Electronic Communications of the EASST
Volume X (2008)

Proceedings of the Third Workshop on Petri Nets and Graph Transformations (PNGT 2008)

Negative Application Conditions for Reconfigurable Algebraic High-Level Systems

Alexander Rein

14 pages

Guest Editors: Paolo Baldan, Barbara König
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Negative Application Conditions for Reconfigurable Algebraic High-Level Systems

Alexander Rein

Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin

Abstract: This paper introduces negative application conditions for reconfigurable algebraic high-level systems. These are algebraic high-level systems, i.e. algebraic high-level nets with an initial marking, together with a set of rules for changing the system dynamically. Negative application conditions are a control structure for restricting the application of a rule if a certain structure is present. The use of negative application conditions is motivated in a short example. Subsequently, the underlying theory is sketched and the most significant results are presented. Finally, the example is resumed and the main results and their usefulness within the example are discussed.

Keywords: AHL net, AHL system, net transformation, control structure, negative application condition

1 Introduction

As the adaptation of a system to a changing environment gets more and more important, Petri systems that can be transformed during runtime have become a significant topic in recent years. Their application area ranges over the description of adaptive workflows, the simulation of dynamic processes, multi-agent systems and mobile networks. The extension of these so called reconfigurable P/T systems by data types to reconfigurable algebraic high-level (AHL) systems provides the advantage that the underlying net structure is usually much smaller than the corresponding net without data types. Moreover, the approach of reconfigurable AHL systems increases the expressiveness of AHL systems and allows a formal description of dynamic changes.

In this context, the double pushout approach, presented in [EEPT06], is used for transforming AHL systems. This approach is based on pure categorical constructions and has a lot of instantiations, for example graphs, hypergraphs, Petri nets, Petri systems but also non-visual instantiations like algebraic specifications. In [Pra08], the theory of adhesive high-level replacement (HLR) systems [EEPT06], which is the categorical framework of the double pushout approach, have been instantiated to AHL nets and systems.

In most transformation systems, there exist basic conditions for the applicability of transformations. For example, a transformation should not be applied in some situations, although a consistent match can be found. For conditions like this some kind of control structure for the applicability of rules is required. Control structures are necessary if rules are applied automated and also reasonable if the rules are applied manually for preventing human failures. Negative application conditions (NACs), introduced in Chapter 7 in [EEPT06], are such a control structure for adhesive HLR systems. They restrict the application of a rule if a certain structure is

present. Such a constraint influences each rule application and, therefore, the properties of the replacement system are changed significantly.

For generalizing well-known and important results like Local Church-Rosser Theorem, Completeness Theorem of Critical Pairs, Concurrency Theorem, Embedding and Extension Theorem and Local Confluence Theorem to the use of NACs, the notion of weak adhesive HLR categories with NACs, basing on weak adhesive HLR categories [EEPT06], is introduced in [Lam07, LEOP08].

By proving that AHL systems are a weak adhesive HLR category with NACs, we can transfer all these results to reconfigurable AHL systems.

In [RPL+08], we introduced NACs for reconfigurable P/T systems and motivated their use with the help of a short example. This paper represents the extension of the results to reconfigurable AHL systems with a changeable data type. Moreover, the airport example of [RPL+08] is extended to AHL systems with some additional features.

This paper is organized as follows: First we introduce our example and discuss the need of additional control structures for the application of rules in Section 2. Then we review the formal notions of reconfigurable AHL systems in Section 3. Based on these notions we define negative application conditions and present the main results concerning parallelism, concurrency and confluence in Section 4. We discuss some of the general results with respect to the example in Section 2. Concluding remarks concern future and related work.

2 Example: Airport

This section contains an example for a reconfigurable AHL system with NACs. We model an airport control system (ACS) which is supposed to prevent accidents in the airport area. The system has to ensure that certain safety properties of an airport are satisfied, for example that some areas of the airport such as the actual runways and gates are secure, i.e. exclusively used by one airplane at the time. The system can handle airplanes and gates of different sizes and manage the coordination of the airplanes at the gates. Thereby, the condition that airplanes can only use gates of exactly the same size has to be fulfilled.

ACS can adapt to various changes of the airport. Every transformation step of the AHL system represents the rearrangement of the airport and every firing of a transition reflects a process at the airport. In this small example, changes at runtime may only concern adding and removing of gates of arbitrary sizes.

Figure 1 shows all required algebraic specifications. Specification **SP-ACS0** forms the base of ACS. The sort *apSize* stands for airplane size. Remaining sorts *blackToken* and *airplane* and the operation *getSize* are self-explanatory. $A_{SP\text{-}ACS0}$ is the algebra to this specification, where $\mathbb{N}^+ = \mathbb{N}\backslash\{0\}$ is the set of all positive integers. The carrier set of the sort *airplane* is a tuple of two positive integers – the first number represents the ID of an airplane and the second one the size.

In this example, the names of places are neglected and only labels and types are visualized, i.e. $l : t$ denotes a place p of the type t with a label l, which means $\lambda(p) = l \in L$ and $type(p) = t$.

Figure 2 pictures the startsystem of ACS with one landing runway, one starting runway and one gate of size 1. Firing transition *approach* represents the arrival of an airplane in the airspace

SP-ACS0 =
sorts: $airplane, apSize, blackToken$
opns: $\bullet :\rightarrow blackToken$, $getSize : airplane \rightarrow apSize$
vars: $b_1 : blackToken$
eqns: $b_1 = \bullet$

$A_{SP\text{-}ACS0} = ((\mathbb{N}^+ \times \mathbb{N}^+), \mathbb{N}^+, \{\bullet\}, \bullet, (a,s) \overset{getSize}{\longmapsto} s)$

SP-ACS1 = SP-ACS0 + opns: $size_1 :\rightarrow apSize$

$A_{SP\text{-}ACS1}$ = $A_{SP\text{-}ACS0}$ with the additional constant $size1_{A_{SP\text{-}ACS1}} = 1$

SP-ACS2 = SP-ACS1 + opns: $size_2 :\rightarrow apSize$

$A_{SP\text{-}ACS2}$ = $A_{SP\text{-}ACS1}$ with the additional constant $size2_{A_{SP\text{-}ACS2}} = 2$

SP-ACS = SP-ACS1 + sorts: nat, opns: $getID : airplane \rightarrow nat$

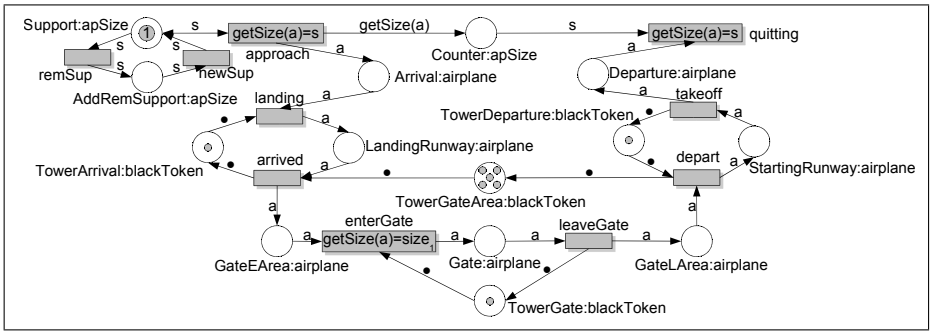
$A_{SP\text{-}ACS}$ = $A_{SP\text{-}ACS1}$ with $A_{SP\text{-}ACS_{nat}} = \mathbb{N}^+$ and $getID_{A_{SP\text{-}ACS}}(a,s) = a$

Figure 1: Algebraic Specifications of ACS

of the airport (place *Arrival*). This transition is only enabled if there is a token of the size of the airplane at the place *Support*. So, only airplanes of supported sizes may enter the airspace and use the airport. The place *AddRemSupport* with the two adjacent transitions is only a help place for adding and removing the support of airplane sizes by applying rules and should not be used in the regular operation of the airport. For each arrived airplane, an airplane token is placed at *Arrival* and a size token of the corresponding size is placed at *Counter*. This place represents a counter for all airplanes at the airport and stores their sizes. Each runway and each gate consists of two places – representing the runway (resp. the gate) itself and a complement place that ensures the exclusive use. Additionally, every runway consists of two transitions *landing* and *arrived*, resp. *depart* and *takeoff*. The transition *landing* of a landing runway is enabled if the runway is not in current use and an arriving airplane is in the airspace of the airport. Firing of this transition leads to a token representing an airplane on the runway. In the lower part of the AHL system, the gate area is modeled. The tokens at place *TowerGateArea* represent the capacity of the gate area. To simplify matters, this capacity is fixed in this example. The first and only gate of the airport is represented by the places *Gate* and the complement place *TowerGate* of this gate. So the exclusive use of the gate by only one airplane is guaranteed. The condition of the transition *enterGate* ensures that only airplanes of size 1 can use this gate.

In the following, the transformation rules of ACS, depicted in Figure 3, are described. To apply a rule to an AHL system a match from the left-hand side L of the corresponding rule to the AHL system has to be found. Then the satisfication of the NACs has to be checked. A NAC is satisfied if a morphism from the NAC to the AHL system via the match does not exist.

Rule 1 and 2 in the top of Figure 3 describe the adding and the removing of airplane sizes.



SP-ACS, $A_{SP\text{-}ACS}$

Figure 2: ACS Startsystem

These rules have an empty net component and change only the specification of an airport net. Adding a new size is modeled by adding a new constant of the sort *apSize* to the AHL system. Its value is determined by the match. Note that arbitrary sizes can be added through different matches. This is feasible although algebra homomorphisms are restricted to isomorphisms because the carrier sets are not changed. A negative application condition is required to prevent the creation of multiple constants with the same value. The second constant of the sort *apSize* in the right-hand side is required to prevent a mapping from sort *apSize* to sort *nat* and operation *getSize* to *getID*. Removing the last constant of the sort *apSize* is restricted by the gluing condition since the last gate cannot be removed. This transformation can be revoked by applying rule 2, which is inverse to rule 1 without the negative application condition.

Adding a gate to the airport requires two rules (3a and 3b in Figure 3). The procedure for the first gate of a size, which is expressed by rule 3a, requires the additional treatment that the support of this size is added to the airport. After applying rule 3a, the transition *newSup* is fired so that airplanes of the new supported size can use the airport. A negative application condition is required to restrict this rule from being applied if a token of this size is at place *Support*, i.e. at least one gate of this size exists. Note that this rule cannot be applied if there is a token at the place *AddRemSupport* because of the gluing condition (see point 8. in Definition 8). This is because this rule formally deletes the place *AddRemSupport* with the two adjacent transitions and then re-adds this structure with a token at this place. In contrast, rule 3b can only be applied if a token of the size of the gate to be added exists, i.e. the airport already has a gate of this size. This rule simply adds the new gate.

To remove gates, the use of the inverse rules of the rules for adding gates is not sufficient because of the additional condition that the last gate of the airport may not be removed. This procedure is described by rules 4a and 4b in the bottom of Figure 3. Rule 4a expresses the removal of a gate under the assumption that at least one gate of the same size exists and rule 4b models the removal of the last gate of a size. Two negative application conditions are required for rule 4b. The first one (NAC_1) restricts the rule from being applied if there are still airplanes of this size at the airport. The second NAC (NAC_2) guarantees that this rule can only be applied if the gate to be removed is the last gate of its size. Note that identifying the constants $size_1 = 1$ and $size_2 = 2$ by a match is not possible since this would need a non-isomorphic algebra

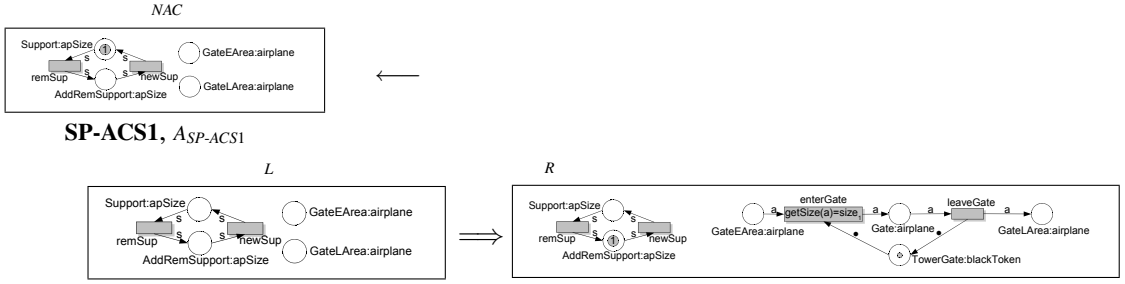
Rule 1: *addNewSize*

$$\textbf{SP-ACS2}, A_{SP\text{-}ACS2} \overset{NAC}{\longleftarrow} \textbf{SP-ACS1}, A_{SP\text{-}ACS1} \overset{L}{\Longrightarrow} \textbf{SP-ACS2}, A_{SP\text{-}ACS2} \overset{R}{}$$

Rule 2: *removeSize*

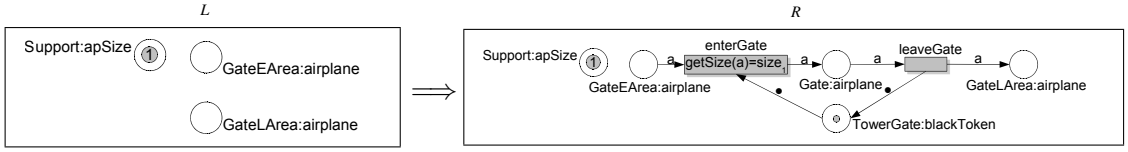
Inverse to rule 1 without NAC.

Rule 3a: *addFirstGate*



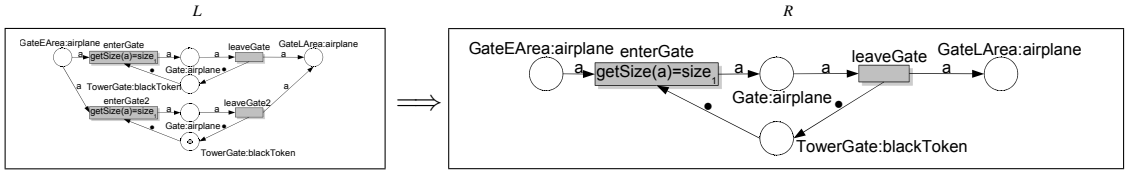
SP-ACS1, $A_{SP\text{-}ACS1}$ **SP-ACS1**, $A_{SP\text{-}ACS1}$

Rule 3b: *addGate*



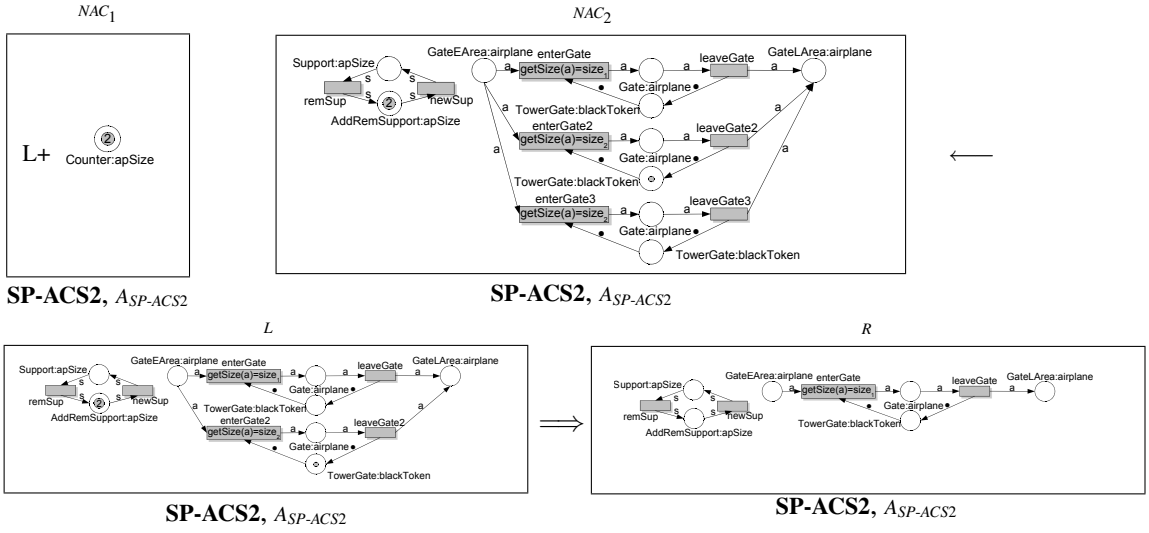
SP-ACS1, $A_{SP\text{-}ACS1}$ **SP-ACS1**, $A_{SP\text{-}ACS1}$

Rule 4a: *removeGate*



SP-ACS1, $A_{SP\text{-}ACS1}$ **SP-ACS1**, $A_{SP\text{-}ACS1}$

Rule 4b: *removeLastGate*



SP-ACS2, $A_{SP\text{-}ACS2}$ **SP-ACS2**, $A_{SP\text{-}ACS2}$

SP-ACS2, $A_{SP\text{-}ACS2}$ **SP-ACS2**, $A_{SP\text{-}ACS2}$

Figure 3: The Rules of ACS

homomorphism.

3 Reconfigurable Algebraic High-Level Nets

In this section, reconfigurable AHL systems are formalized. We use the algebraic notation of "Petri nets are Monoids" in [MM90], extended by a data type and a labeling function for places. So, an AHL net is given by $AN = (SP, P, T, pre, post, cond, type, A, \lambda)$ with algebraic specification $SP = (S, OP, E, X)$ with additional variables X for the net, (S, OP, E)-algebra A, pre- and post-domain functions $pre, post : T \to P^{\oplus}$, firing conditions $cond : T \to \mathscr{P}_{fin}(Eqns(S, OP, X))$, typing of places $type : P \to S$ and a labeling function $\lambda : P \to L$, where L is a fixed alphabet for places and $P^{\oplus}$ is the free commutative monoid over the set P of places. An AHL system is given by an AHL net with an initial marking (AN, M), where $M \in CP^{\oplus}$ with $CP = (A \otimes P) = \{(a, p) | a \in A_{type(p)}, p \in P\}$.

In order to define rules and transformations of AHL systems, we introduce AHL morphisms which are compatible with pre- and post-domains, fire conditions, typing of places and labeling. Additionally, they require that the initial marking at corresponding places is equal or increasing.

Definition 1 (AHL Morphism) Given AHL systems $AS_i = (SP_i, P_i, T_i, pre_i, post_i, cond_i, type_i, A_i, \lambda_i, M_i)$ with $SP_i = (S_i, OP_i, E_i, X_i)$ for $i = 1, 2$, an AHL morphism $f : AS_1 \to AS_2$ is given by $f = (f_{SP}, f_P, f_T, f_A)$, where (f_{SP}, f_A) is a generalized algebra homomorphism with specification morphism $f_{SP} = (f_S, f_{OP}, f_X) : SP_1 \to SP_2$ with a seperate injective mapping of the additional variables $f_X = (f_{X_s})_{s \in S_1} : X_1 \to X_2$ and algebra isomorphism $f_A = (f_{A_s})_{s \in S_1} : A_1 \to V_{f_{SP}}(A_2)$ and $f_P : P_1 \to P_2$ and $f_T : T_1 \to T_2$ are two functions mapping the places and the transitions such that

$$\lambda_1 = \lambda_2 \circ f_P \tag{1}$$

$$\forall (a, p) \in (A_1 \otimes P_1) : M_1(a, p) \leq M_2(f_A(a), f_P(p)) \tag{2}$$

and the following diagrams commute:

$$\begin{array}{ccccc}
\mathscr{P}_{fin}(Eqns(S_1, OP_1, X_1)) & \xleftarrow{cond_1} & T_1 & \underset{post_1}{\overset{pre_1}{\rightrightarrows}} & (T_{OP_1}(X_1) \otimes P_1)^{\oplus} \\
\downarrow{\scriptstyle \mathscr{P}_{fin}(f^{\#}_{SP} \times f^{\#}_{SP})} & (=) & \downarrow{\scriptstyle f_T} & (=) & \downarrow{\scriptstyle (f^{\#}_{SP} \otimes f_P)^{\oplus}} \\
\mathscr{P}_{fin}(Eqns(S_2, OP_2, X_2)) & \xleftarrow[cond_2]{} & T_2 & \underset{post_2}{\overset{pre_2}{\rightrightarrows}} & (T_{OP_2}(X_2) \otimes P_2)^{\oplus}
\end{array}
\qquad
\begin{array}{ccc}
P_1 & \xrightarrow{type_1} & S_1 \\
\downarrow{\scriptstyle f_P} & (=) & \downarrow{\scriptstyle f_S} \\
P_2 & \xrightarrow[type_2]{} & S_2
\end{array}$$

Remark 1

1. *$f^{\#}_{SP}$ is the extension of specification morphism f_{SP} to terms and equations which maps the (specification) variables bijectively.*
2. *$V_{f_{SP}}$ is the forgetful functor with respect to f_{SP}.*
3. *$\mathscr{P}_{fin}$ is the power set functor that maps a set to its power set.*
4. *f_X injective implies $Var(t) \subseteq Var(f_T(t))$ that ensures the preservation of firing.*

Moreover, the AHL morphism f is called strict if and only if f is componentwise injective, $f^{\#^{-1}}_{SP}(E_2) \subseteq E_1$ (strict injectivity) and $\forall (a, p) \in (A_1 \otimes P_1) : M_1(a, p) = M_2(f_A(a), f_P(p))$ (marking strictness).

The category defined by AHL systems and AHL morphisms is denoted by **AHLSystems** where the composition of AHL morphisms is defined componentwise. The class of all strict AHL morphisms is denoted by $\mathscr{M}$.

The gluing condition is a sufficient and necessary condition for the applicability of a rule $L \xleftarrow{l} K \xrightarrow{r} R$ at a given match $m : L \to G$. The so called pushout complement, which is required for the first step of a transformation, exists if and only if the gluing condition is satisfied. Therefore, gluing points, identification points and dangling points are defined. In this context, points means places, transitions, sorts, operations, terms or equations. Gluing points are all points of L that have a preimage in K, identification points are all points of L that are not mapped injectively by m and dangling points are all points in L with an image in G that is adjacent with a structure without a preimage in L. The gluing condition is satisfied if all identification and dangling points are also gluing points. The formal definition of the gluing condition for AHL systems can be found in the appendix (Definition 8).

Next, we present rule-based transformations of AHL systems following the double-pushout (DPO) approach of graph transformations in the sense of [Roz97, EEPT06].

Definition 2 (AHL System Rule) Given AHL systems $AS_i = (SP_i, P_i, T_i, pre_i, post_i, cond_i, type_i, A_i, \lambda_i, M_i)$ with $SP_i = (S_i, OP_i, E_i, X_i)$ for $i \in \{1, L, K, R\}$, then a rule $rule = (AS_L \xleftarrow{l} AS_K \xrightarrow{r} AS_R)$ consists of AHL systems AS_L, AS_K, and AS_R, called left-hand side, interface, and right-hand side of $rule$ and two strict AHL morphisms $l : AS_K \to AS_L$ and $r : AS_K \to AS_R$.

The rule $rule$ is applicable at the match $m : AS_L \to AS_1$ if and only if the gluing condition (see Definition 8) is satisfied for l and m. In this case, we obtain AHL system AS_0 leading to a transformation step $AS_1 \overset{rule,m}{\Rightarrow} AS_2$ consisting of the following pushout diagrams (1) and (2). The AHL morphism $n : AS_R \to AS_2$ is called comatch of the transformation step.

$$\begin{array}{ccccc} AS_L & \xleftarrow{l} & AS_K & \xrightarrow{r} & AS_R \\ {\scriptstyle m}\downarrow & (1) & \downarrow{\scriptstyle c} & (2) & \downarrow{\scriptstyle n} \\ AS_1 & \xleftarrow[l^*]{} & AS_0 & \xrightarrow[r^*]{} & AS_2 \end{array}$$

Now we are able to define reconfigurable AHL systems, which allow the modification of the net structure using rules and transformations of AHL systems.

Definition 3 (Reconfigurable AHL System) Given an AHL system AS and a set of rules $RULES$, a reconfigurable AHL system is defined by $(AS, RULES)$.

In the example in Section 2, the reconfigurable AHL system consists of the AHL system depicted in Figure 2 and the set of rules pictured in Figure 3. Note that the application of some of these rules is restricted by NACs and we will present the notion of reconfigurable AHL systems with NACs in Section 4.

4 Negative Application Conditions

In this section, we first introduce the main technical result that **AHLSystems** is a weak adhesive HLR category with NACs. This means, the category is a weak adhesive HLR category and satisfies some additional properties such that negative application conditions can be used within AHL system transformations without losing important qualities of adhesive HLR systems like Local Church-Rosser Theorem, Parallelism Theorem, Completeness Theorem of Critical Pairs, Concurrency Theorem, Embedding and Extension Theorem and Local Confluence Theorem. All these theorems are introduced in [EEPT06] and their extension for the use of NACs is presented in [LEOP08, Lam07].

In addition to morphism class $\mathscr{M}$, presented in Section 3, two other morphism classes are required. On the one hand there is morphism class $\mathscr{Q}$ connecting the NAC and the source net, which is the class of all (componentwise) injective morphisms, and on the other hand there is a class of morphism pairs $\mathscr{E}$, which is mainly used for constructions and proofs. This morphism class consists of jointly equation strict and minimal jointly surjective AHL morphisms. Minimal means that the markings in the codomain are as small as possible and equation strict means that the codomain contains exactly the translated equations of both domains, i.e. for $f_1 : AS_1 \to AS_3$, $f_2 : AS_2 \to AS_3$ with $(f_1, f_2) \in \mathscr{E}$ we have that (1.) f_1, f_2 are (componentwise) jointly surjective, (2.) $M_3(a_3, p_3) = max(\{M_1(a,p) | a \in f_{1_A}^{-1}(a_3) \wedge p \in f_{1_P}^{-1}(p_3)\} \cup \{M_2(a,p) | a \in f_{2_A}^{-1}(a_3) \wedge p \in f_{2_P}^{-1}(p_3)\})$ (are minimal) and (3.) $E_3 = f_{1_{SP}}^{\#}(E_1) \cup f_{2_{SP}}^{\#}(E_2)$ (are jointly equation strict).

Definition 4 (Morphism classes in **AHLSystems**) Given the category **AHLSystems** of AHL systems and AHL morphisms, then the following morphism classes are defined:

$\mathscr{M}$	:	strict AHL morphisms
$\mathscr{Q}$	:	injective AHL morphisms
$\mathscr{E}$	:	jointly equation strict and minimal jointly surjective AHL morphisms

Theorem 1 (**AHLSystems** is a Weak Adhesive HLR Category with NACs) *The category* **AHLSystems** *with the morphism classes $\mathscr{M}$, $\mathscr{Q}$ and $\mathscr{E}$ as defined above is a weak adhesive HLR category with NACs, i.e. a weak adhesive HLR category with the following additional properties:*

1. *unique $\mathscr{E}$-$\mathscr{Q}$ pair factorization,*
2. *unique epi-$\mathscr{M}$ factorization,*
3. *$\mathscr{M}$-$\mathscr{Q}$ pushout-pullback decomposition property,*
4. *initial pushouts over $\mathscr{Q}$-morphisms,*
5. *$\mathscr{Q}$ is closed under pushouts and pullbacks along $\mathscr{M}$-morphisms,*
6. *induced pullback-pushout property for $\mathscr{M}$ and $\mathscr{Q}$ and*
7. *$\mathscr{Q}$ is closed under composition and decomposition*

The proof that **AHLSystems** is a weak adhesive HLR category can be found in [Pra08] and a description and the proof of the additional properties can be found in [Rei08]. Now we can state negative application conditions for reconfigurable AHL systems.

Definition 5 (Negative Application Condition) A negative application condition of a rule $rule = (L \xleftarrow{l} K \xrightarrow{r} R)$ in the weak adhesive HLR category with NACs $(\mathbf{C}, \mathscr{M}, \mathscr{E}, \mathscr{Q})$ is of the form $NAC(n)$, where $n : L \to N$ is a $\mathbf{C}$-morphism.
A morphism $m : L \to G$ satisfies $NAC(n)$, written $m \models NAC(n)$, if there does not exist a morphism $q : N \to G \in \mathscr{Q}$ with $q \circ n = m$.

Definition 6 (Rule with NACs) A rule in a weak adhesive HLR category with NACs $(\mathbf{C}, \mathscr{M}, \mathscr{E}, \mathscr{Q})$ with a set of negative application conditions *NACS* is called rule with NACs.

Definition 7 (Applicability of a Rule with NACs) Given a rule $rule = (L \xleftarrow{l} K \xrightarrow{r} R)$ with a set of negative application conditions *NACS* and a match $m : L \to G$ such that *rule* without NACs is applicable at m, then the rule *rule* with NACs is applicable if and only if m satisfies all NACs of the set *NACS*.

For these new rules and their restricted application we obtain the same results as known for net transformations in general. These results have been shown for NACs at the level of weak adhesive HLR categories in [LEOP08, LEO06, Lam07]. Their instantiation to AHL systems requires Theorem 1 above.

Results (For Reconfigurable AHL Systems with NACs)

1. **Local Church-Rosser and Parallelism:** Simplified, the Local-Church Rosser property states that the order of two sequential transformations does not matter if they are sequentially independent, i.e. one transformation does not create or delete some structure that the other one requires. In this case, a parallel transformation that performs both transformations in one step exists.

   In the airport example of Section 2, it makes no difference if a gate of size 1 or a new size is added first to the startsystem. The adding of a gate of size 1 is obviously still possible after adding a new size and the inverse direction also holds. Moreover, both transformation steps can be combined to a parallel transformation, which adds both a new size and a gate of size 1 to the airport.

2. **Conflicts and Critical Pairs:** A critical pair describes a conflict between two transformations in a minimal context. A conflict between two transformations means in particular that either one transformation deletes some structure that is used by the other one or one transformation produces a structure that is forbidden by the NAC of the other transformation. The critical pairs and their completeness are a significant concept because an infinite number of transformations, which are in conflict, can be reduced to a finite number of critical pairs (under the assumption that the set of rules is finite). Critical pairs are used to show the (local) confluence of a transformation system, which is described later.

   Consider in the airport example of Section 2 an airport system corresponding to the startsystem shown in Figure 2 with a second constant $size_2 = 2$. The transformation that removes the size $size_2$ is in conflict with the transformation that adds a new gate of $size_2$ to the airport system. This pair of transformations causes a delete-use conflict (with respect

to constant $size_2$), which is expressed by the critical pair shown in Figure 4.

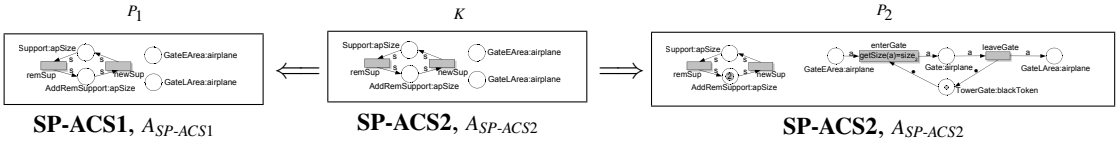


Figure 4: Critical Pair for Rules *remSize* and *addFirstGate*

3. **Concurrency:** Through parallelism, the summary of several independent direct transformations into one equivalent direct transformation is possible. However, in general there are dependencies between several direct transformations of a transformation sequence. In this case, the Concurrency Theorem with NACs that allows to translate every transformation sequence into an equivalent direct transformation by a concurrent rule with NACs can be applied. The construction of such a concurrent rule with NACs is explained in [LEOP08]. A concurrent rule summarizes in one rule which parts of the net should be present, preserved, deleted and produced when applying the corresponding rule sequence to this net. Moreover, we have a summarized set of NACs on the concurrent rule expressing which net parts are forbidden when applying the corresponding rule sequence with NACs to the net.

   Consider in the airport example of Section 2 the airport system described in the last item and the following transformation sequence. First, $size_2$ is removed, then $size_2$ is added again and, finally, a gate of $size_2$ is added. This sequence can be expressed by the concurrent rule depicted in Figure 5. Note that the interface object K of this rule contains the specification **SP-ACS1** and not **SP-ACS2**. This concurrent rule holds two NACs, which originate by the translation of the NACs of the single rules [Lam07].

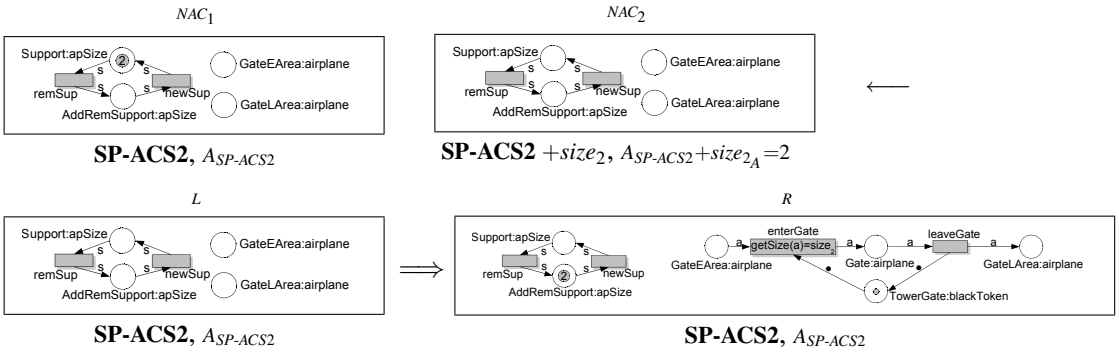


Figure 5: Concurrent Rule with NACs

4. **Embedding and Extension:** Under several conditions, a transformation $t : G_0 \overset{*}{\Rightarrow} G_n$ can be extended to a transformation $t' : G'_0 \overset{*}{\Rightarrow} G'_n$ via an extension morphism $k_0 : G_0 \to G'_0$. The transformation t' is based on the same rules as t. The Embedding Theorem with NACs describes a condition for the existence of this so called embedding and the Extension The-

orem with NACs states that this condition is not only sufficient, but also necessary.
For the embedding of a transformation, the extension morphism k_0 has to be boundary consistent [EEPT06], which means intuitively that places which are deleted by the transformation t cannot be embedded into places connected with new transitions in the bigger AHL system G'_0, and NAC-consistent [LEO06], which means intuitively that the extended match satisfies the NACs of the rule.

5. **Local Confluence:** Confluence describes the behaviour of a whole transformation system. A pair of transformations of the same source object is called confluent if it is possible to transform the two results of the single transformations into the same object. A transformation system is called locally confluent if this property holds for every pair of transformations. Confluence is the main property of interest for adhesive HLR systems (with NACs). The reasons therefore are quite obvious. Every transformation system which provides multiple possibilities to transform the start object into a different object shows this behaviour for at least one pair of transformations. In the small example of this paper, this result is almost obvious since every transformation step is reversible, although, this result is nontrivial in general, e.g. for transformation systems with non-reversible transformation steps. A sufficient condition for the local confluence of an adhesive HLR system with NACs is if every critical pair is strict NAC-confluent [Lam07]. Intuitively, this means that a strict solution for every critical pair exists, i.e. both results of the critical pair can be transformed into the same result and this solution preserves everything that is preserved in common by the critical pair itself. Moreover, whenever the embedding of a critical pair into some larger context is possible, the extension morphism should be NAC-consistent with respect to the critical pair solution. This means in particular that all NACs occurring in the solution of the critical pair are still satisfied by the embedding into the larger context.

   Consider in the example of Section 2 the critical pair shown in Figure 4. The solution of this critical pair is shown in Figure 5 for the left-hand side P_1. This solution is strictly confluent because the places *Support*, *GateEArea* and *GateLArea* are preserved both by the critical pair and the solution. The satisfaction of the first NAC of the concurrent rule of this solution (see Figure 5) is directly implied by the NAC of the critical pair (i.e. the NAC of rule *addFirstGate*). The second NAC is always fulfilled in the case of ACS since two constants with the same value cannot be created by applying rules to the startsystem. Therefore, strict NAC-confluence [Lam07] holds with respect to the language generated by grammar ACS, which is sufficient for a reconfigurable AHL system since a startsystem is always included. This result means that every conflict, where this critical pair can be embedded, can also be resolved. If every critical pair of a transformation system is strictly NAC-confluent, then the whole system is locally confluent.

5 Conclusion

We conclude with a short discussion of related and future work:
Related Work: Reconfigurable nets base on arbitrary net transformations. This approach can be restricted to distinguished transformations that preserve specific properties as safety or live-

ness (see [PU03]). A different approach for changing Petri nets is presented in [LO04, LO06a, LO06b], where rewriting of Petri nets in terms of graph grammars is used for the reconfiguration of nets. These marked-controlled reconfigurable nets (MCRN) are extended by some control technique that allows changes of the net for specific markings. The enabling of a rule is not only dependent on the net topology, but also dependent on the marking of specific control places. *MCReNet* [LO06a] is the corresponding tool for the modeling and verification of MCRNs.

Future Work: One ongoing research task is the extension of this paper's results to open algebraic high-level systems, which are AHL systems with additional open places and communication transitions (see [Ull08]). Therefore, the same conditions have to be proven for the category **OAHLSystems** of open AHL systems and open AHL morphisms. Another one are algebraic higher order (AHO) nets (see [HEM05]), which allow the existence of dynamical tokens like Petri systems and transformation rules. These nets are used for modeling workflows of mobile ad-hoc networks. Up to now, neither NACs nor AHL systems can be used within AHO nets. Therefore, the data type of AHO nets has to be upgraded, such that both AHL systems and NACs are formalized in this data type.

References

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs. Springer, 2006.

[HEM05] K. Hoffmann, H. Ehrig, T. Mossakowski. High-Level Nets with Nets and Rules as Tokens. In *Proceedings of ICATPN'05*. LNCS 3536, pp. 268–288. Springer, 2005.

[Lam07] L. Lambers. Adhesive High-Level Replacement Systems with Negative Application Conditions. Technical report, Technische Universität Berlin, 2007.

[LEO06] L. Lambers, H. Ehrig, F. Orejas. Conflict Detection for Graph Transformation with Negative Application Conditions. In *Proceedings of ICGT'06*. LNCS 4178, pp. 61–76. Springer, 2006.

[LEOP08] L. Lambers, H. Ehrig, F. Orejas, U. Prange. Parallelism and Concurrency in Adhesive High-Level Replacement Systems with Negative Application Conditions. *ENTCS*, 2008. to appear.

[LO04] M. Llorens, J. Oliver. Structural and Dynamic Changes in Concurrent Systems: Reconfigurable Petri Nets. *IEEE Transactions on Computers* 53(9):1147–1158, 2004.

[LO06a] M. Llorens, J. Oliver. A Basic Tool for the Modeling of Marked-Controlled Reconfigurable Petri Nets. *ECEASST* 2:13, 2006.

[LO06b] M. Llorens, J. Oliver. Marked-Controlled Reconfigurable Workflow Nets. In *SYNASC*. Pp. 407–413. IEEE Computer Society, 2006.

[MM90] J. Meseguer, U. Montanari. Petri Nets are Monoids. *Information and Computation* 88(2):105–155, 1990.

[Pra08] U. Prange. Towards Algebraic High-Level Systems as Weak Adhesive HLR Categories. *ENTCS*, 2008. to appear.

[PU03] J. Padberg, M. Urbášek. Rule-Based Refinement of Petri Nets: A Survey. In *Petri Net Technology for Communication-Based Systems*. LNCS 2472, pp. 161–196. Springer, 2003.

[Rei08] A. Rein. Reconfigurable Petri Systems with Negative Application Conditions. Technical report 2008/01, Technische Universität Berlin, 2008. Diploma Thesis.

[Roz97] G. Rozenberg (ed.). *Handbook of Graph Grammars and Computing by Graph Transformation, Vol 1: Foundations*. World Scientific, 1997.

[RPL+08] A. Rein, U. Prange, L. Lambers, K. Hoffmann, J. Padberg. Negative Application Conditions for Reconfigurable Place/Transition Systems. *Electronic Communications of the EASST*, 2008.

[Ull08] C. Ullrich. Reconfigurable Open AHL Systems. Technical report 2008/10, Technische Universität Berlin, 2008. Diploma Thesis.

A Gluing Condition

In this section, we formalize the gluing condition for AHL systems. It is a sufficient and necessary condition for the applicability of a rule via a match. The proof therefore can be found in [Rei08].

Definition 8 (Gluing Condition for **AHLSystems**) Given AHL systems $AS_i = (SP_i, P_i, T_i, pre_i, post_i, cond_i, type_i, A_i, \lambda_i, M_i)$ with $SP_i = (\Sigma_i, E_i, X_i)$ and $\Sigma_i = (S_i, OP_i)$ for $i \in \{1, L, K\}$ and AHL morphisms $l : AS_K \to AS_L \in \mathscr{M}$ and $m : AS_L \to AS_1$, then the gluing points GP, the dangling points DP and the identification points IP are defined by

$$
\begin{aligned}
GP(P) &= l_P(P_K),\\
GP(T) &= l_T(T_K),\\
GP(S) &= l_S(S_K),\\
GP(OP) &= l_{OP}(OP_K),
\end{aligned}
$$

$$
\begin{aligned}
IP(P) &= \{\ p \in P_L | \exists p' \in P_L : p \neq p' \wedge m_P(p) = m_P(p')\}\\
IP(T) &= \{\ t \in T_L | \exists t' \in T_L : t \neq t' \wedge m_T(t) = m_T(t')\}\\
IP(S) &= \{\ s \in S_L | \exists s' \in S_L : s \neq s' \wedge m_S(s) = m_S(s')\}\\
IP(OP) &= \{\ op \in OP_L | \exists op' \in OP_L : op \neq op' \wedge m_{OP}(op) = m_{OP}(op')\}
\end{aligned}
$$

$$
\begin{aligned}
DP(P) = \{\ & p \in P_L \mid \exists t \in T_1 \backslash m_T(T_L) :\\
& pre_1(t) = \sum_{i=1}^{n} \lambda_i (r_i, p_i) \text{ with } \lambda_i \neq 0 \text{ and } m_P(p) = p_i\\
& \text{for some } i \text{ or}\\
& post_1(t) = \sum_{i=1}^{n} \lambda_i (r_i, p_i) \text{ with } \lambda_i \neq 0 \text{ and } m_P(p) = p_i\\
& \text{for some } i\}\\
DP(S) = \{\ & s \in S_L \mid \exists op \in OP_1 \backslash m_{OP}(OP_L) \text{ which contains } m_S(s) \text{ as one of}\\
& \text{the sorts in its signature}\}\\
\cup \{\ & s \in S_L \mid \exists p \in P_1 \backslash m_P(P_L) : type_1(p) = m_S(s)\}\\
DP(T_{OP}) = \{\ & r \in T_{OP_L}(X_L) \mid \exists t \in T_1 \backslash m_T(T_L) :\\
& pre_1(t) = \sum_{i=1}^{n} \lambda_i (r_i, p_i) \text{ with } \lambda_i \neq 0 \text{ and } m^{\#}_{SP}(r) = r_i\\
& \text{for some } i \text{ or}\\
& post_1(t) = \sum_{i=1}^{n} \lambda_i (r_i, p_i) \text{ with } \lambda_i \neq 0 \text{ and } m^{\#}_{SP}(r) = r_i\\
& \text{for some } i\}\\
DP(EQNS) = \{\ & e \in EQNS(\Sigma_L, X_L)\\
& \mid \exists t \in T_L \backslash m_T(T_L) : m^{\#}_{SP}(e) \in cond_1(t)\}
\end{aligned}
$$

Note that the dangling points of the additional variables X are implicitly defined by $DP(T_{OP})$

and $DP(EQNS)$.
The match m satisfies the gluing condition with respect to p if and only if

1. $IP(P) \cup DP(P) \subseteq GP(P)$
2. $IP(T) \subseteq GP(T)$
3. $DP(S) \cup IP(S) \subseteq GP(S)$
4. $IP(OP) \subseteq GP(OP)$
5. the set $E_C = (E_1 \setminus m^{\#}(E_L)) \cup m^{\#}_{SP}(l^{\#}_{SP}(E_K))$ is a set of equations over the signature (S_C, OP_C), where $S_C = (S_1 \setminus m_S(S_L)) \cup m_S(l_S(S_K))$
   $OP_C = (OP_1 \setminus m_{OP}(OP_L)) \cup m_{OP}(l_{OP}(OP_K))$
6. $DP(T_{OP}) \subseteq l^{\#}_{SP}(T_{OP_K}(X_K))$
7. $DP(EQNS) \subseteq l^{\#}_{SP}(EQNS(\Sigma_K, X_K))$
8. m is marking strict on places to be deleted, i.e. $\forall (a,p) \in (A_L \otimes (P_L \setminus l_P(P_K))) : M_L(a,p) = M_1(m_A(a), m_P(p))$