Conditional Reactive Systems*

H. J. Sander Bruggink[1], Raphaël Cauderlier[2], Mathias Hülsbusch[1], and Barbara König[1]

1 **Universität Duisburg-Essen, Germany**
{sander.bruggink,mathias.huelsbusch,barbara_koenig}@uni-due.de
2 **ENS de Cachan, France**
rcauderl@dptinfo.ens-cachan.fr

Abstract

We lift the notion of nested application conditions from graph transformation systems to the general categorical setting of reactive systems as defined by Leifer and Milner. This serves two purposes: first, we enrich the formalism of reactive systems by adding application conditions for rules; second, it turns out 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 very elegantly in the more general setting.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases reactive systems, graph transformation, graph logic, Hoare triples, critical pair analysis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2011.191

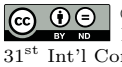
1 Introduction

Reactive Systems were introduced in [12, 11] to obtain a framework for deriving bisimulation congruences. They provide a general categorical setting for modelling abstract rewriting: both graph transformation systems and process calculi can be seen as special cases of reactive systems.

One of the main possibilities to control reductions is to fix a set of reactive contexts in which reductions are possible. However, this gives little control for complex specifications for which it is necessary to describe very precisely under which conditions a rule is applicable. Graph transformation systems provide the notion of negative application conditions or, more generally, nested application conditions [18, 6, 2, 7]. Nested application conditions are used extensively for specifications, for instance in (UML) model transformations.

Here we lift the idea of (application) conditions from the setting of graph transformation to reactive systems. This serves two purposes: first, we enrich the formalism of reactive systems by adding application conditions for rules. Second, it turns out that some constructions which are fairly cumbersome in the case of graph transformation (such as computation of weakest preconditions, strongest postconditions and critical pairs) become very elegant in this more abstract high-level setting. Interestingly it turns out that idem pushout squares (IPOs) [12], needed for label derivation in reactive systems, have a natural interpretation on application conditions and form the basis of an important construction, called shift or partial evaluation.

* Supported by the DFG project Behaviour-GT.



© H.J.S. Bruggink, R. Cauderlier, M. Hülsbusch, and B. König;
licensed under Creative Commons License ND

31st Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011).
Editors: Supratik Chakraborty, Amit Kumar; pp. 191–203

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The paper is structured as follows: in Section 2 we introduce conditional reactive systems and boolean operations on conditions. Furthermore we briefly introduce adhesive categories and cospan categories. We then show in Section 3 that if conditions are interpreted in the base category, they are equivalent in expressiveness to a first-order logic on subobjects [1]. Section 4 introduces representative squares, a generalization of IPOs, and uses them in Section 5 in order to define logical operations such as shift and quantification on conditions. We study the properties of these operations, especially a characterization via adjunctions. Then in Section 6 we investigate applications, such as Hoare triples, weakest preconditions, strongest postconditions and a critical pair lemma for proving confluence.

We assume some basic knowledge of category theory.

2 Reactive Systems and Conditions

We now define the notion of reactive systems, first introduced in [12, 11] for label derivation and the definition of bisimulation congruences.

▶ **Definition 1** (Reactive System). Let $\mathbf{C}$ be a category with a distinguished object 0 (not necessarily initial). A *reactive system rule* is a pair $R = (\ell, r)$ of arrows – called left-hand side and right-hand side respectively – with $\ell, r \colon 0 \to I$ for some object I. Let $\mathcal{R}$ be a set of rules. We say that an arrow $a \colon 0 \to J$ *reduces* to $b \colon 0 \to J$ with the rules in $\mathcal{R}$ (in symbols: $a \Rightarrow_{\mathcal{R}} b$ or simply $a \Rightarrow b$) if there exists a rule $(r, \ell) \in \mathcal{R}$ with $\ell, r \colon 0 \to I$ and an arrow $c \colon I \to J$ such that $a = \ell\,;c$ and $b = r\,;c$.[1]

An important class of reactive systems can be defined over a base category $\mathbf{D}$ which has all pushouts along monos and in which pushouts preserve monos. Then we define as $\mathbf{C} = \mathit{ILC}(\mathbf{D})$ the category which has as objects the objects of $\mathbf{D}$ and as arrows cospans of the form $A \rightarrowtail^{f} B \leftarrow^{g} C$ (called *input-linear cospans*, because the left arrow f is a mono), where the middle object is taken up to isomorphism. Composition of cospans is performed via pushouts.

As base category $\mathbf{D}$ we will mainly use adhesive categories [10] where pushouts are well-behaved with respect to pullbacks and where, among other properties, monos are preserved by pushouts. Their properties make them suitable for deriving bisimulation congruences [20]. Here we need adhesive categories only in a limited way and the results can be stated and understood without the exact definition, which we therefore do not give. In some of the examples we will use the adhesive category $\mathbf{Graph_{fin}}$ which has finite graphs (with node and edge labels) as objects and graph morphisms as arrows. Reactive systems over $\mathit{ILC}(\mathbf{Graph_{fin}})$ coincide exactly with DPO graph transformation systems with injective matches (see [20]).

We will now define conditions, similar to the presentation in [18, 6], as tree-like structures, where nodes are annotated with quantifiers and objects and edges are annotated with arrows.

▶ **Definition 2** (Conditions). Let $\mathbf{C}$ be a category. A *condition* (in $\mathbf{C}$) is a triple $\mathcal{A} = (A, \mathcal{Q}, S)$ where

- A is an object of $\mathbf{C}$ (called the root object of $\mathcal{A}$ or $\mathrm{RO}(\mathcal{A})$),
- $\mathcal{Q}$ is a quantifier (either $\forall$ or $\exists$) and
- S is a set of pairs $(\mathcal{A}', f)$ such that $\mathcal{A}'$ is a condition and $f \colon A \to \mathrm{RO}(\mathcal{A}')$ is a $\mathbf{C}$-arrow.

[1] For arrows $f \colon A \to B$ and $g \colon B \to C$ we use the notation $f\,;g$ for their composition, that is $f\,;g \colon A \to C$.

The pair $(\mathcal{A}', f)$ will be denoted, by a slight abuse of notation, by $A \xrightarrow{f} \mathcal{A}'$. A condition $\mathcal{A}$ can be viewed as a tree, with $\mathrm{RO}(\mathcal{A})$ as the root and edges labelled with arrows. Note that in most cases we will require that S is a finite set, so that the trees are finitely branching.

▶ **Definition 3.** For all conditions $\mathcal{A}$, all objects C and all arrows $c\colon \mathrm{RO}(\mathcal{A}) \to C$ we define a satisfaction relation as follows:

$c \models (A, \forall, S)$ iff for every $(A \xrightarrow{f} \mathcal{A}') \in S$ and every arrow $\alpha\colon \mathrm{RO}(\mathcal{A}') \to C$ such that $f\,;\alpha = c$ we have $\alpha \models \mathcal{A}'$

$c \models (A, \exists, S)$ iff for some $(A \xrightarrow{f} \mathcal{A}') \in S$ there is an arrow $\alpha\colon \mathrm{RO}(\mathcal{A}') \to C$ with $f\,;\alpha = c$ and $\alpha \models \mathcal{A}'$

Now, given a reactive system over $\mathbf{C}$, it is natural to associate conditions with the target object of left-hand and right-hand sides and to interpret them on the contexts.

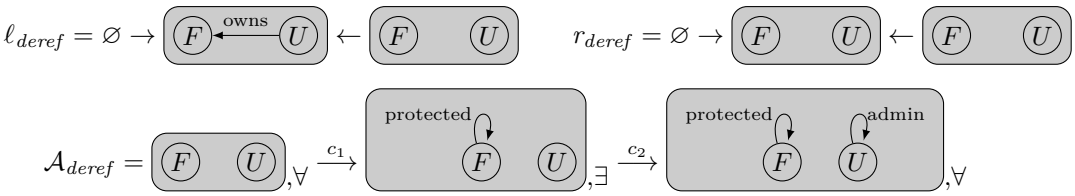
▶ **Definition 4** (Rules with Application Conditions)**.** Let $\mathbf{C}$ be a category with a distinguished object 0. A *rule with application condition* is a triple $(\ell, r, \mathcal{A})$ where $\ell, r\colon 0 \to I$ and $\mathcal{A}$ is a condition with root object I. We say that the rule is applicable to $a\colon 0 \to J$ whenever $a = \ell\,;c$ for some $c\colon I \to J$ such that $c \models \mathcal{A}$. The result of the rule application is $r\,;c$ as in Definition 1. Again we denote by $\Rightarrow_{\mathcal{R}}$ the rewriting relation induced by a set $\mathcal{R}$ of rules with application conditions.

▶ **Example 5.** For this example we are working in the category $ILC(\mathbf{Graph_{fin}})$.

Consider the following situation: In a file system a user can only dereference a file if it is owned by the user and the file is not marked as protected. To dereference protected files, the user must have administrator rights. Files which are no longer owned by any user are later removed by a garbage-collecting process.

Files are modelled by F-labeled nodes and users by U-labeled nodes. The fact that a user owns a files is represented by an edge which is labeled *owns* from the user to the file node. Protected files and users with administrator rights are designated with loops labeled *protected* and *admin*, respectively.

The derefencing is modelled by the rule $R_{deref} = (\ell_{deref}, r_{deref}, \mathcal{A}_{deref})$ with application condition $\mathcal{A}_{deref}$. The components are:



where $\varnothing$ is the empty graph and c_1 and c_2 are cospans where the right leg is the identity and the left morphism is induced by the labels. The condition expresses, that for all matches such that a *protected*-loop is connected to the file node, a *admin*-loop is connected to the user node. This condition is equivalent to the description above. Applying the rule has the effect that the *owns*-label is removed.

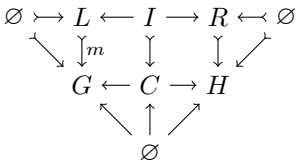
The condition for the garbage collection is:

$$\mathcal{B} = \varnothing, \exists \xrightarrow{c} \varnothing, \forall \quad \text{with } c = \varnothing \to \boxed{F} \leftarrow \varnothing,$$

which is true on exactly those cospans (with source object $\varnothing$) with isolated F-nodes in the center graph. In standard nested application conditions this requirement can be specified

only by saying that the node is not connected to any edge, which means quantifying over all edge labels. This does not yield a finite condition in the case of infinitely many edge labels [9].

Note that in graph transformation, nested application conditions are defined and handled in a slightly different way: they are defined in the base category instead of the cospan category and they are attached to the object L of the left-hand side, instead of the interface object I. In more detail: assume that $\mathbf{C} = ILC(\mathbf{D})$, hence rules have the form $\ell\colon \varnothing \rightarrowtail L \leftarrow I, r\colon \varnothing \rightarrowtail R \leftarrow I$ and we assume that contexts are of the form $c\colon I \rightarrowtail C \leftarrow \varnothing$. Note that the empty graph $\varnothing$ acts as the distinguished object 0. The conditions $a = \ell\,;c$, $b = r\,;c$ lead to a double-pushout diagram as shown on the right.



Now, a nested application condition in the sense of [6] is defined as a condition in $\mathbf{D}$ with root object L and it is evaluated on the match m. As above, a rule may only be applied if the condition is satisfied. This has some consequences. Most importantly, such nested application conditions are equal in expressivity to a first-order logic (see [18] for the case of graphs and Section 3 for the general case), whereas conditions on cospans are slightly more expressive: The paper [9] shows that every condition $\mathcal{A}$ in $\mathbf{D}$, consisting only of monos, can be translated into a condition $\mathcal{A}'$ in $\mathbf{C}$ such that $c\colon A \rightarrowtail B$ satisfies $\mathcal{A}$ iff $c\colon A \rightarrowtail B \leftarrow_{\mathrm{id}_B} B$ satisfies $\mathcal{A}'$. A translation in the other direction is in general impossible. We can always, even in the presence of infinitely many edge labels, specify that an isolated node exists; see also Example 5.

We feel that viewing conditions in the cospan category $\mathbf{C} = ILC(\mathbf{D})$ gives us an additional and very helpful layer of abstraction that simplifies many of the constructions to come. In Section 6 we will compare weakest preconditions and strongest postconditions as well as critical pair analysis in our setting with analogous notions in graph transformation and obtain constructions that are much simpler and straightforward. Hence we believe that switching to this higher level of abstraction is of great benefit.

We will in the following write $\mathcal{A} \models \mathcal{B}$ for two conditions $\mathcal{A}$, $\mathcal{B}$ with the same root object, whenever every arrow satisfying $\mathcal{A}$ satisfies $\mathcal{B}$ as well. We write $\mathcal{A} \equiv \mathcal{B}$ iff $\mathcal{A} \models \mathcal{B}$ and $\mathcal{B} \models \mathcal{A}$. We now define the usual boolean operations on conditions.

▶ **Definition 6** (Boolean Operations on Conditions)**.** We define operations on conditions as follows:

Constants: For an object A define $\mathit{false}_A := (A, \exists, \varnothing)$, $\mathit{true}_A := (A, \forall, \varnothing)$.

Negation: For a condition of the form $(A, \mathcal{Q}, S)$ with $\mathcal{Q} \in \{\forall, \exists\}$ we define:

$\neg(A, \forall, S) := (A, \exists, \{A \xrightarrow{f} \neg\mathcal{A}' \mid (A \xrightarrow{f} \mathcal{A}') \in S\})$

$\neg(A, \exists, S) := (A, \forall, \{A \xrightarrow{f} \neg\mathcal{A}' \mid (A \xrightarrow{f} \mathcal{A}') \in S\})$

Conjunction und Disjunction: For two conditions $\mathcal{A}, \mathcal{B}$ with root object C we define:

$\mathcal{A} \wedge \mathcal{B} := (C, \forall, \{C \xrightarrow{id_C} \mathcal{A}, C \xrightarrow{id_C} \mathcal{B}\})$

$\mathcal{A} \vee \mathcal{B} := (C, \exists, \{C \xrightarrow{id_C} \mathcal{A}, C \xrightarrow{id_C} \mathcal{B}\})$

These definitions are easily generalized to conjunctions and disjunctions over infinite sets of conditions.

▶ **Proposition 7** (Boolean Operations on Conditions)**.** *The operations and constants of Definition 6 satisfy the following laws:*

- *No arrow $c\colon A \to B$ satisfies $c \models \mathit{false}_A$ and every arrow $c\colon A \to B$ satisfies $c \models \mathit{true}_A$.*

- *For all conditions* $\mathcal{A}$ *and all arrows* $c\colon \mathrm{RO}(\mathcal{A}) \to B$*, we have* $c \models \neg\mathcal{A} \iff c \not\models \mathcal{A}$.
- *For all conditions* $\mathcal{A}, \mathcal{B}$ *with* $C = \mathrm{RO}(\mathcal{A}) = \mathrm{RO}(\mathcal{B})$ *and all arrows* $c\colon C \to D$ *it holds that* $(c \models \mathcal{A} \wedge \mathcal{B} \iff c \models \mathcal{A}$ *and* $c \models \mathcal{B})$ *and* $(c \models \mathcal{A} \vee \mathcal{B} \iff c \models \mathcal{A}$ *or* $c \models \mathcal{B})$.

3 Comparison to a First-Order Logic on Subobjects

Logic on subobjects [1] is a generalization of monadic second order logic of graphs to the world of categories. Nodes and edges are replaced by subobjects of fixed structure, sets of nodes and sets of edges are replaced by subobjects of arbitrary structure. Here we consider the first-order fragment of the logic on subobjects, which, as shown in [1], instantiates to first-order logic on graphs if we choose $\mathbf{Graph}_{\mathbf{fin}}$ as the underlying category.

In the following we will work in the subcategory $\mathbf{D}$ of an adhesive category $\mathbf{C}$, where $\mathbf{D}$ contains only monos. While this does not make a difference for the logic on subobjects, where we only quantify over monos anyway, for conditions it means that only monos are used in the evaluation, which is a typical restriction. (For an investigation of the effects of evaluating conditions on monos only see [5]. It is shown that under mild conditions both variants of conditions are expressively equivalent.) In addition the category $\mathbf{D}$ has a canonical embedding into $ILC(\mathbf{C})$. We remark again that, as discussed on Page 194, conditions in $ILC(\mathbf{C})$ are in general more expressive than conditions in the base category $\mathbf{D}$.

3.1 Syntax and semantics

We fix a category $\mathbf{C}$. The first-order logic of subobjects consists of *expressions* and *formulae*:

- Expressions are of the form $e = f \fatsemi x$, where x is a variable typed by an object A and f is a mono with codomain A. Expressions represent subobjects restricted by a mono.
- Formulae are generated by the grammar:

  $$\varphi ::= e \sqsubseteq e \mid \varphi \wedge \varphi \mid \neg\varphi \mid (\forall x\colon A)\ \varphi$$

  where x is a variable and A is an object. We use $e_1 = e_2$ as an abbreviation for $(e_1 \sqsubseteq e_2) \wedge (e_2 \sqsubseteq e_1)$. The notations $\varphi_1 \vee \varphi_2$, $\varphi_1 \to \varphi_2$ and $(\exists x\colon A)\ \varphi$ are defined in the usual way.

Let C be an object. A C-valuation maps variables to monos with codomain C (that is, to subobjects of C). We will overload the operator ; to the composition of C-valuations with monos, that is, for a C-valuation η and a mono $c\colon C \rightarrowtail D$, $\eta\,;c$ is a D-valuation defined as: $(\eta\,;c)(x) = \eta(x)\,;c$ for all x in the domain of η.

▶ **Definition 8.** For all objects C and all C-valuations η, we define a modelling relation as follows:

$C, \eta \models (f_1 \fatsemi x_1 \sqsubseteq f_2 \fatsemi x_2)$ iff $(f_1\,;\eta(x_1) \leq f_2\,;\eta(x_2))$.[2]
$C, \eta \models \varphi_1 \wedge \varphi_2$ iff $C, \eta \models \varphi_1$ and $C, \eta \models \varphi_2$.
$C, \eta \models \neg\varphi$ iff $C, \eta \not\models \varphi$.
$C, \eta \models (\forall x\colon T)\ \varphi$ iff for all monos $m\colon T \rightarrowtail C$ we have $C, \eta[x \mapsto m] \models \varphi$

[2] For monos $a\colon A \rightarrowtail T$ and $b\colon B \rightarrowtail T$ we write $a \leq b$ if there exists a mono $c\colon A \rightarrowtail B$ such that $a = c\,;b$.

3.2 From Conditions to the Logic on Subobjects

In this subsection we will translate conditions into the logic on subobjects.

▶ **Definition 9.** We define a translation from conditions to formulae as follows (where the variable x is of type A):

$$[\![(A, \forall, S)]\!](x) := \bigwedge_{(f,\mathcal{A}') \in S} (\forall x' \colon \mathrm{RO}(\mathcal{A}'))\ (f \fatsemi x' = x \ \to\ [\![\mathcal{A}']\!](x'))$$
$$[\![(A, \exists, S)]\!](x) := \bigvee_{(f,\mathcal{A}') \in S} (\exists x' \colon \mathrm{RO}(\mathcal{A}'))\ (f \fatsemi x' = x \ \wedge\ [\![\mathcal{A}']\!](x'))$$

A condition and its translation are equivalent in the following sense:

▶ **Proposition 10.** *If $\mathcal{A}$ is a condition then for all objects C and all monos $c \colon \mathrm{RO}(\mathcal{A}) \rightarrowtail C$ we have $c \models \mathcal{A}$ iff $C, (x \mapsto c) \models [\![\mathcal{A}]\!](x)$.*

3.3 From the Logic on Subobjects to Conditions

The idea behind the translation from formulae of the first-order logic of subobjects to conditions, is to consider all the different ways how the variables can overlap. That is, every time we quantify over a variable, we consider all possible ways how the objects pointed to by the new variable can overlap with the other variables, and build sub-conditions for all the possible overlaps. Formally, the set of overlaps of two objects A, B is defined as follows:

$$\mathrm{Ovl}(A, B) := \{(C, a, b) \mid a \colon A \rightarrowtail C \text{ and } b \colon B \rightarrowtail C \text{ are monos and jointly epi}\}$$

We assume in the following that $\mathrm{Ovl}(A, B)$ contains only one representative for each isomorphism class.

▶ **Definition 11.** Let B be an object and η a B-valuation. We define:

$$[\![f \fatsemi x \sqsubseteq g \fatsemi y]\!]^\eta_B := \begin{cases} \mathit{true}_B & \text{if } f\,;\eta(x) \leq g\,;\eta(y) \\ \mathit{false}_B & \text{if } f\,;\eta(x) \not\leq g\,;\eta(y) \end{cases}$$
$$[\![\varphi_1 \wedge \varphi_2]\!]^\eta_B := [\![\varphi_1]\!]^\eta_B \wedge [\![\varphi_2]\!]^\eta_B$$
$$[\![\neg\varphi]\!]^\eta_B := \neg[\![\varphi]\!]^\eta_B$$
$$[\![(\forall x \colon T)\ \varphi]\!]^\eta_B := \Big(B, \forall, \{B \overset{a}{\rightarrowtail} [\![\varphi]\!]^{\eta_{a,b}}_C \mid (C, a, b) \in \mathrm{Ovl}(T, B)\}\Big)$$
$$\text{where } \eta_{a,b}(y) = \begin{cases} a & \text{if } y = x \\ \eta(y)\,;b & \text{otherwise} \end{cases}$$

Note that the tree produced by Definition 11 is finitely branching (and because of the finite depth of formulas thus finite) if $\mathrm{Ovl}(A, B)$ is finite (up to isomorphism) for all objects A, B. In the category $\mathbf{Graph_{fin}}$ this is the case.

▶ **Proposition 12.** *Let $\mathbf{C}$ be an adhesive category. For any formula φ of the logic on subobjects, any objects B and C, any mono $c \colon B \rightarrowtail C$ and any B-valuation η it holds that $C, \eta\,; c \models \varphi$ if and only if $c \models [\![\varphi]\!]^\eta_B$.*

4 Representative Squares

We will now define the notion of representative squares, which describe representative ways to close a span of arrows. Such squares are intimately related to idem pushouts [12] or borrowed context diagrams [4].

▶ **Definition 13** (Representative classes of squares)**.** A class κ of commuting squares in a category $\mathbf{C}$ is called *representative* if κ satisfies the following property: for every commuting square of $\mathbf{C}$ (such as the one consisting of a, b, c', d' on the left) there exists a square in κ (consisting of a, b, c, d) and an arrow $e\colon M \to N$ which makes the diagram commute (on the right).

For two arrows $a\colon I \to J$, $b\colon I \to K$ we denote by $\kappa(a, b)$ the set of pairs (c, d) of arrows $c\colon J \to M$ and $d\colon K \to M$ such that a, b, c, d form a representative square in κ.

In the following, we fix a representative class κ of squares and we shall call every square in κ *representative*. Also note that the class of all squares of $\mathbf{C}$ is representative. The interesting classes of representative squares, however, have the property that $\kappa(a, b)$ is finite, which means that the constructions described below are effective since the finiteness of the transformed conditions is preserved.

Naturally, in a category with pushouts, pushouts are the most natural candidate for representative squares. Alternatively in adhesive categories they can be replaced by jointly epi squares. In [12] idem pushouts (IPOs) were introduced as a means to close squares in a representative way. (IPOs satisfy more properties than required in Definition 13, but those are not needed here.) A concrete manifestation of IPOs (or rather groupoidal idem pushouts or GIPOs) are borrowed context squares in the category $ILC(\mathbf{D})$, where the base category $\mathbf{D}$ is adhesive [20, 19].

For many categories of interest – such as the category of finite graphs (and graph morphisms) and the category of (input-linear) cospans of finite graphs – we can indeed guarantee a choice of κ such that each set $\kappa(a, b)$ is finite.

5 Shift and Quantification

One central operation is the shift of a condition along an arrow. The name shift is taken from an analogous operation for nested application conditions (see [13]). Intuitively a shift corresponds to a partial evaluation, where we assume that the arrows on which the condition is to be evaluated are of the form $\varphi\,;c$ for a fixed φ.

▶ **Definition 14** (Shift of a Condition)**.** Given a fixed set κ of representative squares, the shift $\mathcal{A}_{\downarrow\varphi}$ of a condition $\mathcal{A} = (A, \mathcal{Q}, S)$ along an arrow $\varphi\colon A \to B$ is inductively defined as follows:

$$\mathcal{A}_{\downarrow\varphi} = (B, \mathcal{Q}, \{(B \xrightarrow{\beta} \mathcal{A}'_{\downarrow\alpha}) \mid (A \xrightarrow{f} \mathcal{A}') \in S,\ (\alpha, \beta) \in \kappa(f, \varphi)\})$$

Note that the shift of a condition depends on the specific representive class of squares chosen. For different representative squares, the constructed condition can be different

(but equivalent). If we require that each set $\kappa(f, \varphi)$ is finite, then every finite condition is transformed into a finite-branching and hence again finite condition.

▶ **Proposition 15** (Shift). *Given two arrows $\varphi\colon A \to B$ and $c\colon B \to C$, and a condition $\mathcal{A}$ with root object A, the following holds:*

$$\varphi\,;c \models \mathcal{A} \iff c \models \mathcal{A}_{\downarrow\varphi}$$

We also define the following two quantification operations on conditions:

▶ **Definition 16** (Quantification). Given a condition $\mathcal{B}$ with root object B and an arrow $\varphi\colon A \to B$ we define for $\mathcal{Q} \in \{\exists, \forall\}$:

$$\mathcal{Q}\varphi.\mathcal{B} = (A, \mathcal{Q}, \{A \xrightarrow{\varphi} \mathcal{B}\})$$

We can view the set $\mathbf{C}_A$ of all conditions over a root object A as a category with an arrow between $\mathcal{A}$ and $\mathcal{B}$ whenever $\mathcal{A} \models \mathcal{B}$. Now the three operations on conditions can be seen as functors between these categories. Furthermore for $\varphi\colon A \to B$ it can be shown that $\exists\varphi\colon \mathbf{C}_B \to \mathbf{C}_A$ is the left adjoint of $\_{\downarrow\varphi}\colon \mathbf{C}_A \to \mathbf{C}_B$ and $\forall\varphi\colon \mathbf{C}_B \to \mathbf{C}_A$ is its right adjoint. These properties can be spelled out as follows.

▶ **Proposition 17** (Adjunction). *Let $\mathcal{A}, \mathcal{B}$ be two conditions with root object A, $\mathcal{C}, \mathcal{D}$ two conditions with root object B and let $\varphi\colon A \to B$. Then it holds that:*

1. *$\mathcal{A} \models \mathcal{B}$ implies $\mathcal{A}_{\downarrow\varphi} \models \mathcal{B}_{\downarrow\varphi}$.*
2. *$\mathcal{C} \models \mathcal{D}$ implies $\mathcal{Q}\varphi.\mathcal{C} \models \mathcal{Q}\varphi.\mathcal{D}$ for $\mathcal{Q} \in \{\exists, \forall\}$.*
3. *$\exists\varphi.(\mathcal{A}_{\downarrow\varphi}) \models \mathcal{A}$ and for every $\mathcal{C}$ with $\exists\varphi.\mathcal{C} \models \mathcal{A}$ we have that $\mathcal{C} \models \mathcal{A}_{\downarrow\varphi}$.*
4. *$\mathcal{A} \models \forall\varphi.(\mathcal{A}_{\downarrow\varphi})$ and for every $\mathcal{C}$ with $\mathcal{A} \models \forall\varphi.\mathcal{C}$ we have that $\mathcal{A}_{\downarrow\varphi} \models \mathcal{C}$.*

The adjunction is strongly reminiscent of categorical logic [14], where logical quantifiers are obtained as left or right adjoints to projections or pullback functor.

One easily obtains the following functoriality and de Morgan laws for shift and quantification:

$$\begin{array}{ccc}
\mathcal{A}_{\downarrow\mathrm{id}} \equiv \mathcal{A} & \mathcal{A}_{\downarrow\varphi\,;\psi} \equiv (\mathcal{A}_{\downarrow\varphi})_{\downarrow\psi} & \neg\mathcal{A}_{\downarrow\varphi} \equiv (\neg\mathcal{A})_{\downarrow\varphi} \\
\forall\mathrm{id}.\mathcal{A} \equiv \mathcal{A} & \forall(\varphi\,;\psi).\mathcal{A} \equiv \forall\varphi.\forall\psi.\mathcal{A} & \neg\forall\varphi.\mathcal{A} \equiv \exists\varphi.(\neg\mathcal{A}) \\
\exists\mathrm{id}.\mathcal{A} \equiv \mathcal{A} & \exists(\varphi\,;\psi).\mathcal{A} \equiv \exists\varphi.\exists\psi.\mathcal{A} & \neg\exists\varphi.\mathcal{A} \equiv \forall\varphi.(\neg\mathcal{A})
\end{array}$$

Since shift has a left *and* a right adjoint, it is a right *and* left adjoint itself. Left adjoints preserve colimits and right adjoints preserve limits. Since conjunction is a product, hence a limit, and disjunction is a coproduct, hence a colimit, we immediately obtain the following laws (which would not be very difficult to prove directly). This is in accordance with predicate logic where universal quantification distributes over conjunction and existential quantification over disjunction.

$$\begin{array}{cc}
(\mathcal{A} \wedge \mathcal{B})_{\downarrow\varphi} \equiv \mathcal{A}_{\downarrow\varphi} \wedge \mathcal{B}_{\downarrow\varphi} & (\mathcal{A} \vee \mathcal{B})_{\downarrow\varphi} \equiv \mathcal{A}_{\downarrow\varphi} \vee \mathcal{B}_{\downarrow\varphi} \\
\forall\varphi.(\mathcal{A} \wedge \mathcal{B}) \equiv \forall\varphi.\mathcal{A} \wedge \forall\varphi.\mathcal{B} & \exists\varphi.\mathcal{A} \vee \exists\varphi.\mathcal{B} \equiv \exists\varphi.\mathcal{A} \vee \exists\varphi.\mathcal{B}
\end{array}$$

Instead if we combine existential quantification and conjunction or universal quantification and disjunction we obtain the following laws involving representative squares.

▶ **Proposition 18** (Quantifier Distribution). *The following quantifier distribution laws hold:*

$$\exists\varphi.\mathcal{A} \wedge \exists\psi.\mathcal{B} \equiv \bigvee_{(\alpha,\beta)\in\kappa(\varphi,\psi)} \exists(\varphi;\alpha).(\mathcal{A}_{\downarrow\alpha} \wedge \mathcal{B}_{\downarrow\beta})$$

$$\forall\varphi.\mathcal{A} \vee \forall\psi.\mathcal{B} \equiv \bigwedge_{(\alpha,\beta)\in\kappa(\varphi,\psi)} \forall(\varphi;\alpha).(\mathcal{A}_{\downarrow\alpha} \vee \mathcal{B}_{\downarrow\beta})$$

6 Applications

After introducing the theory we will now give two applications: Hoare logic and critical pair analysis. Both applications generalize the special case of graph transformation to the setting of reactive systems.

Compared to earlier work on graph transformation our presentation is much simpler (compare with [13] for Hoare logic and [3] for critical pair analysis), which is due to the switch to a higher level of abstraction. Note that, as explained in Section 2, we are also working with a slightly more expressive logic.

Furthermore we improve the result in [15] by exhibiting an if-and-only-if result for critical pair analysis without application conditions and we strengthen the theorem in [3] for critical pair analysis with application conditions.

6.1 Weakest Preconditions and Strongest Postconditions

We will now show how to define Hoare triples, weakest preconditions and strongest postconditions in this framework.

▶ **Definition 19** (Hoare Triple, Weakest Precondition, Strongest Postcondition). Let $R = (\ell, r, \mathcal{C})$ be a rule with $\ell, r\colon 0 \to I$ and application condition $\mathcal{C}$ and let $\mathcal{A}, \mathcal{B}$ be conditions with root object 0. We say that $\mathcal{A}, R, \mathcal{B}$ form a *Hoare triple* (written as $\{\mathcal{A}\}\, R\, \{\mathcal{B}\}$) if for all $a, b\colon 0 \to J$ with $a \models \mathcal{A}$ and $a \Rightarrow_{\{R\}} b$ we have that $b \models \mathcal{B}$.

A condition $\mathcal{A}$ is called a *precondition* for R and $\mathcal{B}$ whenever $\{\mathcal{A}\}\, R\, \{\mathcal{B}\}$. Similarly $\mathcal{B}$ is called a *postcondition* for $\mathcal{A}$ and R.

A condition $\mathcal{A}$ is the *weakest precondition* for R and $\mathcal{B}$ (written $wp(R, \mathcal{B})$), whenever it is a precondition and for every other precondition $\mathcal{A}'$ we have that $\mathcal{A}' \models \mathcal{A}$. A condition $\mathcal{B}$ is the *strongest postcondition* for $\mathcal{A}$ and R (written $sp(\mathcal{A}, R)$), whenever it is a postcondition and for every other postcondition $\mathcal{B}'$ we have that $\mathcal{B} \models \mathcal{B}'$.

With all the machinery in place it is now easy to construct weakest preconditions and strongest postconditions.

▶ **Proposition 20** (Weakest Precondition, Strongest Postcondition). *Let $R = (\ell, r, \mathcal{C})$ be a rule with application condition as in Definition 19 and let $\mathcal{A}, \mathcal{B}$ be conditions with root object 0. Then*

- $wp(R, \mathcal{B}) = \forall\ell.(\mathcal{C} \to \mathcal{B}_{\downarrow r})$
- $sp(\mathcal{A}, R) = \exists r.(\mathcal{C} \wedge \mathcal{A}_{\downarrow\ell})$

Compared to the constructions for graph transformation or transformation systems over adhesive categories in [13] our definitions are much simpler. This is due mainly to two reasons: First, the shift operation relies on the powerful underlying notion of representative squares. Second, our conditions are defined in the same category as the rules, i.e., they would be cospans in the setting of [13]. As already discussed in [9] this simplifies matters and allows us to express dangling and inhibition conditions directly in the logics. When spelled out,

the constructions are more or less identical, but we believe that this more abstract view is very helpful to better understand the theory and to find additional applications, such as the following critical pair lemma.

6.2 Critical Pair Lemma

In order to show the fact that a given reactive system with rule set $\mathcal{R}$ is confluent, we use the well-known result from rewriting theory that states that a terminating rewriting system is confluent if and only if it is locally confluent [21]. Local confluence means that for all arrows a, b_1, b_2 with $a \Rightarrow b_1$, $a \Rightarrow b_2$ there exists an arrow c such that $b_1 \Rightarrow^* c$ and $b_2 \Rightarrow^* c$.

Local confluence can be reduced to showing confluence for so-called critical pairs, i.e., overlapping left-hand sides. Overlaps of left-hand sides can be described by our notion of representative squares. When there are only finitely many representative squares in $\kappa(a, b)$, showing local confluence becomes a much easier task.

▶ **Definition 21** (Critical Pair)**.** Let $R_i = (\ell_i, r_i)$, for $i \in \{1, 2\}$ with $\ell_i, r_i \colon 0 \to I_i$ be two rules. A *critical pair* for R_1, R_2 is a pair (c_1, c_2) of arrows $c_1 \colon I_1 \to K$ and $c_2 \colon I_2 \to K$ such that $(\ell_1, \ell_2, c_1, c_2)$ is a representative square.

▶ **Proposition 22** (Local Confluence for Rules without Application Conditions)**.** *Let $\mathcal{R}$ be a set of rules without application conditions. Then $\Rightarrow_{\mathcal{R}}$ is locally confluent if and only if for every critical pair (c_1, c_2) for rules (ℓ_i, r_i), $i \in \{1, 2\}$, in $\mathcal{R}$ there exists an arrow d with $r_1 ; c_1 \Rightarrow^* d$ and $r_2 ; c_2 \Rightarrow^* d$.*

The above result is not directly comparable to Plump's results in [15, 16]. Plump establishes the "if" direction of a criticial pair lemma, but his notion of confluence is different. Our notion of confluence is connected to what Plump calls "strongly joinable", which means that the common reducts must not only be isomorphic, but also have the same interface. Interestingly, our notion of confluence is decidable for terminating graph transformation systems – since the set of critical pairs is finite and constructible, and for terminating graph transformation systems (strong) joinability of the critical pairs is trivially decidable – whereas Plump's notion of confluence is not.

In order to extend Proposition 22 to rules with application conditions, we first have to collect conditions over a reduction sequence.

▶ **Definition 23** (Conditions for Reductions)**.** Let $\mathcal{R}$ be a set of rules with application conditions. For two arrows $a, b \colon 0 \to J$ we define $\mathcal{A} \rhd a \Rightarrow_{\mathcal{R}} b$ (where $\mathcal{A} \in \mathbf{C}_J$) if for all $d \colon J \to K$ with $d \models \mathcal{A}$ there exists $(\ell, r, \mathcal{B}) \in \mathcal{R}$ and an arrow c such that $a = \ell ; c$, $b = r ; c$ and $c ; d \models \mathcal{B}$.

The intuitive meaning of $\mathcal{A} \rhd a \Rightarrow_{\mathcal{R}} b$ is that a can reduce to b whenever it is put into a passive context satisfying $\mathcal{A}$. It is easy to show that $\mathcal{A} \rhd a \Rightarrow_{\mathcal{R}} b$ and $d \models \mathcal{A}$ imply $a ; d \Rightarrow_{\mathcal{R}} b ; d$.

▶ **Lemma 24.** *Let $\mathcal{R}$ be a set of rules with application conditions and let $a, b \colon 0 \to J$ two arrows. Then the weakest condition $\mathcal{A}$ with $\mathcal{A} \rhd a \Rightarrow_{\mathcal{R}} b$ can be obtained as follows:*

$$\mathcal{A} = \bigvee \{ \mathcal{B}_{\downarrow c} \mid (\ell, r, \mathcal{B}) \in \mathcal{R},\ \ell ; c = a,\ r ; c = b \}$$

▶ **Proposition 25.** *Let $\mathcal{R}$ be a set of rules with application conditions. The following laws hold:*

$$\frac{}{\mathcal{A}_{\downarrow c} \rhd \ell\,;c \Rightarrow^*_{\mathcal{R}} r\,;c} \text{ if } (\ell, r, \mathcal{A}) \in \mathcal{R} \qquad \frac{\mathcal{A} \rhd a \Rightarrow^*_{\mathcal{R}} b \quad \mathcal{B} \rhd a \Rightarrow^*_{\mathcal{R}} b}{\mathcal{A} \vee \mathcal{B} \rhd a \Rightarrow^*_{\mathcal{R}} b}$$

$$\frac{\mathcal{A} \rhd a \Rightarrow^*_{\mathcal{R}} b \quad \mathcal{B} \rhd b \Rightarrow^*_{\mathcal{R}} c}{\mathcal{A} \wedge \mathcal{B} \rhd a \Rightarrow^*_{\mathcal{R}} c} \qquad \frac{\mathcal{B} \rhd a \Rightarrow^*_{\mathcal{R}} b \quad \mathcal{A} \models \mathcal{B}}{\mathcal{A} \rhd a \Rightarrow^*_{\mathcal{R}} b}$$

If, for two arrows a, b, there are only finitely many derivation paths leading from a to b, then the rules above are complete. This is for instance the case if $\Rightarrow_{\mathcal{R}}$ is finitely branching and the system is terminating. Otherwise there is no guarantee that the weakest condition $\mathcal{A}$ satisfying $\mathcal{A} \rhd a \Rightarrow^*_{\mathcal{R}} b$ is even expressible as a finite (first-order) condition.

▶ **Proposition 26** (Local Confluence for Rules with Application Conditions). *Let $\mathcal{R}$ be a set of rules with application conditions. Then $\Rightarrow_{\mathcal{R}}$ is locally confluent if, for every critical pair (c_1, c_2) of rules $(\ell_i, r_i, \mathcal{A}_i)$ $(i \in \{1, 2\})$ in $\mathcal{R}$, there exist arrows $d_1, \ldots, d_m$ and conditions $\mathcal{C}^1_1, \mathcal{C}^1_2, \ldots, \mathcal{C}^m_1, \mathcal{C}^m_2$ such that $\mathcal{C}^i_1 \rhd r_1\,;c_1 \Rightarrow^*_{\mathcal{R}} d_i$ and $\mathcal{C}^i_2 \rhd r_2\,;c_2 \Rightarrow^*_{\mathcal{R}} d_i$ and*

$$(\mathcal{A}_1)_{\downarrow c_1} \wedge (\mathcal{A}_2)_{\downarrow c_2} \models \bigvee_{i=1}^{m} (\mathcal{C}^i_1 \wedge \mathcal{C}^i_2).$$

That is, we have to show that the condition describing that both left-hand sides match (shifted to the common context) implies one of the conditions specifiying that the reduction sequences can again be joined. The ideas underlying this result are taken from [3]. Note however that our result is stronger, since we weaken the precondition: the precondition in [3], transferred to reactive systems, would require that there is a single d such that $\mathcal{C}_1 \rhd r_1\,;c_1 \Rightarrow^*_{\mathcal{R}} d$ and $\mathcal{C}_2 \rhd r_1\,;c_2 \Rightarrow^*_{\mathcal{R}} d$ and $(\mathcal{A}_1)_{\downarrow c_1} \wedge (\mathcal{A}_2)_{\downarrow c_2} \models \mathcal{C}_1 \wedge \mathcal{C}_2$.

In Proposition 26 it is probably hard to obtain "if and only if", due to the non-monotonicity of rules with application conditions. Consider the following example:

▶ **Example 27.** We perform rewriting of labelled sets (basically Petri nets), where the start set contains a single element labelled A. A can either be replaced by B or C (that is, we have a critical pair). Now both B and C can be rewritten to D, but only if no E is present. Hence the system as such is not (locally) confluent. If however we add a rule removing E's the system would become confluent, since we could remove all E's first. However, this could take an arbitrary number of steps since there could be arbitrarily many E's around.

The problem is also, in a sense, that by writing $\mathcal{A} \rhd a \Rightarrow b$ we talk about a *passive* context (satisfying $\mathcal{A}$), on which the conditions are evaluated, but which does not truly interact with a.

7 Conclusion

We have shown how reactive systems can be extended with conditions, generalizing well-known constructions and results (axioms, pre- and postconditions, critical pair lemma) to the very general setting of reactive systems.

With the computation of weakest preconditions and strongest postconditions we now have the means to do Hoare logic reasoning (similar to [17]) for graph transformation and, even more interesting, to set up a framework for counterexample-guided abstraction refinement (CEGAR) in the sense of [8].

Another question of future research is to determine whether the axioms presented in Section 5 can be extended to a complete set of axioms. The thesis by Pennemann [13] contains some interesting developments going in this direction, including tool support. However, note

that this question will for sure also depend on the category: it is known that for finite graphs the satisfiability problem is semi-decidable, while the validity problem is not, whereas it is exactly the other way around for arbitrary (finite and infinite) graphs.

Finally we want to extend the derivation of labels and generation of bisimulation congruences to the case of conditional reactive systems.

References

1 H.J.S. Bruggink and B. König. A logic on subobjects and recognizability. In *Proc. of IFIP-TCS '10*, volume 323 of *IFIP AICT*, pages 197–212. Springer, 2010.

2 H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. Springer, 2006.

3 H. Ehrig, A. Habel, L. Lambers, F. Orejas, and U. Golas. Local confluence for rules with nested application conditions. In *Proc. of ICGT '10*, pages 330–345. Springer, 2010. LNCS 6372.

4 H. Ehrig and B. König. Deriving bisimulation congruences in the DPO approach to graph rewriting with borrowed contexts. *MSCS*, 16(6):1133–1163, 2006.

5 A. Habel and K.-H. Pennemann. Satisfiability of high-level conditions. In *Proc. of ICGT '06*, pages 430–444. Springer, 2006. LNCS 4178.

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

7 R. Heckel and A. Wagner. Ensuring consistency of conditional graph rewriting - a constructive approach. In *Proc. of the Joint COMPUGRAPH/SEMAGRAPH Workshop on Graph Rewriting and Computation*, volume 2 of *ENTCS*, 1995.

8 T.A. Henzinger, R. Jhala, R. Majumdar, and K.L. McMillan. Abstractions from proofs. In *Proc. of POPL '04*, pages 232–244. ACM, 2004.

9 M. Hülsbusch. Application conditions for reactive systems with applications to bisimulation theory. In *ICGT 2010 – Doctoral Symposium*, ECEASST 38, 2011.

10 S. Lack and P. Sobociński. Adhesive and quasiadhesive categories. *RAIRO – Theoretical Informatics and Applications*, 39(3), 2005.

11 J.J. Leifer. *Operational congruences for reactive systems*. PhD thesis, University of Cambridge Computer Laboratory, September 2001.

12 J.J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In *Proc. of CONCUR 2000*, pages 243–258. Springer, 2000. LNCS 1877.

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

14 A.M. Pitts. Categorical logic. In *Handbook of Logic in Computer Science V*. Oxford University Press, 2001.

15 D. Plump. Hypergraph rewriting: Critical pairs and undecidability of confluence. In M.R. Sleep, M.J. Plasmeijer, and M.C. van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, chapter 15, pages 201–214. John Wiley, 1993.

16 D. Plump. Confluence of graph transformation revisited. In A. Middeldorp, V. van Oostrom, F. van Raamsdonk, and R. de Vrijer, editors, *Festschrift Jan Willem Klop*. Springer, 2005. LNCS 3838.

17 C.M. Poskitt and D. Plump. A Hoare calculus for graph programs. In *Proc. of ICGT '10*, pages 139–154. Springer, 2010. LNCS 6372.

18 A. Rensink. Representing first-order logic using graphs. In *Proc. of ICGT '04*, pages 319–335. Springer, 2004. LNCS 3256.

19 V. Sassone and P. Sobociński. Reactive systems over cospans. In *Proc. of LICS '05*, pages 311–320. IEEE, 2005.

20 P. Sobociński. *Deriving process congruences from reaction rules.* PhD thesis, Department of Computer Science, University of Aarhus, 2004.
21 Terese. *Term Rewriting Systems.* CTTCS 55. Cambridge University Press, 2003.