Sufficient Criteria for Applicability and Non-Applicability of Rule Sequences

Leen Lambers[1], **Hartmut Ehrig**[2], **Gabriele Taentzer**[3]

[1] leen@cs.tu-berlin.de, [2] ehrig@cs.tu-berlin.de
Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin, Germany
[3] taentzer@mathematik.uni-marburg.de
Fachbereich Mathematik und Informatik
Philipps-Universität Marburg, Germany

Abstract: In several rule-based applications using graph transformation as underlying modeling technique the following questions arise: How can one be sure that a specific sequence of rules is applicable (resp. not applicable) on a given graph? Of course, it is possible to use a trial and error strategy to find out the answer to these questions. In this paper however, we will formulate suitable sufficient criteria for applicability and other ones for non-applicability. These criteria can be checked in a static way i.e. without trying to apply the whole rule sequence explicitly. Moreover if a certain criterion is not satisfied, then this is an indication for reasons why rule sequences may or may not be applicable. Consequently it is easier to rephrase critical rule sequences. The results are formulated within the framework of double pushout (DPO) graph transformations with negative application conditions (NACs).

Keywords: graph transformation, analysis, applicability of rules

1 Introduction

When analyzing integrated specifications of rules with control structures [LEMP07, MMT06] for consistency, a considerable amount of rule sequences is to be checked for applicability (resp. non-applicability). Hence, it is not appropriate to check the applicability or non-applicability of each rule sequence explicitly. Therefore static analysis techniques are desirable. Statically means that it is not necessary to use a trial and error strategy and therefore backtracking can be avoided. In the following, we present sufficient criteria for the applicability and for the non-applicability of a rule sequence to a graph. These criteria can be checked in a static way, since they are based mainly on the dependency or independency of rules. Moreover the non-satisfaction of one of the criteria gives a hint to the reason for a rule sequence to be applicable or inapplicable. Applicability criteria have been studied also in [VL07] for simple digraphs using matrix graph grammars.

As example we consider a simple Mutual Exclusion Algorithm implemented by graph transformation rules with a start graph G and type graph presented in Fig. 1. The algorithm enables two processes to access a resource according to the mutual exclusion principle. The purpose of this algorithm is to control the usage of the resource such that at most one process holds the

resource at time (safety property). Furthermore, if a process demonstrates a request for the resource it should be served eventually (liveness property). In the following, we will check safety (resp. liveness) by checking if certain rule sequences are not applicable (resp. applicable) on the start graph.

The paper is structured as follows: Section 2 presents preliminaries. At first we repeat the main definitions for rule-based transformations with means of double pushout (DPO) graph transformations [CMR+97] with negative application conditions (NACs) [HHT96]. Then we define asymmetric dependency and independency on the level of transformations. Therefrom we can deduce the concept of asymmetric independency for rules. In Section 3 and 4 we define sufficient criteria for applicability and non-applicability of rule sequences. Section 5 takes up the example again and shows how to check the criteria presented in Section 3 and 4 with some support of the graph transformation tool AGG [Tae04].

2 Independency and Dependency of Rules

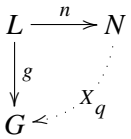
2.1 Graph Transformation with NACs

We repeat the basic definitions for double pushout graph transformation with negative application conditions (NACs). A graph rule holding a NAC n can be applied on a graph G only if the forbidden structure expressed by n is not present in G.

Definition 1 (graph, graph morphism, rule) A *graph* $G = (G_E, G_V, s, t)$ consists of a set G_E of edges, a set G_V of vertices and two mappings $s,t : G_E \to G_V$, assigning to each edge $e \in G_E$ a source $q = s(e) \in G_V$ and target $z = t(e) \in G_V$. A *graph morphism* (short *morphism*) $f : G_1 \to G_2$ between two graphs $G_i = (G_{i,E}, G_{i,V}, s_i, t_i)$, $(i = 1,2)$ is a pair $f = (f_E : G_{E,1} \to G_{E,2}, f_V : G_{V,1} \to G_{V,2})$ of mappings, such that $f_V \circ s_1 = s_2 \circ f_E$ and $f_V \circ t_1 = t_2 \circ f_E$. A morphism $f : G_1 \to G_2$ is injective (resp.surjective) if f_V and f_E are injective (resp. surjective) mappings. A graph transformation *rule* $p : L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$ consists of a rule name p and a pair of injective morphisms $l : K \to L$ and $r : K \to R$. The graphs L, K and R are called the left-hand side (LHS), the interface, and the right-hand side (RHS) of p, respectively.

Definition 2 (rule and transformation with NACs, applicability of rule with NACs)

- A *negative application condition* or $NAC(n)$ *on* p for a rule $p : L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$ (l,r injective) is an arbitrary morphism $n : L \to N$. A morphism $g : L \to G$ *satisfies* $NAC(n)$ on L, written $g \models NAC(n)$, if and only if $\nexists\, q : N \to G$ injective such that $q \circ n = g$.



A set of NACs on p is denoted by $NAC_p = \{NAC(n_i) | i \in I\}$. A morphism $g : L \to G$ *satisfies* NAC_p if and only if g satisfies all single NACs on p i.e. $g \models NAC(n_i)\ \forall i \in I$.

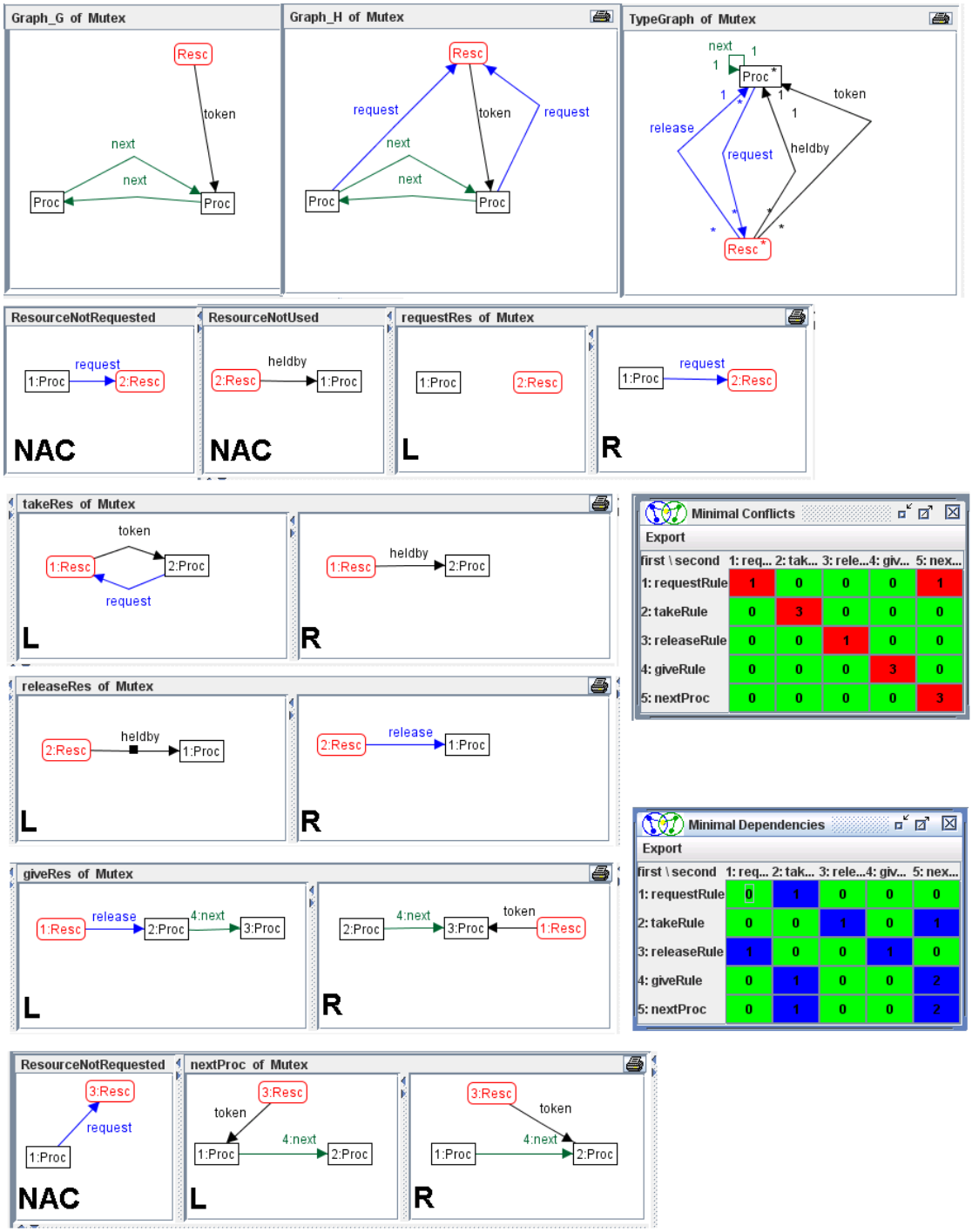


Figure 1: Start graph G, graph H and type graph of Mutual Exclusion - Rules *requestRes*,*takeRes*,*releaseRes*,*giveRes* and *nextProc* - Dependency and Conflict Matrix in AGG

- A *rule* (p, NAC_p) *with NACs* is a rule with a set of NACs on p.
- A *direct transformation* $G \overset{p,g}{\Rightarrow} H$ via a rule $p : L \leftarrow K \rightarrow R$ with NAC_p and a match $g : L \rightarrow G$ consists of the double pushout [CMR+97] (DPO)

$$\begin{array}{ccccc} L & \longleftarrow & K & \longrightarrow & R \\ {\scriptstyle g}\downarrow & & \downarrow & & \downarrow{\scriptstyle h} \\ G & \longleftarrow & D & \longrightarrow & H \end{array}$$

  where g satisfies NAC_p, written $g \models NAC_p$. Since pushouts along injective morphisms always exist, the DPO can be constructed if the pushout complement of $K \rightarrow L \rightarrow G$ exists. If so, we say that the match g satisfies the gluing condition of rule p. If there exists a morphism $g : L \rightarrow G$ which satisfies the gluing condition and $g \models NAC_p$ we say that rule p is *applicable* on G via the match g.

Example 1 *Consider the graph transformation rules of the Mutual Exclusion Algorithm presented in Fig. 1. Rule* requestRes *expresses a request of a process to access the resource if this process has not requested the resource yet or is using the resource already. Rule* takeRes *enables a process to start using the resource if this process possesses a token.* releaseRes *can release the resource again and* giveRes *passes the token to the other process after releasing. Finally rule* nextProc *can pass a token from one process to the other one as long as the first one has not expressed a request.*

In [LEOP07] it is proven that to each rule p with NAC_p there exists an inverse rule p^{-1} with $NAC_{p^{-1}}$ such that each transformation can be inverted. The following definition shows how to construct from NAC_p on rule p equivalent NACs $NAC_{p^{-1}}$ on the inverse rule p^{-1} [EEHP04]:

Definition 3 (construction of NACs on inverse rule) For each $NAC(n_i)$ with $n_i : L \rightarrow N_i$ on $p = (L \leftarrow K \rightarrow R)$, the equivalent NAC $R_p(NAC(n_i))$ on $p^{-1} = (R \leftarrow K \rightarrow L)$ is defined in the following way:

$$\begin{array}{ccccc} L & \longleftarrow & K & \longrightarrow & R \\ {\scriptstyle n_i}\downarrow & (1) & \downarrow & (2) & \downarrow{\scriptstyle n_i'} \\ N_i & \longleftarrow & Z & \longrightarrow & N_i' \end{array}$$

- If the pair $(K \rightarrow L, L \rightarrow N_i)$ has a pushout complement, we construct $(K \rightarrow Z, Z \rightarrow N_i)$ as the pushout complement (1). Then we construct pushout (2) with the morphism $n_i' : R \rightarrow N_i'$. Now we define $R_p(NAC(n_i)) = NAC(n_i')$.
- If the pair $(K \rightarrow L, L \rightarrow N_i)$ does not have a pushout complement, we define $R_p(NAC(n_i)) = true$.

For each set of NACs on p, $NAC_p = \cup_{i \in I} NAC(n_i)$ we define the following set of NACs on p^{-1}: $NAC_{p^{-1}} = R_p(NAC_p) = \cup_{i \in I'} R_p(NAC(n_i))$ with $i \in I'$ if and only if the pair $(K \rightarrow L, L \rightarrow N_i)$ has a pushout complement.

With means of this construction we can formulate the following fact [EEHP04, LEOP07]:

Fact 1 (inverse direct transformation with NACs) *For each direct transformation with NACs $G \Rightarrow H$ via a rule $p : L \leftarrow K \rightarrow R$ with NAC_p a set of NACs on p, $H \Rightarrow G$ is a direct transformation with NACs via the inverse rule p^{-1} with $NAC_{p^{-1}}$.*

Example 2 *Consider rule* requestRes. *The inverse rule of* requestRes *deletes the* request *edge between a process and the resource if there is no other* request *edge nor a* heldby *edge between this process and the resource.*

2.2 Independency and Dependency of Rules

The (non-)applicability of a rule sequence on a graph G is affected by dependencies and independencies between the rules in the rule sequence. In particular it is affected by asymmetric dependencies as explained in this section.

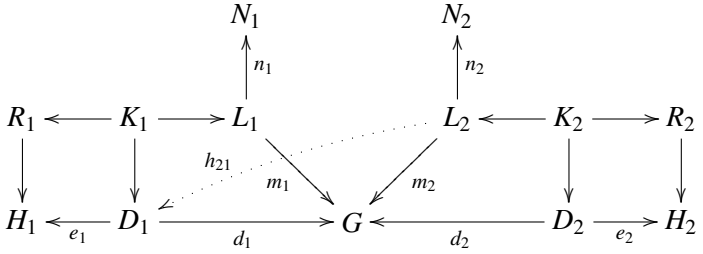
Two direct transformations are in conflict if they are not parallel independent. In [LEO06] a Conflict Characterization is given in which different reasons for conflicting transformations become clear. It is possible that one transformation deletes (resp. produces) a structure which is used (resp. forbidden) by the other one and the other way round. In particular we can deduce that two direct transformations are in conflict if *at least* one transformation depends on the other one. This asymmetric parallel dependency and on the contrary asymmetric parallel independency are expressed by the following definition.

Definition 4 (asymmetrically parallel dependent and independent transformations) A direct transformation $G \overset{(r_2,m_2)}{\Longrightarrow} H_2$ with NAC_{r_2} is *asymmetrically parallel dependent* on $G \overset{(r_1,m_1)}{\Longrightarrow} H_1$ with NAC_{r_1} if:

1. $\nexists h_{21} : L_2 \rightarrow D_1 : d_1 \circ h_{21} = m_2$ *(delete-use-conflict)*
   OR

2. there exists a unique $h_{21} : L_2 \rightarrow D_1 : d_1 \circ h_{21} = m_2$, but $e_1 \circ h_{21} \not\models NAC_{r_2}$ *(produce-forbid-conflict).*

A direct transformation $G \overset{(r_2,m_2)}{\Longrightarrow} H_2$ with NAC_{r_2} is *asymmetrically parallel independent* on $G \overset{(r_1,m_1)}{\Longrightarrow} H_1$ with NAC_{r_1} if $G \overset{(r_2,m_2)}{\Longrightarrow} H_2$ is not asymmetrically parallel dependent on $G \overset{(r_1,m_1)}{\Longrightarrow} H_1$. This means in particular that $\exists h_{21} : L_2 \rightarrow D_1 : d_1 \circ h_{21} = m_2$ such that $e_1 \circ h_{21} \models NAC_{r_2}$.



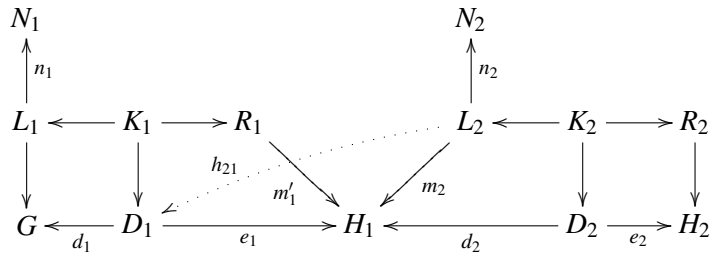
If a sequence of two direct transformations is not sequentially independent this is because either the second transformation depends on the first one or the other way round. The case in which the second transformation depends (resp. is independent) on the first one is described by asymmetric sequential dependency (resp. independency) as in the following definition. This case occurs in particular when the first transformation produces a structure which is used by the second one or the first transformation deletes a structure forbidden by the second one. In other words the first transformation triggers the second one.

Definition 5 (asymmetrically sequential dependent and independent transformations) A direct transformation $H_1 \overset{(r_2,m_2)}{\Longrightarrow} H_2$ with NAC_{r_2} is *asymmetrically sequential dependent* on $G \overset{(r_1,m_1)}{\Longrightarrow} H_1$ with NAC_{r_1} if

1. $\nexists h_{21} : L_2 \to D_1 : e_1 \circ h_{21} = m_2$ *(produce-use-dependency)*
   OR

2. there exists a unique $h_{21} : L_2 \to D_1 : e_1 \circ h_{21} = m_2$, but $d_1 \circ h_{21} \not\models NAC_{r_2}$ *(delete-forbid-dependency)*.

A direct transformation $H_1 \overset{(r_2,m_2)}{\Longrightarrow} H_2$ with NAC_{r_2} is *asymmetrically sequential independent* on $G \overset{(r_1,m_1)}{\Longrightarrow} H_1$ with NAC_{r_1} if $H_1 \overset{(r_2,m_2)}{\Longrightarrow} H_2$ with NAC_{r_2} is not asymmetrically sequential dependent on $G \overset{(r_1,m_1)}{\Longrightarrow} H_1$ with NAC_{r_1}. This means in particular that $\exists h_{21} : L_2 \to D_1 : e_1 \circ h_{21} = m_2$ such that $d_1 \circ h_{21} \models NAC_{r_2}$.



Now we can define asymmetric parallel (resp. sequential) independency for rules by demanding that the corresponding transformations are asymmetrically independent. Analogously it is possible to define asymmetric parallel (resp. sequential) dependency for rules.

Definition 6 (asymmetrically parallel independent rules) The rule r_2 is *asymmetrically parallel independent* on r_1 if every transformation $G \overset{(r_2,m_2)}{\Longrightarrow} H_2$ via r_2 with NAC_{r_2} is asymmetrically parallel independent on any other transformation $G \overset{(r_1,m_1)}{\Longrightarrow} H_1$ via r_1 with NAC_{r_1}.

Definition 7 (asymmetrically sequential independent rules) A rule r_2 is *asymmetrically sequential independent* on r_1 if every transformation $H_1 \overset{(r_2,m_2)}{\Longrightarrow} H_2$ via r_2 with NAC_{r_2} is asymmetrically sequential independent on any other transformation $G \overset{(r_1,m_1)}{\Longrightarrow} H_1$ via r_1 with NAC_{r_1} .

Example 3 *Rule* nextProc *is asymmetric parallel dependent on* requestRes, *since rule* requestRes *produces a* request *edge which is forbidden by rule* nextProc. *On the contrary, rule* requestRes *is asymmetric parallel independent on* nextProc, *since rule* nextProc *neither deletes anything what can be used by* requestRes *nor produces anything forbidden by* requestRes. *Rule* requestRes *is asymmetric sequentially dependent on rule* releaseRes, *since* releaseRes *deletes a* heldby *edge which is forbidden by* requestRes. *On the contrary, rule* releaseRes *is asymmetric sequentially independent on rule* requestRes, *since* requestRes *neither produces anything what can be used by* releaseRes *nor deletes anything forbidden by* requestRes.

For the criteria defined in the next section we need a special case of asymmetric sequential dependency for the case without NACs. It is possible namely that a rule r_1 produces everything which is needed by r_2 regardless of what is already present in the corresponding transformations.

Definition 8 (purely sequential dependent rules) A rule $r_2 : L_2 \leftarrow K_2 \rightarrow R_2$ is *purely sequential dependent* on $r_1 : L_1 \leftarrow K_1 \rightarrow R_1$ if r_2 is a rule without NACs and there exists an injective morphism $l_{21} : L_2 \rightarrow R_1$.

Remark 1 *r_2 is asymmetrically sequential dependent on r_1, if r_2 is purely sequential dependent on r_1 and the following mild assumptions are satisfied: a morphism $k_{21} : L_2 \rightarrow K_1$ does not exist such that $r_1 \circ k_{21} = l_{21}$, r_2 is non-deleting on nodes and $id : R_1 \rightarrow R_1 \models NAC_{r_1^{-1}}$.*

Example 4 *Rule* releaseRes *is purely sequential dependent on rule* takeRes, *because its LHS can be embedded completely into the RHS of* takeRes. *Rule* takeRes *is asymmetric sequentially, but not purely dependent on rule* requestRes, *since* requestRes *does not produce a* token *edge.*

We can analogously derive from the usual definition of sequential (resp. parallel) independence of transformations with NACs [LEO06] the definition for sequential (resp. parallel) independency on rules by demanding that all existing transformations via these rules are sequentially (resp. parallel) independent. Recall that for each pair of parallel independent transformations with NACs $H_1 \overset{r_1,m_1}{\Leftarrow} G \overset{r_2,m_2}{\Rightarrow} H_2$, there are an object G' and direct transformations $H_1 \overset{r_2,m_2'}{\Rightarrow} G'$ and $H_2 \overset{r_1,m_1'}{\Rightarrow} G'$ such that $G \overset{r_1,m_1}{\Rightarrow} H_1 \overset{r_2,m_2'}{\Rightarrow} G'$ and $G \overset{r_2,m_2}{\Rightarrow} H_2 \overset{r_1,m_1'}{\Rightarrow} G'$ are sequentially independent. The transformations can thus be performed in any order with the same result (Local Church-Rosser Theorem with NACs [LEO06]). Note that we have the following relationship between parallel and sequential independency: $G \overset{r_1}{\Rightarrow} H_1 \overset{r_2}{\Rightarrow} H_2$ are sequentially independent iff $G \overset{r_1^{-1}}{\Leftarrow} H_1 \overset{r_2}{\Rightarrow} H_2$ are parallel independent. In the next section we need sequential independency of a rule pair in order to be able to switch adjacent rules in a rule sequence.

Definition 9 (parallel and sequential independent rules) Rules r_1 and r_2 are *parallel independent* if each pair of transformations $G \overset{(r_1,m_1)}{\Longrightarrow} H_1$ via r_1 with NAC_{r_1} and $G \overset{(r_2,m_2)}{\Longrightarrow} H_2$ via r_2 with NAC_{r_2} is parallel independent. The pair of rules (r_1, r_2) is *sequentially independent* if each sequence of transformations $G \overset{(r_1,m_1)}{\Longrightarrow} H_1$ via r_1 with NAC_{r_1} and $H_1 \overset{(r_2,m_2)}{\Longrightarrow} G'$ via r_2 with NAC_{r_2} is sequentially independent.

Remark 2 *Note that the following correspondences between asymmetric parallel (resp. sequential) independency and parallel (resp. sequential) independency exists. Rules r_1 and r_2 are parallel independent if and only if r_1 is asymmetrically parallel independent of r_2 and r_2 is asymmetrically parallel independent of r_1. The rule pair (r_1, r_2) is sequentially independent if and only if r_2 is asymmetrically sequential independent on r_1 and r_1^{-1} is asymmetrically sequential independent on r_2^{-1}.*

Example 5 *The rule pair (*requestRes,nextProc*) is sequentially independent. Note that the NAC of rule* nextProc *forbids a process to shift the token if this process expressed a request. Thus whenever (*requestRes,nextProc*) can be applied in this order the request is expressed by the process to which the token is shifted. This is equivalent to first shifting the token to this process which then expresses a request.*

3 Applicability of Rule Sequences

3.1 Applicability Criteria

Let $s : r_1 r_2 \dots r_n$ be a sequence of n rules and G_0 a graph on which this sequence should be applied. The criteria defined in the following definition guarantee this applicability. The initialization criterion is trivial, since it just requires the first rule being applicable to graph G_0. The no node-deleting rules criterion avoids dangling edges. The third criterion ensures that the applicability of a rule r_i is not impeded by one of the predecessor rules r_j of r_i. Criterion 4a will be satisfied if rule r_i is purely sequential dependent from a rule r_j occurring somewhere before r_i in the sequence s. In this case r_j triggers the applicability of r_i regardless of what is present already in the start graph G_0. As soon as the sequential dependencies are not pure though this criterion is not satisfiable. Therefore we have also a more general criterion 4b. It ensures the applicability of a rule r_i needing some subgraph of a direct predecessor rule together with parts of the start graph G_0. This is expressed by the fact that a concurrent rule r_c of r_{i-1} and r_i exists which is applicable on the start graph G_0. The construction of a concurrent rule with NACs is explained in [LEOP07]. The correctness of the criteria is described in Theorem 1 which is proven in [LET08].

Definition 10 (applicability criteria) Given a sequence $s : r_1 r_2 \dots r_n$ of n rules and a graph G_0. Then we define the following applicability criteria for s on G_0:

1. r_1 is applicable on G_0 via the injective match $m_1 : L_1 \to G_0$ *(initialization)*
2. each rule occurring in s is non-deleting on nodes *(no node-deleting rules)*
3. $\forall r_i, r_j$ in s with $1 \leq j < i \leq n$, r_i is asymmetrically parallel independent on r_j *(no impeding predecessors)*
4. $\forall r_i$ in s with $1 < i \leq n$ which are not applicable on G_0 via an injective match
   (a) there exists a rule r_j in s which is applicable via an injective match on G_0 with $1 \leq j < i \leq n$ and r_i is purely sequential dependent on r_j *(pure enabling predecessor)*,

which especially means that r_i has no NACs
OR

(b) there exists a concurrent rule r_c of r_{i-1} and r_i such that r_c is applicable via an injective match on G_0 and r_c is asymmetrically parallel independent on r_j for all $j < (i-1)$ and r_j is asymmetrically parallel independent on r_c for all $i < j \leq n$. *(direct enabling predecessor)*

Theorem 1 (correctness of applicability criteria) *Given $s : r_1 r_2 \dots r_n$ a sequence of n rules and a graph G_0. If the criteria in Def. 10 are satisfied for rule sequence s and graph G_0, then this rule sequence is applicable on G_0 with injective matching i.e. there exists a graph transformation $G_0 \overset{r_1}{\Rightarrow} G_1 \dots G_{n-1} \overset{r_n}{\Rightarrow} G_n$ with injective matching.*

3.2 Applicability of Summarized Rule Sequences

The 4th criterion in Def. 10 will be satisfied only, if the rule r_i is purely sequential dependent from one single rule r_j occurring before r_i in the sequence or if it is asymmetrically sequential dependent on the rule r_{i-1}. In some transformations though a rule needs not only a subgraph from one single predecessor, but from several ones. For these cases the criteria could then be satisfied by a sequence in which exactly these rules are summarized to a concurrent rule. Note that the correctness of the following theorem is proven in [LET08].

Theorem 2 (summarized rule sequences) *Given $s : r_1 r_2 \dots r_n$ a sequence of n rules and a graph G_0. If the criteria in Def. 10 are satisfied for a rule sequence $s' : r'_1 r'_2 \dots r'_m$ with $m < n$ in which neighbored rules in s can be summarized by a concurrent rule, then the original rule sequence s is applicable on G_0 with injective matching i.e. there exists a graph transformation $G_0 \overset{r_1}{\Rightarrow} G_1 \dots G_{n-1} \overset{r_n}{\Rightarrow} G_n$ with injective matching.*

3.3 Applicability of Shift-Equivalent Rule Sequences

If it is not possible to satisfy criterion 4b in Def. 10 for a rule sequence $s : r_1 r_2 \dots r_n$, then it is still possible to exploit shift-equivalence. This is because criterion 4b could be satisfiable for a rule sequence s' in which a normal predecessor is shifted to be a direct one. Note that a rule r_j in s can be switched with a rule r_{j+1} only if the pairs of rules (r_j, r_{j+1}) and (r_{j+1}, r_j) is sequentially independent. If the criteria then hold though for rule sequence s' in which some rules have been shifted, they hold also for the original rule sequence s.

Definition 11 (shift-equivalent rule sequences) A rule sequence s' is *shift-equivalent* with a rule sequence $s : r_1 r_2 \dots r_m$ if s' can be obtained by switching rules r_j with r_{j+1} and the switching is allowed only if (r_j, r_{j+1}) and (r_{j+1}, r_j) are sequentially independent according to Def. 9.

Theorem 3 (checking shift-equivalent rule sequences) *If the criteria in Def. 10 are satisfied for a rule sequence $s : r_1 r_2 \dots r_n$ and a graph G_0, then all shift-equivalent rule sequences as defined in Def. 11 are applicable on G_0 with injective matching as well with the same result.*

Proof. This follows directly from Def. 11, the Local Church-Rosser Theorem with NACs [LEO06],

Def. 9 and Theorem 1. □

Remark 3 *Note that a somewhat weaker version of this theorem holds as well. Namely, suppose that we want to prove applicability of a rule sequence $s : r_1 r_2 \dots r_n$ to a graph G_0. Then it is sufficient to show the satisfaction of the criteria in Def. 10 for a rule sequence s' which can be deduced from s by switching forward rule r_{i+1} with r_i only if rule pair (r_{i+1}, r_i) is sequentially independent.*

4 Non-Applicability of Rule Sequences

Let $s : r_1 r_2 \dots r_n$ be a sequence of n rules and G_0 a graph. The satisfaction of the following criteria for s and G_0 guarantee that the sequence s will not be applicable on G_0. Criterion 1 is trivial, since it just requires the first rule being non-applicable to graph G_0. Criterion 2 checks if predecessors for a rule r_i which is not applicable already on G_0 are present in the sequence such that they can trigger the applicability of r_i. If not, r_i will not be applicable and therefore neither the rule sequence will be applicable. Note that the correctness of the following criteria is proven in [LET08].

Definition 12 (non-applicability criteria) Given $s : r_1 r_2 \dots r_n$ a sequence of n rules and a graph G_0. Then we define the following non-applicability criteria for s on G_0:

1. r_1 is not applicable on G_0 *(initialization error)*
   OR

2. $\exists r_i$ in s with $1 < i \leq n$ such that r_i is not applicable on G_0 but for all rules r_j in s with $1 \leq j < i \leq n$, r_i is asymmetrically sequential independent on r_j *no enabling predecessor.*

Theorem 4 (correctness of non-applicability criteria) *Given a sequence $s : r_1 r_2 \dots r_n$ of n rules and a graph G_0. If the criteria in Def. 12 are satisfied for rule sequence s and graph G_0, then this rule sequence is not applicable on G_0 i.e. there exists no graph transformation $G_0 \overset{r_1}{\Rightarrow} G_1 \dots G_{n-1} \overset{r_n}{\Rightarrow} G_n$.*

Analogously to Theorem 3 we can formulate the following theorem expressing that shift-equivalent-rule sequences are not applicable to a graph G_0 if one of the sequences satisfies the non-applicability criteria.

Theorem 5 (checking shift-equivalent rule sequences) *If the criteria in Def. 12 are satisfied for a rule sequence $s : r_1 r_2 \dots r_n$ and a graph G_0, then all shift-equivalent rule sequences as defined in Def. 11 are not applicable on G_0 either.*

Proof. This follows directly from Def. 11, the Local Church-Rosser Theorem with NACs [LEO06], Def. 9 and Theorem 4. □

Remark 4 *Note that a somewhat weaker version of this theorem holds as well. Namely, suppose that we want to prove non-applicability of a rule sequence $s : r_1 r_2 \dots r_n$ to a graph G_0. Then it is*

sufficient to show the satisfaction of the criteria in Def. 12 for a rule sequence s' which can be deduced from s by switching forward rule r_{i+1} with r_i only if rule pair (r_i, r_{i+1}) is sequentially independent.

5 Checking the Criteria

5.1 Checking for Sequential and Parallel Dependency of Rules in AGG

To check asymmetric sequential and parallel dependency of rules, we compute all corresponding critical pairs by AGG [Tae04]. A critical pair represents the parallel (resp. sequential) dependency of rules in a minimal context. The minimal conflicts (i.e. asymmetric parallel dependencies) are represented in a conflict matrix. The minimal dependencies (i.e. asymmetric sequential dependencies) are represented in a dependency matrix. The entry numbers within these matrices indicate how many minimal conflicts and dependencies, resp. have been found. For the example they are shown in Fig. 1.More precisely, entry (r_j, r_i) (row, column) in the conflict matrix in AGG describes all $G \overset{(r_i,m_i)}{\Rightarrow} H_i$ which are asymmetrically parallel dependent on $G \overset{(r_j,m_j)}{\Rightarrow} H_j$ in a minimal context. Entry (r_j, r_i) in the dependency matrix in AGG describes all $G \overset{(r_j,m_j)}{\Rightarrow} H_j \overset{(r_i,m_i)}{\Rightarrow} G'$ such that the second transformation is asymmetrically sequential dependent on the first one in a minimal context. Note that it is not possible yet to check with AGG if a pair of rules (r_j, r_i) is sequentially independent as defined in Def.9. It is part of current work though to enrich the dependency matrix with more information and thus enable this possibility.

5.2 Checking Applicability Criteria on Mutual Exclusion

We check applicability (i.e. liveness) of the following rule sequences:

requestRes, takeRes, releaseRes should be applicable to the right process in the graph G shown in Figure 1. We thus check for this rule sequence and graph G that the applicability criteria in Def. 10 are satisfied.

1. Rule *requestRes* is applicable to G.
2. Rule *takeRes* is asymmetrically parallel independent of rule *requestRes*. Furthermore, rule *releaseRes* is asymmetrically parallel independent of *requestRes* as well as *takeRes*.
3. Rules *takeRes, releaseRes, giveRes* are not applicable to G. Thus, we have to show that their application is enabled by the rule applications performed before. Rule *takeRes* is asymmetrically sequential dependent on *requestRes*, and the concurrent rule of *takeRes* and *requestRes* is applicable on G. It expresses how a resource can be requested and taken in one step. *requestRes* is thus a direct enabling predecessor for *takeRes*. Moreover rule *releaseRes* is purely sequential dependent on rule *takeRes* and therefore *takeRes* is a pure enabling predecessor for *releaseRes*.

requestRes, takeRes, releaseRes, giveRes is a slightly longer sequence and should still be applicable to G. It is not possible though to fulfill in particular criterion 4 in Def. 10 for this

sequence. However, it is still possible to fulfill the applicability criteria for a summarized sequence as proven in Theorem 2. In this case, it is possible to satisfy the criteria for the summarized sequence only. It consists of one concurrent rule *requestTakeReleaseGiveRes* which is equal to the rule *nextProc*. *nextProc* is applicable on G and therefore the original sequence is applicable on G as well. Thus sometimes an applicable rule sequence as given in this example might be hard to detect. This is because *giveRes* needs some subgraph of the start graph G which is not needed by the other rules and in addition all rules (except for the first one) are asymmetrically sequential dependent of the previous rule.

5.3 Checking Non-Applicability Criteria on Mutual Exclusion

We check non-applicability (i.e. safety) of the following rule sequences:

requestRes, requestRes, nextProc specifies a request of both processes and then a token transfer. This sequence should not be applicable on the start graph G in Fig. 1, since before transferring a token one of the requests should be processed. We check the non-applicability criteria in Def. 12 for this rule sequence on G.

1. *requestRes* is applicable on the start graph G.
2. *nextProc* is not applicable on the start graph G and is sequentially independent on *requestRes*. Therefore the second criterion is fulfilled and there is no enabling predecessor.

requestRes, giveRes specifies a request and then a token shift because the resource has been released. This sequence should not be applicable to the start graph G in Fig. 1, since the resource is still unused. It is easy to verify that the criteria in Def. 12 are fulfilled. Moreover rule pair *(requestRes,giveRes)* and *(giveRes,requestRes)* are sequentially independent. Therefore due to Theorem 5 we can conclude directly that sequence *giveRes,requestRes* is not applicable either on G.

takeRes, takeRes specifies a take of the resource by both processes simultaneously. This sequence should not be applicable on the graph H in Figure 1 in which both processes request the resource. Thus we check the non-applicability criteria in Def. 12 for this sequence and graph H.

1. The first rule *takeRes* is applicable to graph H on the right process.
2. The second rule is again *takeRes* and we just mentioned that it is applicable on the right process in H. We want to check though for safety reasons that it is impossible to apply the second *takeRes* on the other (i.e. *left*) process simultaneously. A matching of *takeRes* on the left process into H does not exist. For this kind of matching criterion 2 holds since in addition rule *takeRes* is asymmetrically sequentially independent from itself. This means in particular that the application of the first *takeRes* on the right process does not enable the application of the second *takeRes* on the left process and hence such a rule application on H is not possible.

This example demonstrates that sometimes it is necessary to include information about the matches for the rule sequence to be checked for safety. Note moreover that if we would have taken a longer sequence consisting of *requestRes, requestRes, takeRes* and *takeRes*, we could not have checked the non-applicability of that sequence on the start graph G by Def. 12, since rule *takeRes* is asymmetrically sequentially dependent on rule *requestRes*. Thus it is crucial in this case to select the kernel sequence where the problem is conjectured and in particular this shows us that our criteria are sufficient but not necessary.

6 Conclusion and Future Work

In this paper sufficient criteria are formulated for the applicability and non-applicability of rule sequences. These criteria can be checked in a static way, i.e. without applying the rule sequences. Future work is concerned with formulating criteria for rules with attributes, further optimization of the criteria, efficiently checking the satisfaction of the criteria and implementing applicability checks in AGG. Furthermore, we like to evaluate the criteria in larger case studies such as [LEMP07, MMT06].

Bibliography

[CMR+97] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, M. Löwe. Algebraic Approaches to Graph Transformation I: Basic Concepts and Double Pushout Approach. In Rozenberg (ed.), *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*. Chapter 3, pp. 163–245. World Scientific, 1997.

[EEHP04] H. Ehrig, K. Ehrig, A. Habel, K.-H. Pennemann. Constraints and Application Conditions: From Graphs to High-Level Structures. In Parisi-Presicce et al. (eds.), *Proc. 2nd Int. Conference on Graph Transformation (ICGT'04)*. LNCS 3256, pp. 287–303. Springer, Rome, Italy, October 2004.

[HHT96] A. Habel, R. Heckel, G. Taentzer. Graph Grammars with Negative Application Conditions. *Special issue of Fundamenta Informaticae* 26(3,4):287–313, 1996.

[LEMP07] L. Lambers, H. Ehrig, L. Mariani, M. Pezzè. Iterative Model-driven Development of Adaptable Service-Based Applications. In *proceedings of the International Conference on Automated Software Engineering*. 2007.

[LEO06] L. Lambers, H. Ehrig, F. Orejas. Conflict Detection for Graph Transformation with Negative Application Conditions. In *Proc. Third International Conference on Graph Transformation (ICGT'06)*. LNCS 4178, pp. 61–76. Springer, Natal, Brazil, September 2006.

[LEOP07] L. Lambers, H. Ehrig, F. Orejas, U. Prange. Parallelism and Concurrency in Adhesive High-Level Replacement Systems with Negative Application Conditions. In Ehrig et al. (eds.), *Workshop on Applied and Computational Category Theory (ACCAT'07)*. Elsevier Science, 2007.

[LET08] L. Lambers, H. Ehrig, G. Taentzer. Sufficient Criteria for Applicability and Non-Applicability of Rule Sequences. Technical report, TU Berlin, 2008.

[MMT06] K. Mehner, M. Monga, G. Taentzer. Interaction Analysis in Aspect-Oriented Models. In *Proc. 14th IEEE International Requirements Engineering Conference*. Pp. 66–75. IEEE Computer Society, Minneapolis, Minnesota, USA, September 2006.

[Tae04] G. Taentzer. AGG: A Graph Transformation Environment for Modeling and Validation of Software. In Pfaltz et al. (eds.), *Application of Graph Transformations with Industrial Relevance (AGTIVE'03)*. Pp. 446 – 456. LNCS 3062, Springer, 2004. AGG-Homepage:http://tfs.cs.tu-berlin.de/agg

[VL07] P. Velasco, J. de Lara. Using Matrix Graph Grammars for the Analysis of Behavioural Specifications: Sequential and Parallel Independence. In *Jornadas de Programacion y Lenguajes (PROLE2007)*. 2007.