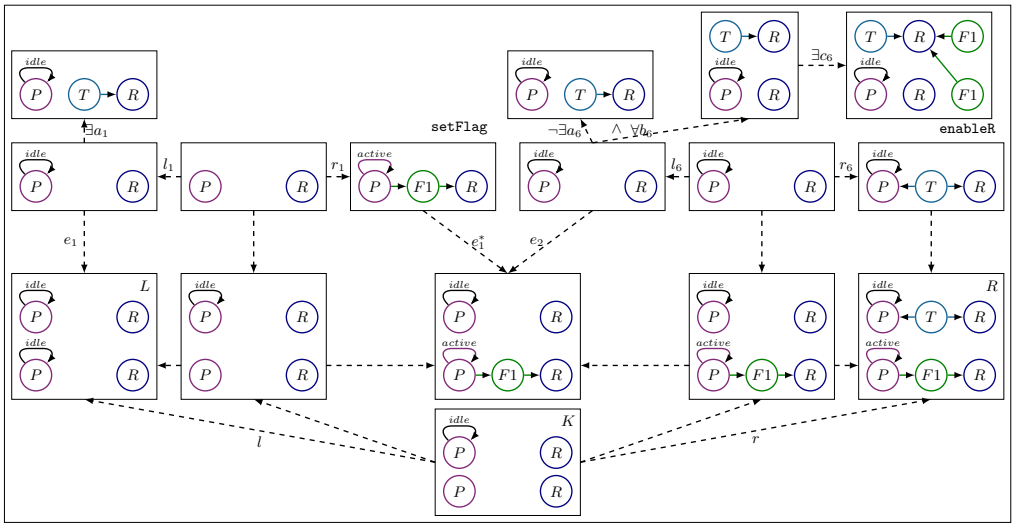


Figure 12. E-concurrent rule construction.

for the E-concurrent rule must guarantee the applicability of the rules in the given order. Nevertheless, the parallel rule of two rules can be constructed from two concurrent rules of the rules, one for each order: For $i, j \in \{1, 2\}$ with $i \neq j$, let $\mathrm{ac}_{L_{ij}}$ be the application condition of the E_{ij}-concurrent rule of ϱ_i and ϱ_j with $E_{ij} = R_i{+}L_j$. Then the rule $p_1{+}p_2$ with application condition $\mathrm{ac}_{L_{12}} \wedge \mathrm{ac}_{L_{21}}$ is called *symmetric* concurrent rule of ϱ_1 and ϱ_2 and is denoted by $\varrho_1 * \varrho_2$.

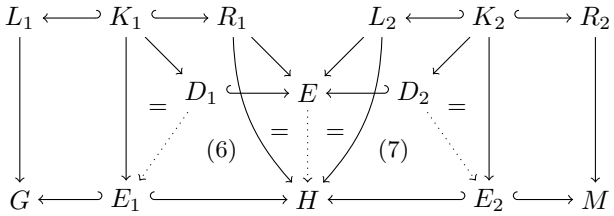
Lemma 4 (parallel and symmetric concurrent rules). For rules ϱ_1 and ϱ_2, the parallel rule and the symmetric concurrent rule are equivalent: $\varrho_1{+}\varrho_2 \equiv \varrho_1 * \varrho_2$.

Proof. For plain rules p_1 and p_2, the parallel rule $p_1{+}p_2$ and the concurrent rules $p_i *_{R_i+L_j} p_j$ are equivalent (Ehrig et al. 2006b). By Definitions 7 and 8, $m \models \mathrm{ac}_L \Leftrightarrow m \models \wedge_{i=1}^{2} \mathrm{Shift}(k_i, \mathrm{ac}_{L_i}) \wedge \mathrm{L}(\varrho_i^*, \mathrm{Shift}(k'_j, \mathrm{ac}_{L_j})) \Leftrightarrow m \models \mathrm{ac}_{L_{12}} \wedge \mathrm{ac}_{L_{21}}$, i.e., the parallel rule and the symmetric concurrent rule are equivalent. □

We consider E-concurrent transformations via E-concurrent rules and E-related transformations via pairs of rules.

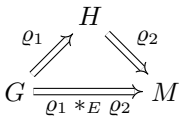
Definition 9 (E-concurrent and E-related transformation). A direct transformation via an E-concurrent rule is called *E-concurrent* direct transformation or E-concurrent transformation, for short. A transformation $G \Rightarrow_{\varrho_1} H \Rightarrow_{\varrho_2} M$ is *E-related* if there are morphisms $E \to H$, $D_1 \to E_1$, and $D_2 \to E_2$ such that the triangles commute

and the diagrams (6) and (7) are pushouts.



Now we present a Concurrency Theorem for rules with application conditions.

Theorem 3 (Concurrency). Let E be a dependency relation for ϱ_1 and ϱ_2. For every E-related transformation $G \Rightarrow_{\varrho_1,m_1} H \Rightarrow_{\varrho_2,m_2} M$, there is an E-concurrent transformation $G \Rightarrow_{\varrho_1 *_E \varrho_2,m} M$. Vice versa, for every E-concurrent transformation $G \Rightarrow_{\varrho_1 *_E \varrho_2,m} M$, there is an E-related transformation $G \Rightarrow_{\varrho_1,m_1} H \Rightarrow_{\varrho_2,m_2} M$.



Proof. Let $G \Rightarrow_{\varrho_1,m_1} H \Rightarrow_{\varrho_2,m_2} M$ be E-related. Then the underlying transformation without application conditions is E-related and, by the Concurrency Theorem without application conditions (Ehrig et al. 2006b), there is an E-concurrent transformation $G \Rightarrow_{p_1 * p_2,m} M$. By assumption, $m_1 \models \mathrm{ac}_{L_1}$ and $m_2 \models \mathrm{ac}_{L_2}$. By Shift-Lemmta 2 and 3 and Definition 8, we have

$$\begin{aligned}
(*) \quad & m_1 \models \mathrm{ac}_{L_1} \text{ and } m_2 \models \mathrm{ac}_{L_2} \\
\Leftrightarrow \quad & m \models \mathrm{Shift}(k_1, \mathrm{ac}_{L_1}) \text{ and } m' \models \mathrm{Shift}(k_2, \mathrm{ac}_{L_2}) \\
\Leftrightarrow \quad & m \models \mathrm{Shift}(k_1, \mathrm{ac}_{L_1}) \text{ and } m \models \mathrm{L}(p_1^*, \mathrm{Shift}(k_2, \mathrm{ac}_{L_2})) \\
\Leftrightarrow \quad & m \models \mathrm{Shift}(k_1, \mathrm{ac}_{L_1}) \wedge \mathrm{L}(p_1^*, \mathrm{Shift}(k_2, \mathrm{ac}_{L_2})) = \mathrm{ac}_L.
\end{aligned}$$

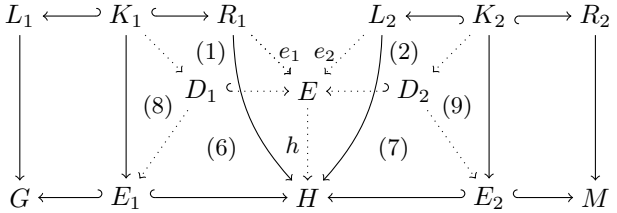
Thus, $m \models \mathrm{ac}_L$, i.e., the E-concurrent transformation satisfies the application condition. Let $G \Rightarrow_{\varrho,m} M$ be an E-concurrent transformation. Then the underlying direct transformation without application conditions is E-concurrent and, by the Concurrency Theorem without application conditions (Ehrig et al. 2006b), there is an E-related transformation $G \Rightarrow_{p_1,m_1} H \Rightarrow_{p_2,m_2} M$. By assumption, $m \models \mathrm{ac}_L$. By statement (*), $m_1 \models \mathrm{ac}_{L_1}$ and $m_2 \models \mathrm{ac}_{L_2}$, i.e., the E-related transformation satisfies the application conditions. □

In order to apply the Concurrency Theorem to a transformation, it remains to construct an E-related transformation corresponding to Fact 5.29 in (Ehrig et al. 2006b). For this purpose, we use an $\mathcal{M}$-adhesive category with $\mathcal{E}'$-$\mathcal{M}$ pair factorization.

Fact 9 (construction of E-related transformations). For every transformation $G \Rightarrow_{\varrho_1,m_1} H \Rightarrow_{\varrho_2,m_2} M$ there is an E-dependency relation E such that $G \Rightarrow_{\varrho_1,m_1} H \Rightarrow_{\varrho_2,m_2} M$ is E-related.

Proof. Given a transformation $G \Rightarrow_{\varrho_1,m_1,m_1^*} H \Rightarrow_{\varrho_2,m_2} M$, let $(e_1, e_2) \in \mathcal{E}'$, $h \in \mathcal{M}$ be an $\mathcal{E}'$-$\mathcal{M}$ pair factorization of m_1^* and m_2 with $h \circ e_1 = m_1^*$ and $h \circ e_2 = m_2$. Construct

now diagram (6) as pullback of $E_1 \hookrightarrow H \hookleftarrow E$. By the universal pullpack property, there is a morphism $K_1 \to D_1$ such that diagrams (1) and (8) commute. Since $h \in \mathcal{M}$, (6) is a pullback, and (1)+(6) is a pushout, the $\mathcal{M}$-pushout-pullback decomposition property implies that diagram (1) below is a pushout. Analogously, diagram (2) below is a pushout.



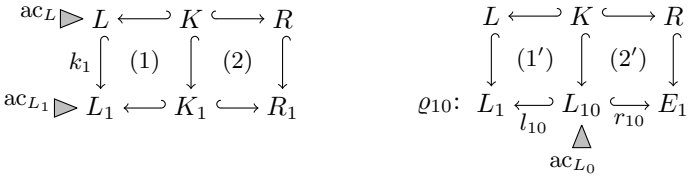
Thus, E with $(e_1, e_2) \in \mathcal{E}'$ is an E-dependency relation and $G \Rightarrow_{\varrho_1,m_1} H \Rightarrow_{\varrho_2,m_2} M$ is E-related. □

5. Amalgamation

In this section, we present an Amalgamation Theorem for rules with application conditions generalizing the well-known one for rules without application conditions (Boehm et al. 1987; Corradini et al. 1997). The Amalgamation Theorem handles two direct transformations, which may be parallel dependent. Roughly speaking, for a ϱ-amalgamable pair of direct transformations $H_1 \Leftarrow_{\varrho_1} G \Rightarrow_{\varrho_2} H_2$, there is a direct transformation $G \Rightarrow M$ via the ϱ-amalgamated rule ϱ', and vice versa. The effect of the ϱ-amalgamated rule ϱ' may be described by the application of ϱ_i and the remainder of ϱ' with respect to ϱ_i $(i = 1, 2)$. The Multi-Amalgamation Theorem in (Golas et al. 2012; Golas 2011) generalizes the Amalgamation Theorem to the case of $n \geq 2$ amalgamable direct transformations.

The amalgamation of rules is based on the notions of a subrule and its remainder. In the following, let $\varrho = \langle p, \mathrm{ac}_L \rangle$ be a rule with $p = \langle L \hookleftarrow K \hookrightarrow R \rangle$.

Definition 10 (subrule and remainder). A rule ϱ is a *subrule* of a rule ϱ_1 if there are embedding $\mathcal{M}$-morphisms $L \hookrightarrow L_1$, $K \hookrightarrow K_1$, and $R \hookrightarrow R_1$ such that diagrams (1) and (2) are pullbacks, the pushout complement $(1')$ of $K \hookrightarrow L \hookrightarrow L_1$ exists, and the application conditions ac_L and ac_{L_1} are *compatible*, i.e., there is some application condition $\mathrm{ac}_{L_{10}}$ over L_{10} such that $\mathrm{ac}_{L_1} \equiv_{\varrho_1} \mathrm{Shift}(k_1, \mathrm{ac}_L) \wedge \mathrm{L}(\varrho_{10}, \mathrm{Shift}(r_{10}, \mathrm{ac}_{L_{10}}))$ where $r_{10} \colon L_{10} \hookrightarrow E_1$ and ϱ_{10} is the rule derived from ϱ and k_1, i.e., $(1')$ and $(2')$ are pushouts. A rule ϱ_1' is a *remainder* of ϱ_1 with respect to ϱ if $\varrho_1 = \varrho *_{E_1} \varrho_1'$ for some E_1-dependency relation for ϱ and ϱ_1'.



Example 9. We want to model an additional behavior of the system, where two active, waiting processes without a turn variable may decide to activate a disabled resource and

one of them gets the turn variable. The first rule is depicted in the top of Fig. 13 and shows the handling of the first process, whose flag is redirected and which gets the new turn variable. The second rule is shown in the bottom of this figure which only redirects the flag of a process to a previously disabled resource. In the middle row of Fig. 13 the subrule is shown which has to ensure that the newly enabled resource and its turn variable are synchronized. This rule is actually a subrule of ϱ_7 and ϱ_8, because the given squares are pullbacks, in both cases the pushout complements exist and are equal to the left-hand sides of the corresponding rule, and for the application conditions we have that $\mathrm{ac}_i \cong \mathrm{Shift}(k_i, \mathrm{ac}_0) \wedge \mathrm{L}(\rho_i^*, \mathrm{Shift}(r_i, \neg\exists b_i))$ for $i = 7, 8$. The remainder rules ϱ_7' and ϱ_8' are given in Fig. 14. Note, that in ϱ_7', the turn variable appears because it has to be connected to the process, while in ϱ_8' we do not need it. In addition, the application condition $\neg\exists b_i$ is translated into an application condition $\neg\exists b_i'$ for both remainder rules with $i = 7, 8$.

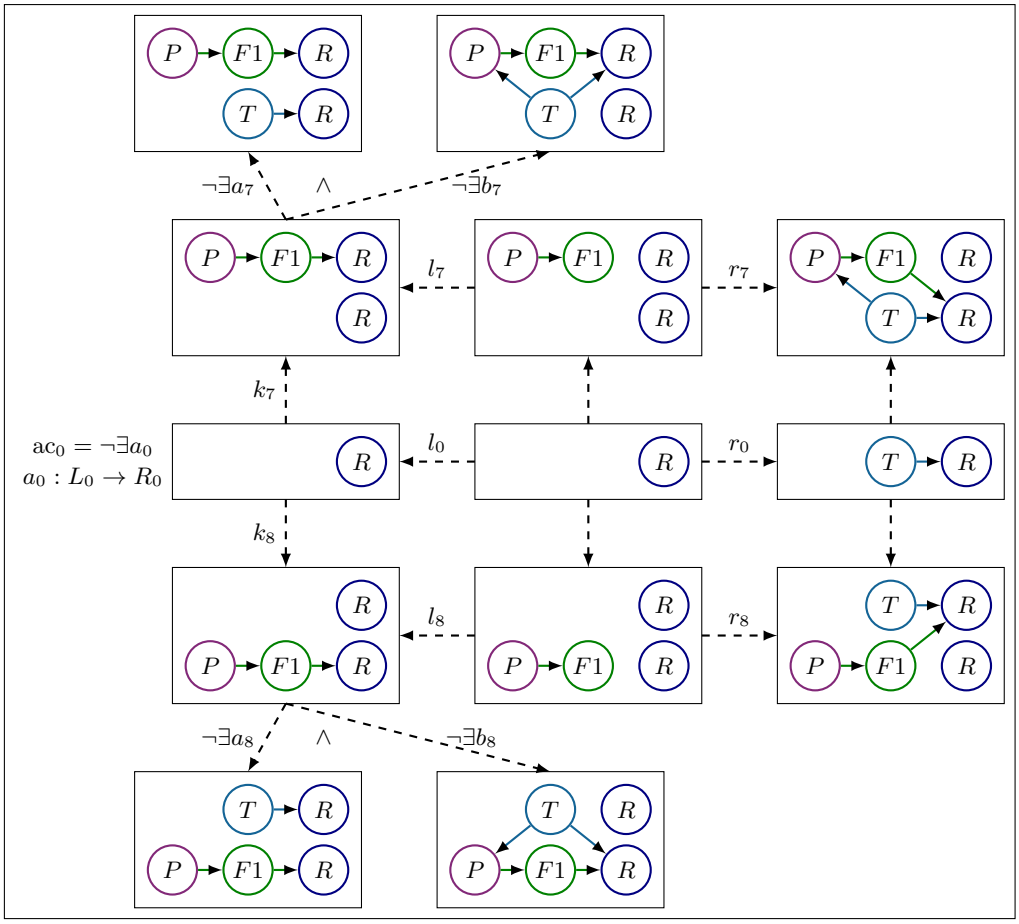


Figure 13. The subrule ϱ_0 of the rules ϱ_7 and ϱ_8.

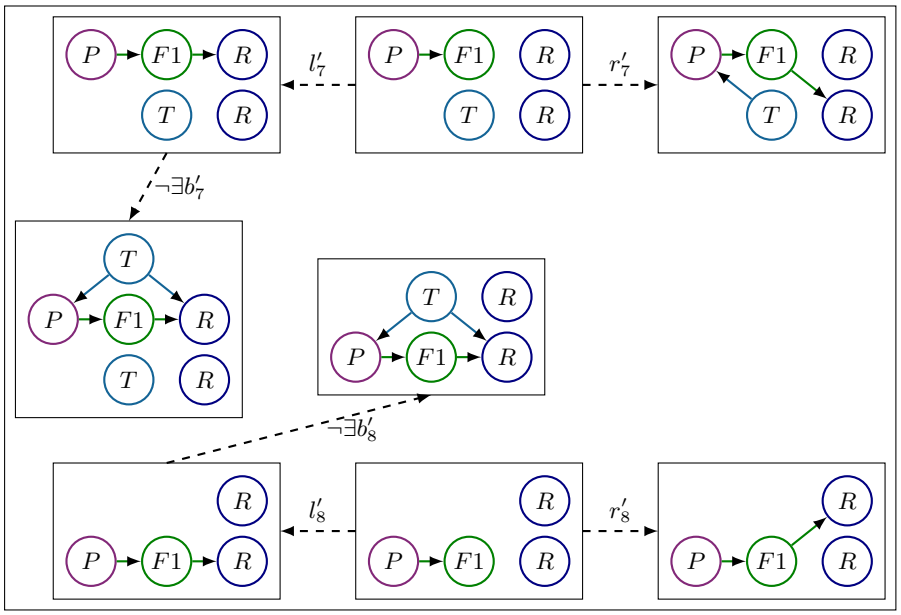


Figure 14. The remainder rules ϱ'_7 and ϱ'_8.

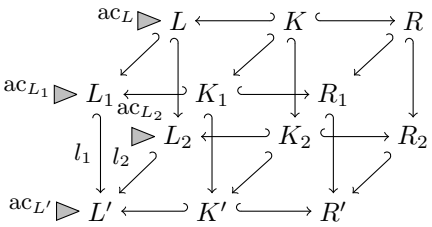
Every rule can be decomposed into the subrule and a remainder.

Theorem 4 (existence of a remainder (Golas et al. 2012)). For every rule ϱ_1 with subrule ϱ, there is a remainder ϱ'_1 of ϱ_1 with respect to ϱ.

The construction of an amalgamated rule generalizes the one of a parallel rule $\varrho_1 + \varrho_2$ of the rules ϱ_1 and ϱ_2: For a common subrule ϱ of ϱ_1 and ϱ_2, the ϱ-amalgamated rule $\varrho_1 \oplus_\varrho \varrho_2$ of ϱ_1 and ϱ_2 can be defined with the help of pushouts along $\mathcal{M}$-morphisms of the components of the rules. It generalizes the construction of amalgamated rules for rules without application conditions (Boehm et al. 1987; Corradini et al. 1997) and makes use of shifting of application conditions over morphisms (Shift-Lemma 2).

Definition 11 (amalgamated rule). Given a common subrule ϱ of rules ϱ_1 and ϱ_2, the *ϱ-amalgamated rule* of ϱ_1 and ϱ_2, denoted by $\varrho_1 \oplus_\varrho \varrho_2$, is the rule $\langle p', \mathrm{ac}_{L'} \rangle$, where L', K', and R' are the pushout objects in the left, middle, and right diagram, respectively, $K' \to L'$ and $K' \to R'$ are the uniquely existing morphisms, $p' = \langle L' \leftarrow K' \to R' \rangle$, and $\mathrm{ac}_{L'} = \mathrm{Shift}(l_1, \mathrm{ac}_{L_1}) \wedge \mathrm{Shift}(l_2, \mathrm{ac}_{L_2})$. Note that the morphisms $K' \hookrightarrow L'$ and $K' \hookrightarrow R'$

are in $\mathcal{M}$.



Example 10. The amalgamated rule $\varrho = \varrho_7 \oplus_{\varrho_0} \varrho_8$ is shown in the upper rows of Fig. 15. It combines the effects of ϱ_7 and ϱ_8 where both rules use the same resource as the new target of the flags and only create one turn variable for this resource. Note that the application condition $\neg\exists d$ forbids that the upper and lower process may be matched non-injectively and are connected via a turn variable to the resource, while different other overlappings, which may not occur in valid systems, are not explicitly shown.

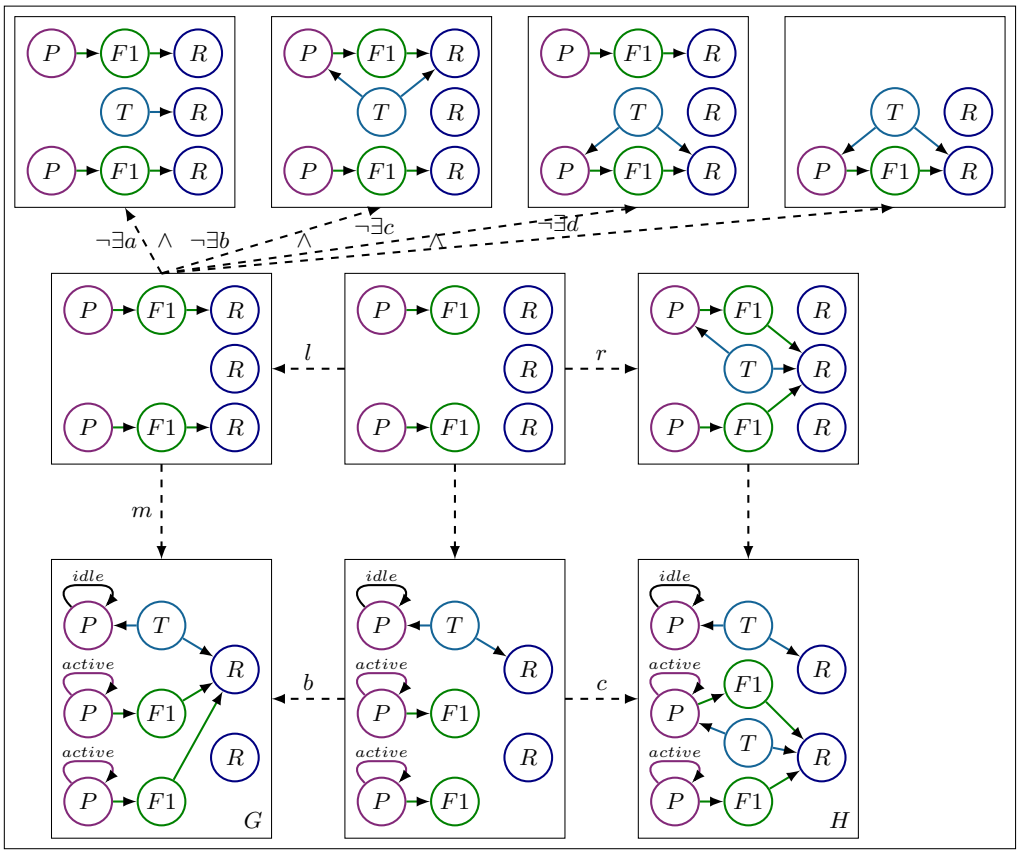


Figure 15. The amalgamated rule $\varrho = \varrho_7 \oplus_{\varrho_0} \varrho_8$.

By definition, parallel rules are special amalgamated rules.

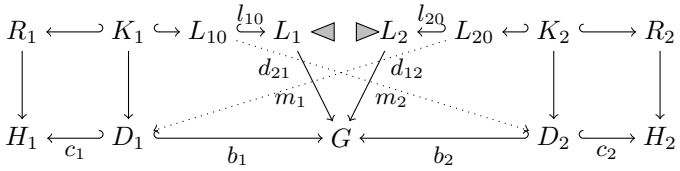
Fact 10 (parallel rules are amalgamated rules). If $\langle \mathcal{C}, \mathcal{M} \rangle$ has an $\mathcal{M}$-initial object I, init $= \langle I \hookleftarrow I \hookrightarrow I \rangle$ is a subrule of ϱ_1 and of ϱ_2 and $\varrho_1 + \varrho_2 \cong \varrho_1 \oplus_{\text{init}} \varrho_2$.

The subrule property is inherited to amalgamated rules: If a rule is a common subrule of rules, then these rules are subrules of the amalgamated rule of the rules.

Lemma 5 (subrule inheritance (Golas et al. 2012)). If ϱ is a common subrule of ϱ_1 and ϱ_2, then ϱ_1 and ϱ_2 are subrules of $\varrho_1 \oplus_\varrho \varrho_2$.

The application of an amalgamated rule yields an amalgamated transformation. Amalgamability of direct transformations generalizes parallel independence of direct transformations.

Definition 12 (amalgamated and amalgamable transformation). A direct transformation via a ϱ-amalgamated rule is called *ϱ-amalgamated* direct transformation or *ϱ-amalgamated* transformation, for short. For $i = 1, 2$, the direct transformations $G \Rightarrow_{\varrho_i, m_i} H_i$ via $\varrho_i = \varrho *_{E_i} \varrho'_i$ are *ϱ-amalgamable* if the matches are *consistent*, i.e., $m_1 \circ k_1 = m_2 \circ k_2 = m$, and, for $i \neq j$, there is a pushout complement L_{i0} of $K \hookrightarrow L \hookrightarrow^{k_i} L_i$ as in Definition 10, and there is a morphism $d_{ij} \colon L_{i0} \to D_j$ such that $b_j \circ d_{ij} = m_i \circ l_{i0}$ and $c_j \circ d_{ij} \models \text{ac}_{L_{i0}}$.



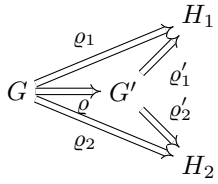
Remark 6. The definition of amalgamable direct transformations generalizes the one of parallel independent ones by requiring the existence of morphisms $L_{i0} \to D_j$ instead of morphisms $L_i \to D_j$.

Example 11. In Fig. 15, the amalgamated transformation $G \Rightarrow_{\varrho, m} H$ is shown which applies the amalgamated rule ϱ to the graph G. The both processes with a flag waiting for one resource enable the second, previously disabled resource and the upper process gets the turn variable.

Fact 11 (parallel independence implies init**-amalgamability).** Parallel independence of direct transformations $G \Rightarrow_{\varrho_i, m_i} H_i$ implies init-amalgamability of $G \Rightarrow_{\varrho_i, m_i} H_i$ where init-amalgamability means ϱ-amalgamability with $\varrho = \text{init} = \langle I \hookleftarrow I \hookrightarrow I \rangle$.

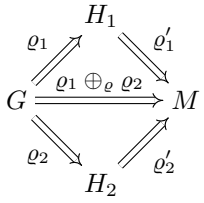
Proof. Let $G \Rightarrow_{\varrho_i, m_i} H_i$ be parallel independent, i.e., there are morphisms $d_{ij} \colon L_i \to D_j$ such that $b_j \circ d_{ij} = m_i$ and $c_j \circ d_{ij} \models \text{ac}_{L_i}$. For the initial rule $\varrho = \text{init}$, we have $L_{i0} = L_i$, $l_{i0} = \text{id}$, $\text{ac}_{L_{i0}} = \text{ac}_{L_i}$, $G \cong G'$, $D_j = D'_j$, and $b'_j = b_j$. Thus, there are morphisms $d_{ij} \colon L_{i0} \to D'_j$ such that $b'_j \circ d_{ij} = b_j \circ d_{ij} = m_i = m_i \circ l_{i0}$ and $c_j \circ d_{ij} \models \text{ac}_{L_{i0}}$. Consequently, $G \Rightarrow_{\varrho_i, m_i} H_i$ is init-amalgamable. □

Lemma 6 (amalgamability implies parallel independence (Golas et al. 2012)). For a common subrule ϱ of ϱ_1 and ϱ_2, ϱ-amalgamability of direct transformations $G \Rightarrow_{\varrho_i,m_i} H_i$ via $\varrho_i = \varrho *_{E_i} \varrho_i'$ implies parallel independence of the direct transformations $G' \Rightarrow_{\varrho_i',m_i'} H_i$ where $G \Rightarrow_{\varrho,m} G'$ and $m = m_i \circ k_i$ for $i = 1, 2$.



Now we present an Amalgamation Theorem for rules with application conditions generalizing the well-known Amalgamation Theorem for rules without application conditions (Boehm et al. 1987) and specializing the Multi-Amalgamation Theorem (Golas et al. 2012) to the case of the amalgamation of two rules.

Theorem 5 (Amalgamation). Let $\varrho' = \varrho_1 \oplus_\varrho \varrho_2$ and $\varrho' = \varrho_i *_{E_i'} \varrho_i'$ for $i = 1, 2$. Given ϱ-amalgamable direct transformations $G \Rightarrow_{\varrho_i,m_i} H_i$, then there is a ϱ-amalgamated transformation $G \Rightarrow_{\varrho',m'} M$ and, for $i = 1, 2$, a direct transformation $H_i \Rightarrow_{\varrho_i'} M$ via ϱ_i' such that $G \Rightarrow_{\varrho_i,m_i} H_i \Rightarrow_{\varrho_i'} M$ is a decomposition of $G \Rightarrow_{\varrho',m'} M$. Given a ϱ-amalgamated direct transformation $G \Rightarrow_{\varrho',m'} M$, then, for $i = 1, 2$, there is a transformation $G \Rightarrow_{\varrho_i,m_i} H_i \Rightarrow_{\varrho_i'} M$ such that the direct transformations $G \Rightarrow_{\varrho_i,m_i} H_i$ are ϱ-amalgamable.



Proof. The Amalgamation Theorem follows immediately from the Multi-Amalgamation Theorem in (Golas et al. 2012) for case $n = 2$. □

6. Related Work

In the following, we present some related work.

Regulated string, term, and graph rewriting. In standard graph transformation (Ehrig 1979), as in standard string rewriting (Salomaa 1973) and standard term rewriting (Baader and Nipkow 1998), a rule can always be applied to a graph if a match is found. However, there are many situations where we would only want to apply a rule if certain conditions are met. The approach to restrict the applicability of rules in graph transformation, even if, superficially, may look similar to the approaches used in string and term rewriting; for term rewriting, the approach is actually very different.

Regulated string rewriting. In string rewriting, there are several approaches for regulated rewriting (Salomaa 1973; Dassow and Păun 1989), e.g., matrix, programmed,

and random context rewriting. There are various applications of formal language theory where context-free grammars are not enough, thus motivating the introduction of regulated (context-free) grammars. Moreover, there are several other applications of regulated rewriting, e.g. relationships with programming languages, regulated rewriting and Petri nets, and modelling of economic processes. Context-sensitive string rewriting (Salomaa 1973), random context rewriting (Dassow and Păun 1989), and string rewriting with local and global context conditions (Csuhaj-Varjú 1993) correspond to context-free graph transformation with positive application conditions.

Conditional term rewriting. In term rewriting, we use conditional rules (Baader and Nipkow 1998), where the conditions have a logical (or operational) nature. Typically, conditions are lists of equations that must be satisfied for the given match, where its satisfaction is checked by term rewriting (usually, checking if the terms of each equation can be rewritten into a common term). This means that the process to see if a rule can be applied to a given term is recursive: to check applicability of a rule, we have to evaluate its conditions, which means applying other rules. Actually, in the general case, applicability of conditional rules is undecidable. Moreover, this recursivity causes various difficulties when trying to extend some results for standard term rewriting to conditional term rewriting.

Local & non-local graph conditions. In graph transformation, we restrict the applicability of rules using application conditions, which essentially have a syntactic nature. In particular, we check the existence (or non-existence) of a given structure that includes the matching. This means that checking application conditions is essentially a matching problem. This is possibly one reason why we are able to extend all the fundamental results of standard graph transformation to this case. Finite nested conditions are expressively equivalent to first-order formulas and local properties (Habel and Pennemann 2009). Non-local properties like there exists a path, is connected, and is cycle-free are not expressible by finite nested conditions, but by finite HR^+ conditions (Habel and Radke 2010), i.e., a finite nested conditions with variables where the variables are place-holders for graphs and the graphs are generated by a hyperedge replacement (HR) system. (Node-)Counting monadic second-order formulas can be transformed into finite HR^+ conditions, the reverse direction is not clear.

Local Church-Rosser, parallelism, confluence for left-linear rules. Adhesive categories provide an abstract setting for the double-pushout approach to rewriting, generalizing classical approaches to graph transformation. Fundamental results about parallelism and confluence, including the Local Church-Rosser Theorem, can be proven in adhesive categories, provided that one restricts to linear rules i.e, rules $\langle L \xleftarrow{l} K \xrightarrow{r} R \rangle$ with l mono and r arbitrary. (Baldan et al. 2011) identify a class of categories, including most adhesive categories used in rewriting, where those same results can be proven in the presence of rules that are merely left-linear, i.e., rules which can merge different parts of a rewritten object. Such rules naturally emerge, e.g., when using graphical encodings for modelling the operational semantics of process calculi.

Local Church-Rosser, termination & confluence. Graph transformation has learnt from term rewriting: The Church-Rosser and Confluence Theorems were origi-

nally developed for term rewriting. Checking local confluence for term rewriting is based on the essential technique for analyzing critical pairs (Knuth and Bendix 1970) and makes use of powerful techniques available for checking termination. If termination is ensured, the local (and global) confluence of the system is shown by checking for all critical pairs. If the system is not confluent, one may apply the (Knuth-Bendix) completion procedures and may try to transform the system into a confluent one by converting all non-confluent critical pairs into rewrite rules (Baader and Nipkow 1998). Checking local confluence for graph transformation (without application conditions), the check is similar (Plump 2005; Ehrig et al. 2006b) although, in this case, the test only provides a sufficient condition. The reason is that local confluence for graph transformation systems is undecidable, even for terminating systems (Plump 2005).

Weakest preconditions & proof systems. Nested graph conditions are used in the verification of graph programs: graph programs (Habel and Plump 2001) generalize the notions of programs on linear structures (Dijkstra 1976) to graphs. As pre- and postconditions of graph programs, (extensions of) nested graph conditions are used. A well-known method for showing the correctness of a program with respect to a pre- and a postcondition (Dijkstra 1976) is to construct a weakest precondition of the program relative to the postcondition and to prove that the precondition implies the weakest precondition. (Habel et al. 2006) use the framework of graphs to construct weakest preconditions for graph programs and (Pennemann 2009) uses his algorithm for approximating the satisfiability problem and his resolution-like theorem prover for graph conditions for trying prove that the precondition implies the weakest precondition. A well-known method for verifying the partial correctness of a program with respect to a pre- and a postcondition (Hoare 1969) is to give a proof system and to show its soundness with respect to the operational semantics of the program. (Poskitt and Plump 2012) use the framework of graphs for verifying the partial correctness of a graph program in the graph programming language GP and show the soundness with respect to the operational semantics of GP.

Weakest preconditions & local confluence. (Bruggink et al. 2011) enrich the formalism of reactive systems by the notion of nested application conditions from graph transformation systems to reactive systems and show that some constructions for graph transformation systems (such as computing weakest preconditions and strongest postconditions and showing Local Confluence by means of critical pair analysis) can be done elegantly in the more general setting.

Model transformation. Negative application conditions, and more general, nested application conditions, are a key ingredient for many model transformations based on graph transformation. The concept of negative application conditions is commonly used in (Ehrig et al 2009a), to define expressive model transformations and to allow the modeler to specify complex model transformations. Currently, the authors are working on the extension of model transformations based on triple graph grammars to the more general nested applications.

OCL constraints. Nested graph conditions are used extremely for specifications, for instance in (UML) model transformations. Restricted OCL constraints (Winkelmann et al. 2008; Ehrig et al. 2009) can be translated to equivalent local graph constraints like the existence or non-existence of certain structures (like nodes and edges or subgraphs)

in an instance graph (positive ones have to be checked after the generation of a meta model instance, negative graph constraints can be checked during the generation) and, by transformation $\mathcal{A}$ in (Habel and Pennemann 2009), graph constraints can be transformed into equivalent application conditions for the corresponding rules. (Note that graph constraints equal application conditions over the empty graph.)

7. Conclusion

In this paper, we have presented the well-known Local Church-Rosser, Parallelism, Concurrency, and Amalgamation Theorems for rules with nested application conditions in the framework of $\mathcal{M}$-adhesive categories. The proofs for transformation systems with nested application conditions are based on the corresponding theorems for transformation systems without application conditions (Ehrig et al. 2006b) and two Shift-Lemmata, saying that application conditions can be shifted over morphisms and rules. Shift-Lemma 2 requires $\mathcal{E}'$-$\mathcal{M}$ pair factorization, the Parallelism Theorem additionally requires binary coproducts, and the Amalgamation Theorem additionally requires initial pushouts over $\mathcal{M}$-morphisms (Golas et al. 2012).

theorem	category	additional requirements
Local Church Rosser	$\mathcal{M}$-adhesive	$\mathcal{E}'$-$\mathcal{M}$ pair factorization
Parallelism	$\mathcal{M}$-adhesive	$\mathcal{E}'$-$\mathcal{M}$ pair fact. & binary coproducts
Concurrency	$\mathcal{M}$-adhesive	$\mathcal{E}'$-$\mathcal{M}$ pair factorization
Amalgamation	$\mathcal{M}$-adhesive	$\mathcal{E}'$-$\mathcal{M}$ pair fact. & initial pushouts over $\mathcal{M}$

In (Golas et al. 2012), a Multi-Amalgamation Theorem for nested application conditions in the framework of $\mathcal{M}$-adhesive categories is given. It generalizes our Amalgamation Theorem to the case of $n \geq 2$ amalgamable direct transformations; Theorem 4 (existence of a remainder) requires $\mathcal{E}'$-$\mathcal{M}$ pair factorization and initial pushouts over $\mathcal{M}$. In (Ehrig et al. 2012), part 2 of this paper, the Embedding and Local Confluence Theorems for nested application conditions in the framework of $\mathcal{M}$-adhesive categories are given; the results require $\mathcal{E}'$-$\mathcal{M}$ pair factorization and initial pushouts over $\mathcal{M}$-morphisms. By the hierarchies of adhesive categories (graph $\Rightarrow$ high-level $\Rightarrow$ weak adhesive HLR $\Rightarrow$ $\mathcal{M}$-adhesive) and application conditions (no $\Rightarrow$ negative $\Rightarrow$ nested), we obtain all results for all these types of categories and application conditions.

Concur	no	negative	nested
graph	$\surd$	$\surd$	$\boxed{\surd}$
high-level	$\surd$	$\surd$	$\boxed{\surd}$
weak adhesive HLR	$\surd$	$\surd$	$\boxed{\surd}$
$\mathcal{M}$-adhesive	$\boxed{\surd}$	$\boxed{\surd}$	$\boxed{\surd}$

Amalg	no	negative	nested
	$\surd$	$\boxed{\surd}$	$\boxed{\surd}$
	$\boxed{\surd}$	$\boxed{\surd}$	$\boxed{\surd}$
	$\boxed{\surd}$	$\boxed{\surd}$	$\boxed{\surd}$
	$\boxed{\surd}$	$\boxed{\surd}$	$\boxed{\surd}$

The Local Church-Rosser, Parallelism, and Concurrency Theorems (Concur) known for weak adhesive HLR transformations systems with negative application conditions (Lambers 2010), marked above by $\surd$, also hold for proper $\mathcal{M}$-adhesive transformations systems

with proper nested application conditions, marked by $\boxed{\surd}$. The Amalgamation Theorem (Amalg) known only for graph transformations systems without application conditions (Boehm et al. 1987) holds for all $\mathcal{M}$-adhesive transformations systems with nested application conditions.

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

References

Adámek, J., Herrlich, H. and Strecker, G. (1990) *Abstract and Concrete Categories.* John Wiley.

Arbib, M.A. and Manes, E.G. (1975) *Arrows, Structures, and Functors.* Academic Press.

Baader, F. and Nipkow, T. (1998) *Term Rewriting and All That.* Cambridge University Press, 1998.

Baldan, P., Gadducci, F. and Sobocinski, P. (2011) Adhesivity is not enough: Local Church-Rosser revisited. In *Mathematical Foundations of Computer Science (MFCS 2011), Lecture Notes in Computer Science* **6907**, 48–59.

Biermann, E. , Ehrig, H., Ermel, C., Golas, U. and Taentzer, G. (2010) Parallel independence of amalgamated graph transformations applied to model transformation. In *Graph Transformations and Model-Driven Engineering, Lecture Notes in Computer Science* **5765**, 121–140.

Paul Boehm, P., Fonio, H.-R., and Habel, A. (1987) Amalgamation of graph transformations: A synchronization mechanism. *Journal of Computer and System Sciences*, **34**, 377–408.

Bruggink, H.J.S., Cauderlier, R., Hülsbusch, M., and König, B. (2011) Conditional reactive systems. *Foundations of Software Thechnology and Theoretical Computer Science (FSTTCS 2011)*, 191-203.

Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., and Löwe, M. (1997) Algebraic approaches to graph transformation. Part I: Basic concepts and double pushout approach. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume **1**, 163–245. World Scientific.

Corradini, A., Rossi, F. and Parisi-Presicce F. (1991). Logic programming as hypergraph rewriting. In *Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT'91), Lecture Notes in Computer Science* **493**, 275–295.

Castellani, I. and Montanari U. (1983). Graph grammars for distributed systems. In *Graph Grammars and Their Application to Computer Science, Lecture Notes in Computer Science* **153**, 20–38.

Courcelle, B. (1997) The expression of graph properties and graph transformations in monadic second-order logic. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume **1**, 313–400. World Scientific.

Csuhaj-Varjú, E. (1993) On grammars with local and global context conditions. *Int. Journal of Computer Mathematics* **47**:17–27.

Dassow, J. and Păun, G. (1989) *Regulated Rewriting in Formal Language Theory, EATCS Monographs on Theoretical Computer Science*, volume **18**. Springer-Verlag.

Degano, P. and Montanari, U. (1987). A model of distributed systems based of graph rewriting. *Journal of the ACM* **34**:411–449.

Dijkstra, E.W. (1965) Solution of a Problem in Concurrent Programming Control. *Communications of the ACM*, **8**:569. *A Discipline of Programming.* Prentice-Hall.

Dijkstra, E.W. (1976) *A Discipline of Programming.* Prentice-Hall.

Ehrig, H. (1979) Introduction to the algebraic theory of graph grammars. In *Graph-Grammars and Their Application to Computer Science and Biology, Lecture Notes in Computer Science* **73**, 1–69.

Ehrig, H., Ehrig, K., Habel, A., and Pennemann, K.-H. (2006) Theory of constraints and application conditions: From graphs to high-level structures. *Fundamenta Informaticae*, **74** (1), 135–166.

Ehrig, H., Ehrig, K., Prange, U., and Taentzer, G. (2006) *Fundamentals of Algebraic Graph Transformation.* EATCS Monographs of Theoretical Computer Science, Springer-Verlag.

Ehrig, H., Engels, G., Kreowski, H.-J., and Rozenberg, G. (1999) editors. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume **2**: Applications, Languages and Tools. World Scientific.

Ehrig, H., Golas, U. and Hermann, F. (2010) Categorical Frameworks for Graph Transformation and HLR Systems based on the DPO Approach. In *Bulletin of the EATCS* **112**, 111-121.

Ehrig, H., Golas, U., Habel, A., Lambers, L., and Orejas, F. (2012) $\mathcal{M}$-Adhesive Transformation Systems with Nested Application Conditions. Part 2: Embedding, Critical Pairs and Local Confluence. *Fundamenta Informaticae.* To appear.

Ehrig, H. and Habel, A. (1986) Graph grammars with application conditions. In G. Rozenberg and A. Salomaa, editors, *The Book of L*, 87–100. Springer-Verlag.

Ehrig, H., Habel, A., Kreowski, H.-J., and Parisi-Presicce, F. (1991) Parallelism and concurrency in high level replacement systems. *Mathematical Structures in Computer Science* **1**, 361–404.

Ehrig, H., Habel, A., and Lambers, L. (2010) Parallelism and concurrency theorems for rules with nested application conditions. In *Electronic Communications of the EASST* **26**.

Ehrig, H., Habel, A., Padberg, J., and Prange, U. (2006) Adhesive high-level replacement systems: A new categorical framework for graph transformation. *Fundamenta Informaticae* **74**, 1–29.

Ehrig, H., Habel, A., and Rosen, B. K. (1986). Concurrent transformations of relational structures. *Fundamenta Informaticae* **IX**:13–50.

Ehrig, H., Hermann, F. and Sartorius, C. (2009) Completeness and Correctness of Model Transformations based on Triple Graph Grammars with Negative Application Conditions. *Electronic Communications of the EASST* **18**.

Ehrig, H. and Kreowski, H.-J. (1980). Applications of graph grammar theory to consistency, synchronization and scheduling in data base systems. *Information Systems* **5**:225–238.

Ehrig, H., Kreowski, H.-J., Montanari, U., and Rozenberg, G. editors.(1999) *Handbook of Graph Grammars and Computing by Graph Transformation*, volume **3**: Concurrency, Parallelism, and Distribution. World Scientific.

Ehrig, H. and Parisi-Presicce, F. (1992) High-level-replacement systems for equational algebraic specifications. In *Algebraic and Logic Programming, Third International Conference, Proceedings, Lecture Notes in Computer Science* **632**, 3–20.

Ehrig, H., Pfender, M. and Schneider H.-J. (1973). Graph grammars: An algebraic approach. In*Proc. 14th Annual IEEE Symposium on Switching and Automata Theory*, Iowa City, 167–180.

Ehrig, H. and Rosen, B. (1980). Parallelism and concurrency of graph manipulations. *Theoretical Computer Science* **11**, 247–275, 1980.

Ehrig, K., Küster, J. M. and Taentzer, G . (2009) Generating instance models from meta models. *Software and System Modeling* **8**(4):479–500.

Golas, U. (2011) Analysis and correctness of algebraic graph and model transformation. Vieweg+Teubner Research.

Golas, U., Habel, A., and Ehrig, H. (2012) Multi-Amalgamation in $\mathcal{M}$-Adhesive Categories. *Mathematical Structures in Computer Science.* (this volume).

Habel, A., Heckel, R., and Taentzer, G. (1996) Graph grammars with negative application conditions. *Fundamenta Informaticae*, **26**, 287–313.

Habel, A. and Pennemann, K.-H. (2009) Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science* **19**, 245–296.

Habel, A., Pennemann, K-H., and Rensink, A. (2006) Weakest preconditions for high-level programs. In *Graph Transformations (ICGT 2006), Lecture Notes in Computer Science* **4178**, 445–460.

Habel, A. and Plump, D. (2001) Computational completeness of programming languages based on graph transformation. In *Proc. Foundations of Software Science and Computation Structures (FOSSACS 2001), Lecture Notes in Computer Science* **2030**, 230–245.

Habel, A. and Radke, H. (2010). Expressiveness of graph conditions with variables. *Electronic Communications of the EASST* **30**.

Heckel, R., Llabrés, M., Ehrig, H., and Orejas, F. (2002) Concurrency and loose semantics of open graph transformation systems. *Mathematical Structures in Computer Science* **12**(4):349–376.

Heckel, R. and Wagner, A. (1995) Ensuring consistency of conditional graph grammars — a constructive approach. In *Proc. Workshop on Graph Rewriting and Computation (SEGRAGRA'95), Electronic Notes in Theoretical Computer Science* **2**, 95–104.

Heindel, T. (2010) Hereditary Pushouts Reconsidered. In *Graph Transformations (ICGT'10), Lecture Notes in Computer Science* **6372**, 250–265.

Hoare, C.A.R. (1969) An axiomatic basis for computer programming. *Communications of the ACM*, **12**:576–580, 583.

Knuth, D.E. and Bendix, P.B . (1970) Simple word problems in universal algebras. In *Computational Problems in Abstract Algebras*, 263–297. Pergamon Press.

Koch, M., Mancini, L.V., and Parisi-Presicce, F. (2005) Graph-based specification of access control policies. *Journal of Computer and System Sciences*, **71**, 1–33.

Kreowski, H.-J. (1977) *Manipulationen von Graphmanipulationen.* PhD thesis, Technical University of Berlin.

Lack, S. and Sobociński P. (2004) Adhesive categories. In *Foundations of Software Science and Computation Structures (FOSSACS'04), Lecture Notes in Computer Science*, **2987**, 273–288.

Lack, S. and Sobociński P. (2005) Adhesive and quasiadhesive categories. *Theoretical Informatics and Application*, **39** (2):511–546.

Lambers, L. (2010). *Certifying Rule-Based Models using Graph Transformation.* PhD thesis, Technical University of Berlin, 2010.

Mahr, B. and Wilharm, A. (1982). Graph grammars as a tool for description in computer processed control: A case study. In *Graph-Theoretic Concepts in Computer Science*, 165–176. Hanser Verlag, München.

Parisi-Presicce, F. (1989) Modular system design applying graph grammar techniques. In*Automata, Languages and Programming (ICALP89), Lecture Notes in Computer Science* **372**, 621–636.

Pennemann, K.-H. (2009) *Development of Correct Graph Transformation Systems.* PhD thesis, Universität Oldenburg.

Plump, D. (2005) Confluence of graph transformation revisited. In *Processes, Terms and Cycles: Steps on the Road to Infinity: Essays Dedicated to Jan Willem Klop on the Occasion of His 60th Birthday, Lecture Notes in Computer Science* **3838**, 280–308.

Poskitt, C.M. and Plump, D. (2012) Hoare-style verification of graph programs. *Fundamenta Informaticae*. To appear.

Rensink, A. (2004) Representing first-order logic by graphs. In *Graph Transformations (ICGT'04), Lecture Notes in Computer Science* **3256**, 319–335.

Ribeiro, L. (1996). A telephone's system specification using graph grammars. Technical report 96-23, Technical University of Berlin.

Rosen, B.K. (1975). A Church-Rosser theorem for graph grammars (announcement). *SIGACT News*, 7(3):26–31, 1975.

Rozenberg, G. (1997), editor. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume **1**: Foundations. World Scientific.

Salomaa, A. (1973) *Formal Languages*. Academic Press, New York.

Taentzer, G., Koch, M., Fischer, I., and Volle, V. (1999). Distributed graph transformation with application to visual design of distributed systems. In *Handbook of Graph Grammars and Computing by Graph Transformation*, volume **3**, 269–340. World Scientific.

Winkelmann, J., Taentzer, G., Ehrig, K., and Küster, J.M. (2008) Translation of restricted OCL constraints into graph constraints for generating meta model instances by graph grammars. In *GT-VMT 2006, Electronic Notes in Theoretical Computer Science* **211**, 159–170.