View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Electronic Communications of the EASST (European Association of Software Science and...

Electronic Communications of the EASST
Volume 38 (2010)

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 Hülsbusch

14 pages

Guest Editor: Andrea Corradini
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Application Conditions for Reactive Systems with Applications to Bisimulation Theory

Mathias Hülsbusch[1]*

[1] Abteilung für Informatik und Angewandte Kognitionswissenschaft,
Universität Duisburg-Essen, Germany
mathias.huelsbusch@uni-due.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 $\mathbb{C}$, 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 $\mathbb{C}$ [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.

* Partially supported by DFG project BehaviorGT

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 concider in this paper:

Definition 1 (*DPO* Rewriting) Let $\mathbb{C}$ be an adhesive category. An object G can be rewritten to H via a *DPO* rule $p = L \overset{l}{\leftarrow} I \overset{r}{\rightarrow} R$, 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.

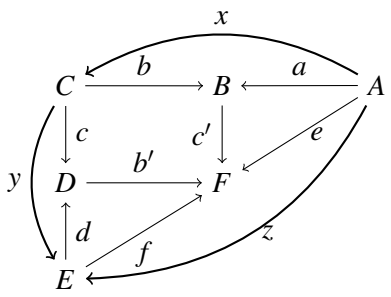
$$\begin{array}{ccccc} L & \xleftarrow{\text{l}} & I & \xrightarrow{\text{r}} & R \\ \downarrow & & \downarrow & & \downarrow \\ G & \longleftarrow & D & \longrightarrow & H \end{array}$$

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

$$\begin{array}{ccccc} J & \xrightarrow{s_l} & I & \xleftarrow{s_r} & J \\ & \searrow^{a} & \downarrow c & \swarrow_{b} & \\ & & K & & \end{array}$$

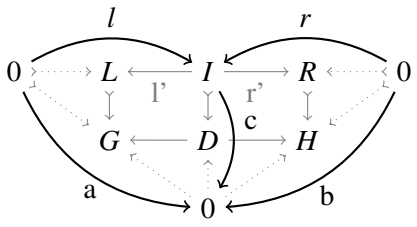
In this paper, we will consider only categories that have an initial object 0 and rules, where J is the initial object of $\mathbb{C}$.

Definition 3 (Cospan category) Let $\mathbb{C}$ be a category with pushouts. A cospan $c\colon \mathsf{A} \xrightarrow{a} \mathsf{B} \xleftarrow{b} \mathsf{C}$ is a pair of $\mathbb{C}$-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 $x = \mathsf{A} \xrightarrow{a} \mathsf{B} \xleftarrow{b} \mathsf{C}$ and $y = \mathsf{C} \xrightarrow{c} \mathsf{D} \xleftarrow{d} \mathsf{E}$ is done via a pushout construction, as shown in the diagram below (where $f = d;b'$ and $e = a;c'$):



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 $\mathsf{cospan}(\mathbb{C})$ is defined as follows: The objects of $\mathsf{cospan}(\mathbb{C})$ are the objects of $\mathbb{C}$, the arrows of $\mathsf{cospan}(\mathbb{C})$ are the semi-abstract cospans.

Reactive systems rewriting over the category $\mathsf{cospan}(\mathbb{C})$ coincides with DPO rewriting in $\mathbb{C}$ [SS05]. Since we work in a setting, where the vertical morphisms in the *DPO* diagram must be monos, we consider the category $\mathsf{cospan}_l(\mathbb{C})$, 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, $\mathsf{cospan}_l(\mathbb{C})$ 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 $\mathbb{C}$. The objects that are rewritten in $\mathbb{C}$ are the inner objects of the cospans a and b (G is rewritten to H via the rule (l',r')).



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[1] 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 $\mathbb{C}$. GAC_A is the set of all triples (A,Q,S) where $Q \in \{\forall,\exists\}$, and S is a set of pairs $(\varphi_i,\mathscr{A}_i)$, where $\varphi_i : A \to A_i$ and $\mathscr{A}_i \in \mathsf{GAC}_{A_i}$. Let $c : A \to B$ be an arrow of $\mathbb{C}$ and $\mathscr{A}$ an element of GAC_A. $c \models \mathscr{A}$ if and only if:

- $c \models (A,\forall,S)$ if for all $(\varphi_i,\mathscr{A}_i) \in S$ and for all $d : A_i \to B$ with $\varphi_i;d = c$ it holds that $d \models \mathscr{A}_i$.
- $c \models (A,\exists,S)$ if there exists a pair $(\varphi_i,\mathscr{A}_i) \in S$ and a morphism $d : A_i \to B$ with $\varphi_i;d = c$ and $d \models \mathscr{A}_i$ holds.

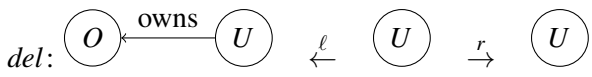
The object A is called *root object* of $\mathscr{A}$ ($\mathrm{RO}(\mathscr{A})$).

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 $\mathbb{C}$ and Q is in $\{\forall,\exists\}$ and the edges are labeled with morphisms of $\mathbb{C}$. 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 $m\colon L \to G$. For a reactive system rule $(l : 0 \to I, r : 0 \to I)$, 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.

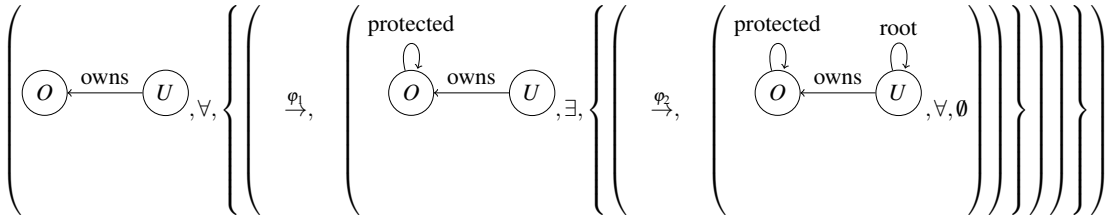
Definition 5 (MonoGAC) $\mathsf{MonoGAC}_A$ is the set of all triples (A,Q,S) from GAC_A, where for every pair $(\varphi_i,\mathscr{A}_i)$ in S holds: φ_i is a mono and $\mathscr{A}_i \in \mathsf{MonoGAC}_{A_i}$. A MonoGAC is evaluated like a normal GAC but all the morphisms d from the objects in the 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:



[1] Nested application conditions allow nesting of quantifiers.

The GAC $\mathscr{A}$, describing the required conditions looks as follows (The mappings of φ_1 and φ_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 $\mathscr{A} = (A, Q, S)$ and $\mathscr{B}$ be from GAC_A and $c : A \to K$.

Negation: $\neg\mathscr{A}$

$$\neg\mathscr{A} = \begin{cases} (A, \exists, S') & \text{if } Q = \forall \\ (A, \forall, S') & \text{if } Q = \exists \end{cases}$$
$$\text{where } S' = \{(\varphi_i, \neg\mathscr{A}_i) \mid (\varphi_i, \mathscr{A}_i) \in S\}$$

Conjunction: $\mathscr{A} \wedge \mathscr{B}$

$$\mathscr{A} \wedge \mathscr{B} = (A, \forall, \{(id_A, \mathscr{A}), (id_A, \mathscr{B})\})$$

Disjunction: $\mathscr{A} \vee \mathscr{B}$

$$\mathscr{A} \vee \mathscr{B} = (A, \exists, \{(id_A, \mathscr{A}), (id_A, \mathscr{B})\})$$

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 $\mathbb{C}$ can be translated into a GAC for the corresponding reactive system rule in $\mathsf{cospan}_l(\mathbb{C})$.

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

The rules p and $\tilde{p}$ and the corresponding GACs are *equivalent* if for every rewriting step $G \rightarrow_p G'$, the step $(0 \rightarrow \mathsf{G} \leftarrow 0) \rightarrow_{\tilde{p}} (0 \rightarrow \mathsf{G}' \leftarrow 0)$ 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 $\mathscr{A} = (L, Q, S)$ can be transformed into an equivalent MonoGAC $\hat{\mathscr{A}} = (I, Q, \hat{S})$, 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 $\mathscr{A} = (L, Q, G)$ from $\mathsf{MonoGAC}_L$ and a morphism $\ell : I \rightarrow L$, the shifting $\mathscr{A}_{\overleftarrow{\ell}}$ of $\mathscr{A}$ via ℓ is defined as follows:

$$\mathscr{A}_{\overleftarrow{\ell}} = \left(I, Q, G'\right), \text{ where}$$
$$G' = \left\{ (\varphi', \mathscr{A}'_{\overleftarrow{\ell'}}) \mid I \overset{\varphi'}{\rightarrowtail} D \overset{\ell'}{\rightarrow} \mathrm{RO}(\mathscr{B}) \text{ is the pushout complement of } I \overset{\ell}{\rightarrow} L \overset{\varphi}{\rightarrowtail} \mathrm{RO}(\mathscr{B}) \text{ and } (\varphi, \mathscr{B}) \in G \right\}$$

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

The second part will then show how $\hat{\mathscr{A}} = \mathscr{A}_{\overleftarrow{\ell}}$ is translated into $\tilde{\mathscr{A}}$ over $\mathsf{cospan}_l(\mathbb{C})$. 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 $\mathscr{A} = (I, Q, S)$ over an adhesive category $\mathbb{C}$ is converted into a GAC $\mathscr{A}^{co}$ over the category $\mathsf{cospan}_l(\mathbb{C})$ by the following construction:

$$\mathscr{A}^{co} = (I, Q, S'), \text{ where}$$
$$S' = \left\{ \left(\mathsf{X} \overset{\varphi_i}{\rightarrow} \mathsf{Y} \overset{id}{\leftarrow} \mathsf{Y}, \mathscr{A}_i^{co} \right) \mid (\varphi_i, \mathscr{A}_i) \in S \right\}$$

Part I

Proof.
Let the following pushout diagram be given (h and m are monos) and let $\mathscr{A}$ be from $\mathsf{MonoGAC}_L$

$$\begin{array}{ccc} I & \xrightarrow{h} & K \\ {\scriptstyle \ell}\downarrow & & \downarrow{\scriptstyle g} \\ L & \xrightarrow[m]{} & G \end{array}$$

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

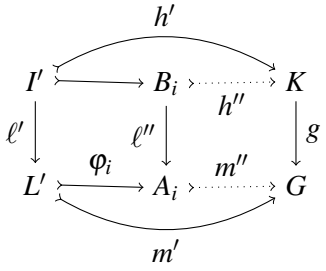
$$(m \models \mathscr{A}) \Leftrightarrow \left(h \models \mathscr{A}_{\underset{\ell}{\leftarrow}}\right)$$

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

$Q = \forall$: Given a GAC $\mathscr{A}' = (L', \forall, \emptyset)$ and a morphism $\ell' : I' \to L'$, then, $\mathscr{A}'_{\underset{\ell'}{\leftarrow}}$ is $(I', \forall, \emptyset)$. For every morphism m' $m' \models \mathscr{A}$ holds. And for every h' $h' \models \mathscr{A}'_{\underset{\ell'}{\leftarrow}}$. Therefore, $\mathscr{A}'$ and $\mathscr{A}'_{\underset{\ell'}{\leftarrow}}$ are equivalent.

$Q = \exists$: Given a GAC $\mathscr{A}' = (L', \exists, \emptyset)$ and a a morphism $\ell' : I' \to L'$, then, $\mathscr{A}'_{\underset{\ell'}{\leftarrow}}$ is $(I', \exists, \emptyset)$. There is no morphism m' such that $m' \models \mathscr{A}$ holds , and there is no h', such that $h' \models \mathscr{A}'_{\underset{\ell'}{\leftarrow}}$ holds. Therefore, $\mathscr{A}'$ and $\mathscr{A}'_{\underset{\ell'}{\leftarrow}}$ are equivalent.

Induction step Assume the following situation: $\mathscr{A} = (L', Q, \{(\varphi_i, \mathscr{A}_i) \mid 1 \leq i \leq n\})$, $\ell' : I' \to L'$ and (ℓ', m', h', g) is a pushout (see diagram below). Let the index set C be defined as follows:
$C = \left\{ i \mid A_i \xleftarrow{\varphi_i} L' \xleftarrow{\ell'} I' \text{ has a pushout complement} \right\}$. For each $i \in C$, the situation is depicted in the figure below ($I' \rightarrowtail B_i \to A_i$ is a corresponding pushout complement):



Note: The pushout splitting property in adhesive categories [LS05] implies: Whenever there exists a $m'' : A_i \rightarrowtail G$, making (φ_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 $I' \rightarrowtail B_i \to A_i$ must exist. Since pullbacks preserve monos, h'' is a mono.

To avoid index overloading, we set $\mathscr{A}_{\underset{\ell}{\leftarrow}} = \mathscr{B}$

1: $(m' \models \mathscr{A}) \Rightarrow (h' \models \mathscr{B})$

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

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

2: $(m' \models \mathscr{A}) \Leftarrow (h' \models \mathscr{B})$

$Q = \exists$: $(h' \models \mathscr{B})$ implies the existence of a mono $h'' : B_i \rightarrowtail K$. 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 $h'' \models \mathscr{B}_i \Leftrightarrow m'' \models \mathscr{A}_i$. Therefore $m' \models \mathscr{A}_i$

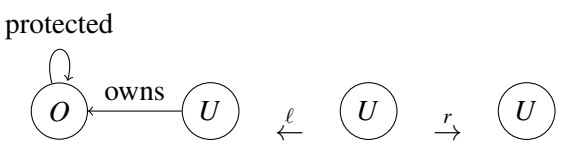
$Q = \forall$: Assume there exists a mono $m'' : A_i \rightarrowtail G$ and $m'' \not\models \mathscr{A}_i$. By the construction given above, there must be a h'', not satisfying $\mathscr{B}_i$. This contradicts that every possible morphism $h'' : B_i \rightarrow K$ satisfies $\mathscr{B}_i$, as the induction hypothesis states.

□

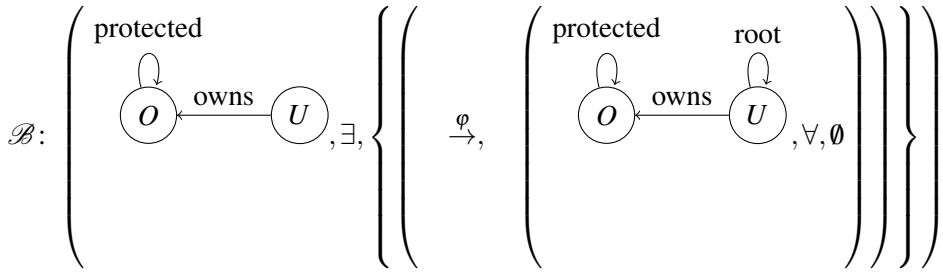
Consider the running example:
It is easy to see, that there is no pushout complement for $\bullet \xrightarrow{\ell} \bullet \xrightarrow{\varphi_1} \bullet$ (dangling edges), and therefore $\mathscr{A}_{\xleftarrow{\ell}}$ is the trivial GAC $\left(\textcircled{U}, \forall, \emptyset\right)$, which 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 del_p, deleting protected objects. The application condition now ensures that the user is root. [2]

del_p:

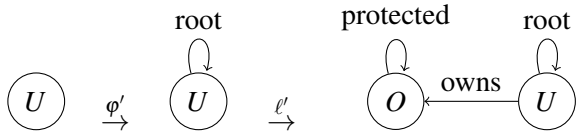


[2] This case can be modeled by a *DPO* rule without application condition, by simply adding the *root* loop to the user node in the LHS, the interface and the RHS, but due to space limitations, we have chosen this small example.

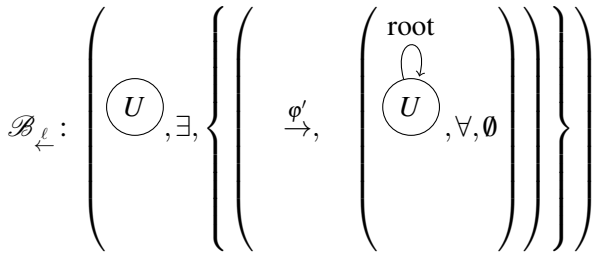


This GAC can be spelled out as: Whenever one wants to delete a protected object, there must *exist* a root loop at the user node.

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



This pushout complement results in:



Part II

Proof.

Let $m = I \rightarrowtail H$ be a mono from $\mathbb{C}$, $\mathscr{A} \in \mathsf{GAC}_I$ and let $c := I \overset{m}{\rightarrowtail} H \leftarrow 0$ be a morphism from $\mathsf{cospan}_l(\mathbb{C})$. It is left to show[3], that the following holds:

$$(m \models \mathscr{A}) \Leftrightarrow (c \models \mathscr{A}^{co})$$

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

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

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

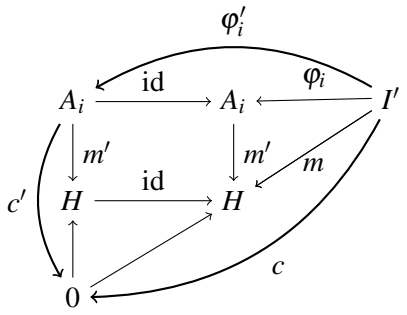
[3] The proof works even for classical *DPO* rewriting (the vertical morphisms are not necessarily monos), but in this setting the constructed cospans are no longer left-linear.

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

Induction step Assume $\mathscr{A} = (I', Q, S)$ with $S = \bigcup_{i=1}^{n} (\varphi_i, \mathscr{A}_i)$ and for all $\mathscr{A}_i$ holds: $(m' \models \mathscr{A}_i) \Leftrightarrow (c' \models \mathscr{A}_i^{co})$, whenever c' is $I \xrightarrow{m'} H \leftarrow 0$.

1:$(m \models \mathscr{A}) \Rightarrow (c \models \mathscr{A}^{co})$

$Q = \exists$**:** From the definition of GACs it follows that there exists a morphism m' and an index i such that $m = \varphi_i; m'$ and $m' \models \mathscr{A}_i$. By induction, we know that $c' = \mathsf{A_i} \xrightarrow{m'} \mathsf{H} \leftarrow 0$ satisfies $\mathscr{A}_i^{co}$. The following diagram shows (since the inner square is a pushout) that for the cospan $\varphi_i' = \mathsf{I'} \xrightarrow{\varphi_i} \mathsf{A_i} \xleftarrow{id} \mathsf{A_i}$ the following holds: $\varphi_i'; c' = c$. Since $c' \models \mathscr{A}_i^{co}$, c satisfies $\mathscr{A}^{co}$.



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

2:$(m \models \mathscr{A}) \Leftarrow (c \models \mathscr{A}^{co})$

$Q = \exists$**:** From the definition of GACs, there must be a cospan $c' = \mathsf{A_i} \xrightarrow{m'} \mathsf{H} \leftarrow 0$ and an index i, such that $c = \varphi_i'; c'$ and $c' \models \mathscr{A}_i^{co}$. By the induction hypothesis, $m' \models \mathscr{A}_i$. Therefore, m satisfies $\mathscr{A}$.

$Q = \forall$**:** Assume there exists a morphism m' and an index i, such that $m = \varphi_i; m'$ and $m' \not\models \mathscr{A}_i$. By the induction hypothesis, the cospan $c' = \mathsf{A_i} \xrightarrow{m'} \mathsf{H} \leftarrow 0$ cannot satisfy $\mathscr{A}_i^{co}$. Since $\varphi_i'; c' = c$, c cannot satisfy $\mathscr{A}^{co}$, which contradicts the assumption.

□

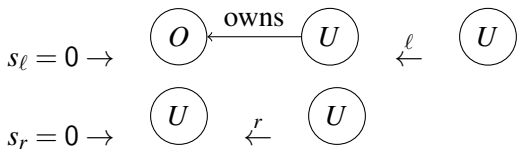
Using these results, the following construction gives an equivalent reactive system rule $\tilde{p}$ with GAC $\tilde{\mathscr{A}}$ over $\mathsf{cospan}_l(\mathbb{C})$ for a *DPO* rule $p = L \xleftarrow{\ell} I \xrightarrow{r} R$ with a MonoGAC $\mathscr{A} \in \mathsf{MonoGAC}_L$

over $\mathbb{C}$.

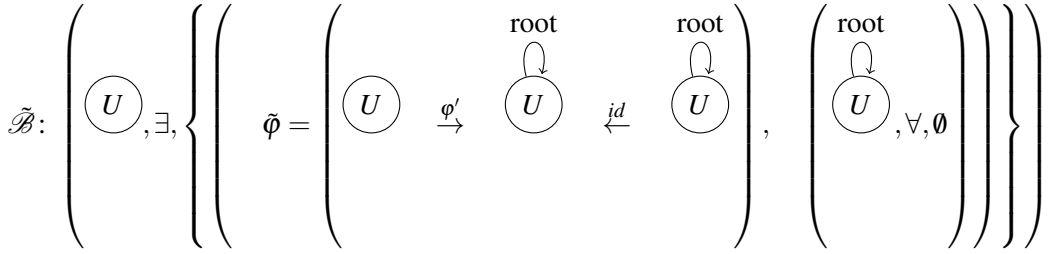
$$\tilde{p} = \left(0 \to \mathsf{L} \overset{\ell}{\leftarrow} \mathsf{I}, 0 \to \mathsf{R} \overset{r}{\leftarrow} \mathsf{I}\right)$$
$$\tilde{\mathscr{A}} = \left(\mathscr{A}_{\overset{\ell}{\leftarrow}}\right)^{co}$$

We have shown that $\mathscr{A}$ is equivalent to $\mathscr{A}_{\overset{\ell}{\leftarrow}}$ and that $\mathscr{A}_{\overset{\ell}{\leftarrow}}$ is equivalent to $\left(\mathscr{A}_{\overset{\ell}{\leftarrow}}\right)^{co}$. Therefore, $\mathscr{A}$ and $\tilde{\mathscr{A}}$ are equivalent.

The construction for the running example:
First, we build the rule $\widetilde{del_p} = (s_\ell, s_r)$ by constructing s_ℓ and s_r.



The GAC $\tilde{\mathscr{B}}$ is finally constructed as:



GACs in $\mathsf{cospan}_l(\mathbb{C})$ 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 $\mathsf{A} \to \mathsf{B} \overset{id}{\leftarrow} \mathsf{B}$.

Definition 9 (finite GAC) A GAC $\mathscr{A} = (I, Q, S)$ 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 $\mathscr{A}$ *in* $\mathsf{cospan}_l(\mathbb{C})$*, such that there is no finite* GAC $\mathscr{B}$ *in* $\mathbb{C}$ *with* $\mathscr{B}^{co} \equiv \mathscr{A}$.

Proof. Let $\mathbb{C}$ be the category of graphs with an infinite alphabet Σ of edge labels, where morphisms must preserve labels. Let $\mathscr{A}$ be defined as $(0, \forall, \{(0 \to \circ \leftarrow 0, (0, \exists, \emptyset))\})$. This representation shows: $\mathscr{A}$ is finite. If a cospan c satisfies $\mathscr{A}$, D (the inner object of the cospan c) contains no isolated node.
Assume a finite GAC $\mathscr{B}$ exists, such that $\mathscr{A} \equiv \mathscr{B}^{co}$. Let L be the set of all edge labels of all the

graphs, contained somewhere in $\mathscr{B}$. Since $\mathscr{B}$ is finite, L is finite. Therefore, there is a label e in Σ, that is not in L. Since there is no edge label e in $\mathscr{B}$, whenever $m = 0 \to (\circ \quad \circ)$ satisfies $\mathscr{B}$, so does $m_e = 0 \to (\circ \xrightarrow{e} \circ)$. This holds, because the edge cannot be in the image of any morphism originating from objects in $\mathscr{B}$, since labels must be preserved. Let c be $0 \to (\circ \xrightarrow{e} \circ) \leftarrow 0$. This gives: $c \not\models \mathscr{A}$, but $m \models \mathscr{B}$, which implies: $\mathscr{A} \not\equiv \mathscr{B}^{co}$. 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. Qur 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 $\mathbb{C}$), 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 $\mathsf{cospan}_l(\mathbb{C})$, 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 Ω exists, bisimilar states are mapped to the same state in Ω. 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 Ω.

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 [KÖ5, 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

[BHE08] D. Bisztray, R. Heckel, H. Ehrig. Verification of Architectural Refactorings by Rule Extraction. In *FASE '08*. LNCS 4961, pp. 347–361. Springer, 2008.

[EK06] H. Ehrig, B. König. Deriving Bisimulation Congruences in the DPO Approach to Graph Rewriting with Borrowed Contexts. *MSCS* 16(6):1133–1163, 2006.

[EKR+08] G. Engels, A. Kleppe, A. Rensink, M. Semenyak, C. Soltenborn, H. Wehrheim. From UML Activities to TAAL - Towards Behaviour-Preserving Model Transformations. In *ECMDA-FA '08*. LNCS 5095, pp. 95–109. Springer, 2008.

[HKR+10] M. Hülsbusch, B. König, A. Rensink, M. Semenyak, C. Soltenborn, H. Wehrheim. Full Semantics Preservation in Model Transformation – A Comparison of Proof Techniques. Technical report TR-CTIT-10-09, Centre for Telematics and Information Technology, University of Twente, 2010.

[KÖ5] A. Königs. Model transformation with Triple Graph Grammars. In *Workshop on Model Transformations in Practice*. 2005.

[LS05] S. Lack, P. Sobocinski. Adhesive and quasiadhesive categories. *Theoretical Informatics and Applications* 39(2):522–546, 2005.
http://eprints.ecs.soton.ac.uk/13599/

[NK06] A. Narayanan, G. Karsai. Towards Verifying Model Transformations. In *GT-VMT '06*. ENTCS 211, pp. 185–194. 2006.

[Pen08] K.-H. Pennemann. Resolution-Like Theorem Proving for High-Level Conditions. In *ICGT '08*. LNCS 5214, pp. 289–304. Springer, 2008.

[Ren04] A. Rensink. Representing First-Order Logic using Graphs. In *ICGT '04*. LNCS 3256, pp. 319–335. Springer, 2004.

[RKE08] G. Rangel, B. König, H. Ehrig. Deriving Bisimulation Congruences in the Presence of Negative Application Conditions. In *FOSSACS '08*. LNCS 4962, pp. 413–427. Springer, 2008.

[Roz97] Handbook of Graph Grammars and Computing by Graph Transformations. In Rozenberg (ed.), *Handbook of Graph Grammars*. Volume Volume 1: Foundations. World Scientific, 1997.

[SK08] A. Schürr, F. Klar. 15 Years of Triple Graph Grammars. In *ICGT '08*. LNCS 5214, pp. 411–425. Springer, 2008.

[SS03] V. Sassone, P. Sobociński. Deriving bisimulation congruences using 2-categories. *Nordic J. of Computing* 10:163–183, May 2003.

[SS05] V. Sassone, P. Sobociński. Reactive systems over cospans. In *LICS '05*. Pp. 311–320. IEEE, 2005.