Proceedings of the Fifth International Conference on
Graph Transformation - Doctoral Symposium
(ICGT-DS 2010)

Application Conditions for Reactive Systems
with Applications to Bisimulation Theory

Mathias Hülbusch

14 pages
Application Conditions for Reactive Systems
with Applications to Bisimulation Theory

Mathias Hülsbusch

1 Abteilung für Informatik und Angewandte Kognitionswissenschaft,
Universität Duisburg-Essen, Germany
mathias.huelsbusch@uni-duesseldorf.de

Abstract: This paper presents generalized application conditions (GACs), a new formalism for nested application conditions. GACs are not only suitable for DPO rewriting, but for rewriting in reactive systems as well. The main theorem states that it is possible to construct an equivalent reactive system rule with a GAC for a DPO rule with application conditions under very mild conditions. The resulting reactive system rules live in the cospan category of the category C, in which the DPO rules live.

It turns out that these GACs for reactive systems provide a slightly more powerful way to control the application of a rewriting rule, than it is possible in the original DPO setting.

At the end, we give a short outlook on the applications of this formalism to the field of bisimulation theory, sketch our latest results and discuss future work.

Keywords: FOL on Graphs, Application Conditions, Reactive Systems, Cospan Category, DPO

1 Introduction

The analysis of dynamic systems is an important and useful task in many domains such as software analysis, MDD (model driven development) and software security. A successful approach to model such systems is to model the system as a graph and to implement system evolution via graph rewriting rules. We consider the DPO approach to rewriting in an adhesive category C [Roz97, LS05]. This modeling technique allows to represent concurrent behavior, changing topologies, etc. and features all the benefits of a graphical modeling language. According to the actual developments, the complexity of systems that software and hardware engineers will face in the future will grow rapidly. For this reason, graphical modeling languages are very likely to play a key role in development approaches.

When modeling real-world processes, one wants to be able to limit the applicability of rules by application conditions. Even if the basic setting is Turing complete (such as graph transformation systems), the process of modeling can be extremely simplified by using application conditions.

To motivate this, here is a short (running-)example, that shows how useful application conditions are: In a file system, a user U wants to delete an object O, that the user owns. This deletion is only allowed if the object is not marked as protected. This is a negative application condition.
But even if the object is protected, a user with root permissions is still allowed to delete it. This second part disables the negative application condition.

The mechanisms of application conditions also gives rise to the idea of pre- and postconditions for rules allowing the use of verification techniques inspired by Hoare logic. So far, (nested) application conditions have been studied mainly for DPO rewriting. Another approach to analyze dynamic systems is the theory of reactive systems. DPO rewriting in adhesive categories and reactive systems rewriting are related very closely. Nevertheless, there is no theory of application conditions for reactive systems. In this paper, we will present a new way to describe application conditions that is suitable for DPO rewriting and for reactive systems rewriting, called generalized application conditions (GACs). Instead of describing conditions on the match, in reactive systems rewriting a GAC gives the conditions the context, the reaction takes place in, has to fulfill. This new approach to application conditions allows to adapt the borrowed context technique to rules with application conditions, giving transition labels that are describing the conditions a borrowed context has to fulfill in order to make the rule applicable. We will give some more detailed information about these ideas in Section 5.

The second Section gives some preliminary definitions. In Section 3, we will present the definition of generalized application conditions (GACs). Section 4 states the main theorem of this paper, describing the relation between GACs for DPO rewriting and GACs for reactive systems. In order to illustrate the presented techniques, we will introduce a running example.

2 Preliminaries

In this section, we will give some basic definitions that are needed later on. First, we will define the rewriting approaches we consider in this paper:

**Definition 1 (DPO Rewriting)** Let \( C \) be an adhesive category. An object \( G \) can be rewritten to \( H \) via a DPO rule \( p = L \leftarrow I \rightarrow R \), whenever the exists the following diagram, where both squares are pushouts. Note that we work in a setting where the vertical morphisms must be monos.

\[
\begin{array}{c}
L \downarrow \quad \quad I \quad \quad \downarrow R \\
\downarrow \quad \quad \quad \downarrow \quad \quad \quad \downarrow \\
G \quad \quad \quad \quad D \quad \quad \quad \quad \quad \quad H
\end{array}
\]

**Definition 2 (Rewriting in reactive systems)** A reactive system \( S \) over a category \( C \) is a set \( R_S \) of rewriting rules. Each rule \( s \) is a pair of arrows \( (l, r) \) from \( C l, r : J \rightarrow I \). An arrow \( a \) is rewritten to an arrow \( b \) \( (a, b : J \rightarrow K) \) via a rule \( s \), iff there exists an arrow \( c \), such that \( a = l; c \) and \( b = r; c \) (see diagram below).

\[
\begin{array}{c}
J \xrightarrow{sl} I \leftarrow J \\
\downarrow a \quad \quad \quad \quad \downarrow c \quad \quad \quad \quad \quad \downarrow b \\
K \quad \quad \quad \quad J
\end{array}
\]
In this paper, we will consider only categories that have an initial object 0 and rules, where \( J \) is the initial object of \( C \).

**Definition 3 (Cospan category)** Let \( C \) be a category with pushouts. A cospan \( c : A \rightarrow B \leftarrow C \) is a pair of \( C \)-arrows with the same codomain. Here, \( A \) is the domain of \( c \) and \( C \) is the codomain. The identity cospan for an object \( E \) is the cospan consisting of twice the identity arrow of \( E \). The composition \( z = x; y \) of two cospans \( x = A \rightarrow B \leftarrow C \) and \( y = C \rightarrow D \leftarrow E \) is done via a pushout construction, as shown in the diagram below (where \( f = d; b' \) and \( e = a; c' \)):

![Cospan diagram](https://example.com/cospan_diagram.png)

A *semi-abstract cospan* is an equivalence class of cospans, where we take the middle object of the cospan up to isomorphism. Now the cospan category \( \text{cospan}(C) \) is defined as follows: The objects of \( \text{cospan}(C) \) are the objects of \( C \), the arrows of \( \text{cospan}(C) \) are the semi-abstract cospans.

Reactive systems rewriting over the category \( \text{cospan}(C) \) coincides with DPO rewriting in \( C \) [SS05]. Since we work in a setting, where the vertical morphisms in the DPO diagram must be monos, we consider the category \( \text{cospan}_l(C) \), where all the cospans must be left-linear (the left leg must be a mono) in this paper. Since pushouts preserve monos in adhesive categories and monos are closed under composition, \( \text{cospan}_l(C) \) is a category. The idea how rules can be translated is illustrated in the following image. The thick black arrows depict the reactive system diagram, the thin gray arrows show the double pushout diagram and the dotted arrows are some additional morphisms from \( C \). The objects that are rewritten in \( C \) are the inner objects of the cospans \( a \) and \( b \) (\( G \) is rewritten to \( H \) via the rule \( (l', r') \)).

![Reactive system diagram](https://example.com/reactive_system_diagram.png)
3 Generalized Application Conditions

The formalism of graph predicates and graph conditions [Ren04] is equivalent to first order logic on graphs and based on morphisms. We exploited this approach to give a definition for application conditions that is suitable for DPO rewriting and for reactive systems as well. It turned out that in some cases, this new definition is slightly more powerful then the standard and well studied approach to nested application conditions [Pen08]. The following definition for application conditions works in both cases. It can be used to describe application conditions for DPO rewriting as well as application conditions for reactive system. It is not restricted to any category.

**Definition 4** (Generalized Application Condition (GAC)) Let $A$ be an object of a category $C$. GAC$_A$ is the set of all triples $(A, Q, S)$ where $Q \in \{\forall, \exists\}$, and $S$ is a set of pairs $(\phi_i, \alpha_i)$, where $\phi_i : A \rightarrow A_i$ and $\alpha_i \in \text{GAC}_A$. Let $c : A \rightarrow B$ be an arrow of $C$ and $\alpha$ an element of GAC$_A$. $c \models \alpha$ if and only if:

- $c \models (A, \forall, S)$ if for all $(\phi_i, \alpha_i) \in S$ and for all $d : A_i \rightarrow B$ with $\phi_i \cdot d = c$ it holds that $d \models \alpha_i$.
- $c \models (A, \exists, S)$ if there exists a pair $(\phi_i, \alpha_i) \in S$ and a morphism $d : A_i \rightarrow B$ with $\phi_i \cdot d = c$ and $d \models \alpha_i$ holds.

The object $A$ is called root object of $\alpha$ (RO($\alpha$)).

This definition shows that GACs can be seen as trees, where the nodes are labeled with pairs $(O, Q)$, where $O$ is an object of $C$ and $Q$ is in $\{\forall, \exists\}$ and the edges are labeled with morphisms of $C$. The domain and codomain of every morphism are the objects at the corresponding nodes.

For a (general) DPO rule $L \leftarrow I \rightarrow R$, a GAC in GAC$_L$ restricts the matching morphism $m : L \rightarrow G$. For a reactive system rule $(l : 0 \rightarrow I, r : 0 \rightarrow I)$, a GAC in GAC$_I$ restricts the possible contexts $c$, the reaction takes place in.

In our DPO rewriting, we want certain morphisms to be monos. Therefore, the following definition of MonoGACs is useful.

**Definition 5** (MonoGAC) MonoGAC$_A$ is the set of all triples $(A, Q, S)$ from GAC$_A$, where for every pair $(\phi_i, \alpha_i)$ in $S$ holds: $\phi_i$ is a mono and $\alpha_i \in \text{MonoGAC}_A$. A MonoGAC is evaluated like a normal GAC but all the morphisms $d$ from the objects in the MonoGAC into the codomain of the morphism, the GAC is evaluated on, must be monos.

Recall the example from the introduction: In a file system, there is a user $U$ that owns an object $O$ and wants to delete this object. The deletion of objects is only allowed whenever the object is not protected, or (if the object is protected) the user has administrator privileges (is root). This example can be modeled in the category Graph. The rule del for deleting an object is defined as follows:

\[
\text{del: } O \xrightarrow{\text{owns}} U \quad \xleftarrow{i} \quad U \xrightarrow{r} U
\]

1 Nested application conditions allow nesting of quantifiers.
The GAC $\mathcal{A}$, describing the required conditions looks as follows (The mappings of $\varphi_1$ and $\varphi_2$ are induced by the node- and edge-labels):

To understand how this GAC models the wanted condition, it is helpful to spell out the condition in the following way:

1. In all cases that the object that should be deleted is marked as protected (if the object is not protected, the rule is applicable).
2. Ensure that there exists a marking (root loop) that the user has administrator rights.
3. If the user is root, the rule is applicable in all cases.

While constructing application conditions, one usually faces the problem of composing two GACs by a boolean operation or of negating a GAC. The following definition will give the needed constructions for the standard set of boolean operations.

**Definition 6 (Operations on GACs)** Let $\mathcal{A} = (A, Q, S)$ and $\mathcal{B}$ be from $\text{GAC}_A$ and $c : A \to K$.

**Negation:** $\neg \mathcal{A}$

\[
\neg \mathcal{A} = \begin{cases} 
(A, \exists, S') & \text{if } Q = \forall \\
(A, \forall, S') & \text{if } Q = \exists 
\end{cases}
\]

where $S' = \{ (\varphi, \neg \mathcal{A}) | (\varphi, \mathcal{A}) \in S \}$

**Conjunction:** $\mathcal{A} \land \mathcal{B}$

\[
\mathcal{A} \land \mathcal{B} = (A, \forall, \{(id_A, \mathcal{A}), (id_A, \mathcal{B})\})
\]

**Disjunction:** $\mathcal{A} \lor \mathcal{B}$

\[
\mathcal{A} \lor \mathcal{B} = (A, \exists, \{(id_A, \mathcal{A}), (id_A, \mathcal{B})\})
\]

Note: since every identity is a mono, MonoGACs are closed under negation (where no morphism is changed or added), conjunction and disjunction.

### 4 GACs for Reactive Systems

The definition of GACs is not restricted to special categories. This allows the usage of GACs in other settings, too. In this section, we will show that GACs provide a way to describe application
condition for reactive systems as well. The main theorem states, that every application condition for a DPO rule for an adhesive category $C$ can be translated into a GAC for the corresponding reactive system rule in cospan$_l(C)$.

**Theorem 1** Every DPO rule $p = L \leftarrow I \rightarrow R$ with MonoGAC $\mathcal{A} = (L, Q, S)$ in an adhesive category $C$ can be converted into an equivalent reactive system rule $\tilde{p}$ with GAC $\mathcal{A}$ in cospan$_l(C)$.

The rules $p$ and $\tilde{p}$ and the corresponding GACs are equivalent if for every rewriting step $G \rightarrow_p G'$, the step $(0 \rightarrow G \leftarrow 0) \rightarrow_p (0 \rightarrow G' \leftarrow 0)$ is possible and vice versa.

The proof of this theorem has two main parts. The first part will show, that for a DPO rule $p$, a MonoGAC $\mathcal{A} = (L, Q, S)$ can be transformed into an equivalent MonoGAC $\hat{\mathcal{A}} = (I, Q, \hat{S})$, where $L$ is the LHS of $p$ and $I$ is the interface of $p$. This part basically recalls results from [Pen08] for the definition of MonoGACs. It is straightforward to connect application conditions to rules via the interface $I$, and not via the LHS $L$, since there is everything known about $L$. The parts of a graph that are of interest, concerning the application condition, are not in $L$ and can be reached via the interface $I$ as well. The main construction of part I is given by the following definition:

**Definition 7** (MonoGAC shifting) Given a MonoGAC $\mathcal{A} = (L, Q, S)$ from MonoGAC$_L$ and a morphism $\ell : I \rightarrow L$, the shifting $\mathcal{A} \ell$ of $\mathcal{A}$ via $\ell$ is defined as follows:

\[
\mathcal{A} \ell = (I, Q, G'), \quad \text{where} \quad G' = \left\{ (\varphi', \mathcal{A}_\ell') \mid I \xrightarrow{\varphi'} D \xrightarrow{\ell} \text{RO}(\mathcal{B}) \text{ is the pushout complement of } I \xrightarrow{\varphi} L \xrightarrow{\varphi} \text{RO}(\mathcal{B}) \text{ and } (\varphi, \mathcal{B}) \in G \right\}
\]

For every $(\varphi, \mathcal{B}) \in G$, one constructs all pushout complements for $I \xrightarrow{\varphi} L \xrightarrow{\varphi} \text{RO}(\mathcal{B})$ and takes only the ones into account, that start with a mono. For a GAC $\mathcal{B} = (B, Q, S)$, RO$(\mathcal{B})$ is $B$, the root object of $\mathcal{B}$. Note that both GACs $\mathcal{A}$ and $\mathcal{A} \ell$ are defined over the same category.

The second part will then show how $\mathcal{A} \ell$ is translated into $\mathcal{A}$ over cospan$_l(C)$. Note that the second part works for all GACs. Only the first part needs MonoGACs. The corresponding construction is defined by:

**Definition 8** (GAC conversion) A MonoGAC $\mathcal{A} = (I, Q, S)$ over an adhesive category $C$ is converted into a GAC $\mathcal{A}^{co}$ over the category cospan$_l(C)$ by the following construction:

\[
\mathcal{A}^{co} = (I, Q, S'), \quad \text{where} \quad S' = \left\{ (X \xrightarrow{\varphi} Y \xleftarrow{id} Y, \mathcal{A}_i^{co}) \mid (\varphi, \mathcal{A}_i) \in S \right\}
\]
Part I

Proof.
Let the following pushout diagram be given \((h \text{ and } m \text{ are monos})\) and let \(A\) be from \(\text{MonoGAC}_{L}L_{I}K_{L}G\):

\[
\begin{array}{c}
I \\ \\
\ell \\ \\
L \rightarrow m \rightarrow G
\end{array}
\]

We will now show (by induction over the structure of \(A\)) that the following holds:

\[(m \models A) \iff (h \models A)\]  

Base of induction Assume that the GAC has the form \((L', Q, \emptyset)\).

\[Q = \forall:\] Given a GAC \(\mathcal{A}' = (L', \forall, \emptyset)\) and a morphism \(\ell' : I' \rightarrow L'\), then, \(\mathcal{A}' \ell'\) is \((I', \forall, \emptyset)\). For every morphism \(m' m' \models \mathcal{A}\) holds. And for every \(h' h' \models \mathcal{A}' \ell'\). Therefore, \(\mathcal{A}'\) and \(\mathcal{A}' \ell'\) are equivalent.

\[Q = \exists:\] Given a GAC \(\mathcal{A}' = (L', \exists, \emptyset)\) and a morphism \(\ell' : I' \rightarrow L'\), then, \(\mathcal{A}' \ell'\) is \((I', \exists, \emptyset)\).

There is no morphism \(m'\) such that \(m' \models \mathcal{A}\) holds, and there is no \(h'\), such that \(h' \models \mathcal{A}' \ell'\) holds. Therefore, \(\mathcal{A}'\) and \(\mathcal{A}' \ell'\) are equivalent.

Induction step Assume the following situation: \(\mathcal{A} = (L', Q, \{(\varphi_i, \mathcal{A}_i) | 1 \leq i \leq n\})\), \(\ell' : I' \rightarrow L'\) and \((\ell', m', h', g)\) is a pushout (see diagram below). Let the index set \(C\) be defined as follows:

\[C = \left\{ i | A_i \xrightarrow{\varphi_i} L' \xrightarrow{\ell'} I' \text{ has a pushout complement} \right\}\]

For each \(i \in C\), the situation is depicted in the figure below (\(I' \rightarrow B_i \rightarrow A_i\) is a corresponding pushout complement):

\[
\begin{array}{c}
I' \\ \\
\ell' \\ \\
L' \rightarrow h'' \rightarrow G
\end{array}
\]

Note: The pushout splitting property in adhesive categories [LS05] implies: Whenever there exists a \(m' : A_i \rightarrow G\), making \((\varphi_i, m'', m')\) a commuting triangle, the outer pushout can be split at \(A_i\) (by taking the pullback of \(m''\) and \(g\)) and therefore the pushout complement \(I' \rightarrow B_i \rightarrow A_i\) must exist. Since pullbacks preserve monos, \(h''\) is a mono.

To avoid index overloading, we set \(\mathcal{A}_{L} = B\)
Application Conditions for Reactive Systems

1: \((m' = A) \Rightarrow (h' = B)\)

\(Q = \exists: \) Following the definition of GACs, \((m' = A)\) implies the existence of a mono \(m''\) and a natural number \(i\), such that \(m' = \varphi_i; m''\) and \(m'' = A_i\). The existence of \(m''\) implies \(i \in C\) (the pushout complement must exist, see note above). Let \(h'' : B_i \rightarrow K\) be the morphism that must exist due to the pushout splitting. Sice \(h'' = B_i\) holds by induction, \(h' = B\) holds as well.

\(Q = \forall: \) Assume there exists a mono \(h''\), that does not satisfy \(B_i\). Splitting the outer pushout in the diagram above at \(B_i\) implies the existence of a morphism \(m''\) and by induction we get: \(m'' \neq A_i\). This contradicts \((m' = A)\), since there is a morphism \(m''\) from \(A_i\) to \(G\), and \(m' = \varphi_i; m''\) holds, but \(m'' = A_i\) does not hold. But by definition for every morphism \(m''\), such that \(m' = \varphi_i; m''\) holds, \(m'' = A_i\) holds.

2: \((m' = A) \Leftarrow (h' = B)\)

\(Q = \exists: \) \((h' = B)\) implies the existence of a mono \(h'' : B_i \rightarrow K\). The morphism \(m''\) is now obtained as the mediating morphism from the left pushout square in the diagram above, considering the morphisms \(m'\) and \(h''; g\). By induction, we know that \(h'' = B_i \leftrightarrow m'' = A_i\). Therefore \(m' = A_i\)

\(Q = \forall: \) Assume there exists a mono \(m'' : A_i \rightarrow G\) and \(m'' \neq A_i\). By the construction given above, there must be a \(h''\), not satisfying \(B_i\). This contradicts that every possible morphism \(h'' : B_i \rightarrow K\) satisfies \(B_i\), as the induction hypothesis states.

Consider the running example:

It is easy to see, that there is no pushout complement for \(\bullet \rightarrow \bullet \rightarrow \bullet\) (dangling edges), and therefore \(A_\ell\) is the trivial GAC \(\left( U, \forall, \emptyset \right)\), which is satisfied by every morphism \(m\) that has a single user node as domain.

This does not contradict Theorem 1, since rule del is never applicable to protected objects due to the dangling edge condition for the protected loop. In order to make the example more interesting, we introduce a new rule delp, deleting protected objects. The application condition now ensures that the user is root.

\[\text{del}_p: \quad \text{protected} \]

\[\begin{array}{c}
O \owns U \quad \xrightarrow{\ell} \quad U \quad \xrightarrow{i} \quad U
\end{array}\]

\[\text{This case can be modeled by a DPO rule without application condition, by simply adding the root loop to the user node in the LHS, the interface and the RHS, but due to space limitations, we have chosen this small example.}\]
This GAC can be spelled out as: Whenever one wants to delete a protected object, there must exist a root loop at the user node.

In this situation, exactly one pushout complement of $\bullet \xrightarrow{\ell} \bullet \xrightarrow{\phi} \bullet$ exists. It is:

This pushout complement results in:

$\mathcal{B}_{\psi}$: 

$\left( \begin{array}{c}
\text{root} \\
\mathcal{U} \xrightarrow{\psi'} \mathcal{U} \\
\text{protected} \\
\mathcal{U} \xrightarrow{\ell} \mathcal{U} \\
\end{array} \right)$ 

$\left( \begin{array}{c}
\text{root} \\
\mathcal{U} \xrightarrow{\psi} \mathcal{U} \\
\text{protected} \\
\mathcal{U} \xrightarrow{\phi} \mathcal{U} \\
\end{array} \right)$

Part II

Proof.

Let $m = I \to H$ be a mono from $\mathcal{C}$, $\mathcal{A} \in \text{GAC}_I$ and let $c := I \xrightarrow{m} H \leftarrow 0$ be a morphism from $\text{cospant}_I(\mathcal{C})$. It is left to show that the following holds:

$(m \models \mathcal{A}) \iff (c \models \mathcal{A}^{co})$

The proof will use induction over the structure of $\mathcal{A}$.

Base of induction Let $\mathcal{A}$ be $(I', Q, \emptyset)$. By construction $\mathcal{A}^{co}$ is $(I', Q, \emptyset)$.

$Q = \exists$: No morphism from $\mathcal{C}$ satisfies $(I', \exists, \emptyset)$ and no cospan from $\text{cospant}(\mathcal{C})$ satisfies $(I', \exists, \emptyset)$. Therefore the two GACs are equivalent.

$\mathcal{A}^{co}$: The proof works even for classical DPO rewriting (the vertical morphisms are not necessarily monos), but in this setting the constructed cospans are no longer left-linear.
**Application Conditions for Reactive Systems**

\( Q = \forall: \) Every morphism from \( C \) satisfies \((I', \forall, \emptyset)\) and every cospan from cospan \((C)\) satisfies \((I', \forall, \emptyset)\). Therefore the two GACs are equivalent.

**Induction step** Assume \( \mathcal{A} = (I', Q, S) \) with \( S = \bigcup_{i=1}^{n} (\mathcal{A}_i, \mathcal{A}_i) \) and for all \( \mathcal{A}_i \) holds:

\[
(m' \models \mathcal{A}_i) \iff (c' \models \mathcal{A}_i^{\text{co}}), \text{ whenever } c' \text{ is } I' \to H \leftarrow 0.
\]

\( Q = \exists: \) From the definition of GACs it follows that there exists a morphism \( m' \) and an index \( i \) such that \( m = \varphi_i; m' \) and \( m' \models \mathcal{A}_i \). By induction, we know that \( c' = A_i \xymatrix{ m' \ar[r] & H \ar[l]_0 } \) satisfies \( \mathcal{A}_i^{\text{co}} \). The following diagram shows (since the inner square is a pushout) that for the cospan \( \varphi_i' = I' \xymatrix{ A_i \ar[r] & H } \) the following holds:

\[
\varphi_i'; c' = c. \text{ Since } c' \models \mathcal{A}_i^{\text{co}}, c \text{ satisfies } \mathcal{A}^{\text{co}}.
\]

\( Q = \forall: \) Assume there exists a cospan \( c \) such that \( c \not\models \mathcal{A}^{\text{co}} \). By definition we get the existence of an index \( i \) and a morphism \( c' \) such that \( \varphi_i; c' = c \) and \( c' \not\models \mathcal{A}_i^{\text{co}} \). The diagram above shows that there exists a morphism \( m' : A_i \to H \). Since \( m \models \mathcal{A}, m' \) must satisfy \( \mathcal{A}_i \). Using the induction hypothesis, this means that \( c' \models \mathcal{A}_i^{\text{co}} \), which contradicts the assumption.

\( Q = \exists: \) From the definition of GACs, there must be a cospan \( c' = A_i \xymatrix{ m' \ar[r] & H \ar[l]_0 } \) and an index \( i \) such that \( c = \varphi_i; c' \) and \( c' \not\models \mathcal{A}_i^{\text{co}} \). By the induction hypothesis, \( m' \models \mathcal{A}_i \). Therefore, \( m \) satisfies \( \mathcal{A} \).

\( Q = \forall: \) Assume there exists a morphism \( m' \) and an index \( i \), such that \( m = \varphi_i; m' \) and \( m' \not\models \mathcal{A}_i \). By the induction hypothesis, the cospan \( \varphi_i' = I' \xymatrix{ A_i \ar[r] & H } \) cannot satisfy \( \mathcal{A}_i^{\text{co}} \). Since \( \varphi_i'; c' = c \), \( c \) cannot satisfy \( \mathcal{A}^{\text{co}} \), which contradicts the assumption.

\[ \square \]

Using these results, the following construction gives an equivalent reactive system rule \( \tilde{p} \) with GAC \( \tilde{\mathcal{A}} \) over cospan \((C)\) for a DPO rule \( p = L \xymatrix{ I \ar[r]^r & R } \) with a MonoGAC \( \mathcal{A} \in \text{MonoGAC}_L \).
over \( \mathbb{C} \).
\[
\bar{p} = \left( 0 \rightarrow L \overset{\ell}{\leftarrow} 1, 0 \rightarrow R \overset{r}{\leftarrow} 1 \right)
\]
\( \mathcal{A} = (\mathcal{A}_\ell)^{co} \)

We have shown that \( \mathcal{A} \) is equivalent to \( \mathcal{A}_\ell \) and that \( \mathcal{A}_\ell \) is equivalent to \( (\mathcal{A}_\ell)^{co} \). Therefore, \( \mathcal{A} \) and \( \mathcal{A}_\ell \) are equivalent.

The construction for the running example:
First, we build the rule \( \tilde{\text{del}}_p = (s_\ell, s_r) \) by constructing \( s_\ell \) and \( s_r \).

\[
s_\ell = 0 \rightarrow \begin{array}{c}
O \quad \text{owns} \\
0 \rightarrow U \\
\end{array} \quad \ell \quad \begin{array}{c}
U \\
\end{array}
\]

\[
s_r = 0 \rightarrow \begin{array}{c}
U \\
\end{array} \quad r \quad \begin{array}{c}
U \\
\end{array}
\]

The GAC \( \mathcal{B} \) is finally constructed as:

\[
\mathcal{B} : \left( \begin{array}{c}
\left( U, \exists, \emptyset \right), \\
\emptyset = \left( \begin{array}{c}
\left( U, \phi \right), \\
\left( U, \uparrow \right), \\
\left( U, \downarrow \right) \\
\left( U, \forall, \emptyset \right) \\
\end{array} \right) \\
\end{array} \right)
\]

GACs in \( \text{cospan}_{\Sigma}(\mathbb{C}) \) allow slightly more control over the application of a rule, than the corresponding GACs in \( \mathbb{C} \). As induced by the results above, these cases must use GACs, where at least one cospan is not of the form \( A \rightarrow B \overset{id}{\leftarrow} B \).

**Definition 9** (finite GAC) A GAC \( \mathcal{A} = (I, Q, S) \) is called finite, whenever the object \( I \) is finite, all GACs in \( S \) are finite and \( S \) is finite.

An object \( I \) is finite iff the numbers of monos (up to isomorphism) with codomain \( I \) is finite.

**Lemma 1** There exists a finite GAC \( \mathcal{A} \) in \( \text{cospan}_{\Sigma}(\mathbb{C}) \), such that there is no finite GAC \( \mathcal{B} \) in \( \mathbb{C} \) with \( \mathcal{B}^{oo} \equiv \mathcal{A} \).

**Proof.** Let \( \mathbb{C} \) be the category of graphs with an infinite alphabet \( \Sigma \) of edge labels, where morphisms must preserve labels. Let \( \mathcal{A} \) be defined as \( (0, \forall, \{(0 \rightarrow o \leftarrow 0, (0, \exists, \emptyset))\}) \). This representation shows: \( \mathcal{A} \) is finite. If a cospan \( c \) satisfies \( \mathcal{A}, D \) (the inner object of the cospan \( c \)) contains no isolated node. Assume a finite GAC \( \mathcal{A} \) exists, such that \( \mathcal{A} \equiv \mathcal{B}^{oo} \). Let \( L \) be the set of all edge labels of all the
graphs, contained somewhere in $\mathcal{B}$. Since $\mathcal{B}$ is finite, $L$ is finite. Therefore, there is a label $e$ in $\Sigma$, that is not in $L$. Since there is no edge label $e$ in $\mathcal{B}$, whenever $m = 0 \rightarrow (\circ \xrightarrow{e} \circ)$ satisfies $\mathcal{B}$, so does $m_e = 0 \rightarrow (\circ \xrightarrow{e} \circ)$. This holds, because the edge cannot be in the image of any morphism originating from objects in $\mathcal{B}$, since labels must be preserved. Let $c$ be $0 \rightarrow (\circ \xrightarrow{e} \circ) \leftarrow 0$. This gives: $c \not\models \mathcal{A}$, but $m \models \mathcal{B}$, which implies: $\mathcal{A} \not\equiv \mathcal{B}^{co}$. This contradicts the assumption.

5 Application to Bisimulation Theory and Future Work

We will now describe the applications of GACs to bisimulation theory. We are considering open systems, that can interact with a context.

**Context aware bisimulations:**

The behavior of two systems might be different, if both systems are in a certain context, but may be bisimilar under another set of contexts. Our key idea is to use GACs to describe the conditions a context has to satisfy such that the behavior of the systems is bisimilar under that context. While exploiting this idea, there are several problems to solve:

1. Is the context of a system fixed forever, or may the context change over time? This includes the problem that sometimes the application of a rule is possible and sometimes it is not.

2. Is the entire context visible to the system, or does a system only see the part of the context that actually matters for the next step?

3. How can one model the different influences on the system, caused by the behavior of the system itself or introduced by the context, the system is in? For example, some branching decisions are made by the actual context, other branches offer the freedom of choice to the system.

To analyze the behavior of a graph rewriting system (in an adhesive graph category $\mathcal{C}$), one could generate a labeled transition system (LTS), where the states of the LTS are graphs, and a transition between two graphs $G_A$ and $G_B$ indicates the fact, that $G_A$ can be rewritten to $G_B$ by a rewriting rule. In order to allow a more specific analysis, the transitions can be labeled with additional information (such as the name of the rule etc.) One very successful mechanism to create labels is the borrowed context technique [EK06]. The borrowed context label shows what has to be added to $G_A$, to make a rule applicable.

Transferring this idea to the setting of reactive systems, the borrowed context diagrams turn out to be a special class of squares in $\text{cospan}_\mathcal{C}(\mathcal{C})$, sharing some nice properties (they are GIPOS in the corresponding 2-category [SS03]). This allows to adapt the borrowed context technique to reactive system rules with application conditions.

We already have adapted (but not yet published) label generation via borrowed contexts to rules with GACs, deriving a GAC the added context has to satisfy.

To analyze the behavior of the system, one can apply methods from bisimulation theory to the generated LTSs. For negative application conditions on $DPO$ rules, this has been done in [RKE08]. We are currently working on the proof that the bisimulations obtained by our label
generation process are congruences and are the coarsest bisimulations.

To find a more general approach to the problem of context-sensitive behavior analysis, we are currently working on a setting that covers all of the cases above. Therefore, we have expanded the following approach: An LTS can be seen as a coalgebra. If the final coalgebra \( \Omega \) exists, bisimilar states are mapped to the same state in \( \Omega \). Our idea is to hide the side-effects the context gives in a monad and then apply this techniques to the resulting Kleisli category. We have sketched an algorithm that computes not the complete final coalgebra, but only the part that is needed to compare the behavior of the given states. The information what behavior a state has under which context is given by the Kleisli arrow into \( \Omega \).

**Applications to the Verification of Model Transformations:**

The verification of model transformations is a well-studied (see [HKR+10, NK06, BHE08, EKR+08]) and interesting playground for the techniques presented here. So far, we concentrated on the *in-situ* transformations, where a part of the source model is replaced by a part of the target model. Other approaches construct the target model while parsing the source model, but do not destroy the source model (See [K05, SK08, HKR+10]). For the verification of a model transformation, three sets of rules are needed. The semantic rules of the source model, of the target model and the set of in situ transformation rules. Our ideas can be used to cover for example: NACs on the semantic rules [RKE08], GACs on the semantic rules and GACs on the transformation rules.

The last topic is not just interesting for classical model transformation, but also for system migration, where subsystems of a model are replaced by other systems. It might be possible to derive a migration plan, that describes in which order the systems must be replaced (under which contexts the systems must be), in order to obtain a behaviorally equivalent system after every migration step. The in-situ approach fits perfect into this setting, since the entire system is not changed in one big step, but in several smaller steps with longer periods between them. Processes like this are a wide-spread source of errors and problems in the IT industry (changing the firewall / the database, migrating to a new authentication mechanism, switch from POP3 to IMAP, from DNS to DNSSEC, from IPv4 to IPv6, etc.).

**Acknowledgements:** The ideas and techniques, presented in this paper, were not discovered by the author alone, but are part of a larger work package the author participates on. Therefore, the author likes to thank Filippo Bonchi, Alexandra Silva, H.J. Sander Bruggink, Christoph Blume, Jan Stückrath, Tobias Heindel and especially Barbara König for all the help, ideas and fruitful discussions.

**Bibliography**

Application Conditions for Reactive Systems


