Electronic Communications of the EASST
Volume 16 (2009)

Proceedings of the
Doctoral Symposium at the
International Conference on Graph Transformation
(ICGT 2008)

Permutation Equivalence of DPO Derivations with Negative Application Conditions based on Subobject Transformation Systems

Frank Hermann

16 pages

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

Permutation Equivalence of DPO Derivations with Negative Application Conditions based on Subobject Transformation Systems

Frank Hermann[1]

[1] frank(at)cs.tu-berlin.de
Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany

Abstract: Switch equivalence for transformation systems has been successfully used in many domains for the analysis of concurrent behaviour. When using graph transformation as modelling framework for these systems, the concept of negative application conditions (NACs) is widely used – in particular for the specification of operational semantics. In this paper we show that switch equivalence can be improved essentially for the analysis of systems with NACs by our new concept of permutation equivalence.

Two derivations respecting all NACs are called permutation-equivalent, if they are switch-equivalent disregarding the NACs. In fact, there are permutation-equivalent derivations which are not switch-equivalent with NACs.

As main result of the paper, we solve the following problem: Given a derivation with NACs, we can efficiently derive all permutation-equivalent derivations to the given one by static analysis. The results are based on extended techniques for subobject transformation systems, which have been introduced recently.

Keywords: Graph Transformation, Adhesive Categories, Subobject Transformation Systems, Negative Application Conditions, Process Analysis

1 Introduction

Transformation systems based on the double pushout (DPO) approach [CMR$^+$97] with negative application conditions (NACs) [HHT96, EEPT06] are a suitable modelling framework for several application domains, e.g. definition of operational semantics and simulation. In this context, the analysis of concurrent behaviour of an execution of the system is of interest. A process of an execution describes all possible equivalent executions. Correspondingly, a process of a derivation defines an equivalence class of derivations. Processes of graph transformation systems based on the DPO approach [CMR96] were defined as occurrence grammars in [Bal00]. Occurrence grammars were lifted to the abstract setting of adhesive rewriting systems [BCH$^+$06] in order to generalise the process construction. This opened possibilities for analysing processes of transformation systems based on arbitrary adhesive categories [LS04], such as typed graphs, graphs with scopes and graphs with second order edges.

This paper extends the standard switch equivalence of derivations without and with negative application conditions to the so-called *permutation equivalence* of derivations with negative application conditions (NACs) in adhesive categories. The main difference is that there are permutation-equivalent derivations with NACs, which cannot be derived by switching NAC-independent neighbouring derivation steps. The challenge is to efficiently calculate derivations, which are equivalent in the sense that all NACs are respected and the matches of the original derivation are preserved. However, a direct construction of all permutation-equivalent derivations is complex in general. First of all, the amount of possible permutations is high in general and furthermore, the permutations have to be derived from the original derivation by computing the new matches and the new intermediate objects from the old ones and checking that all NACs are fulfilled.

The main result of this paper is a framework for the efficient analysis of permutation equivalence, i.e. the efficient construction of all derivations, which are permutation-equivalent to a given one (see Theorems 1 - 3 in Sec. 5). The presented technique is based on subobject transformation systems (STSs) [CHS08], which can be constructed in advance, i.e. possibly before a user requests an analysis. They are based on the process construction given in [BCH$^+$06] and we extend them for the case with NACs. This builds the basis for efficient dependency checks between components of pairs of rule occurrences, where expensive pattern matching is avoided.

The next section reviews transformation systems and introduces the new notion of permutation equivalence. Thereafter, subobject transformation systems (STSs) as process model of a derivation are reviewed and Section 4 shows how the process construction can be extended to systems with NACs. Thereafter, Section 5 presents the analysis of permutation equivalence of derivations with NACs and shows as a main result that the analysis can be transferred to the derived STS leading to correct results for the original system. Section 6 concludes the main results and discusses future work within the presented framework.

2 Transformation Systems and Permutation Equivalence

In this section we review transformation systems based on the double pushout (DPO) approach and the standard switch equivalence of derivations. We present an example in the context of workflow modelling, where we use negative application conditions (NACs) and show that switch equivalence does not lead to all intuitively equivalent derivations in our example. For this reason we introduce the new notion of permutation equivalence, which leads to the discussed equivalent derivations.

A derivation in an adhesive category $\mathbf{C}$ is given by a sequence of rule applications within a grammar. A transformation rule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ consists of three objects $L, K, R \in Obj(\mathbf{C})$ being left-hand side, interface, and a right-hand side, respectively, and two monomorphisms $l, r \in Mor(\mathbf{C})$. The interface K contains the part which is not changed by the rule and hence occurs in both L and R. Applying a rule p to an object G means to find a monomorphism $m : L \to G$ and to replace the matched part in G by the corresponding right-hand side R of the rule, thus leading to a transformation step $G \overset{p,m}{\Longrightarrow} H$.

$$\begin{tikzcd}
NAC \arrow[dr, "q"', "/" marking] & L \arrow[l, "n"'] \arrow[d, "m"'] \arrow[d, phantom, shift left=4ex, "(PO_1)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] \arrow[d, phantom, shift left=4ex, "(PO_2)"] & R \arrow[d, "m^*"] \\
 & G & D \arrow[l] \arrow[r] & H
\end{tikzcd}$$

In this paper, matches are required to be monomorphisms. But as explained in [HE08, Her09b], the analysis based on subobject transformation systems is also possible for systems with non-monomorphic matching. A transformation step is given by a double-pushout (DPO), where D is the intermediate object after constructing the pushout complement for pushout (PO_1) and in (PO_2) H is constructed as gluing of D and R

via K. A sequence of transformation steps $d = (d_1;\dots;d_n)$ is called a derivation. A rule may contain a set of negative application conditions (NACs) [HHT96, EEPT06]. A NAC $(N, n : L \to N)$ of a rule consists of a negative pattern N together with a monomorphism n from the left hand side of the rule to N. Intuitively, it forbids the presence of a certain pattern in an object G to which the rule shall be applied. A match $L \xrightarrow{m} G$ satisfies a NAC $n : L \to N$, written $m \models N$, if there is no monomorphism $N \xrightarrow{q} G$ with $q \circ n = m$.

Typed transformations are based on a type object TG and the derived slice category $\mathbf{C} \downarrow TG$, where each object G is typed over TG by $type_G : G \to TG$ and morphisms are compatible with the typing morphisms. A grammar specifies a start object, a type graph and a set of rules for performing typed transformations.

Definition 1 (Grammar) Given a category $\mathbf{C}$, a grammar $GG = (SG, TG, P, \pi)$ consists of a type object TG, a start object SG, a finite set of rule names P and a function π, which maps a rule name to a rule with NACs $p = (\underline{p}, N)$ containing a rule $\underline{p} = (L \xleftarrow{l} K \xrightarrow{r} R)$ and a finite set of negative application conditions N. The start graph and the productions of GG are typed over TG.

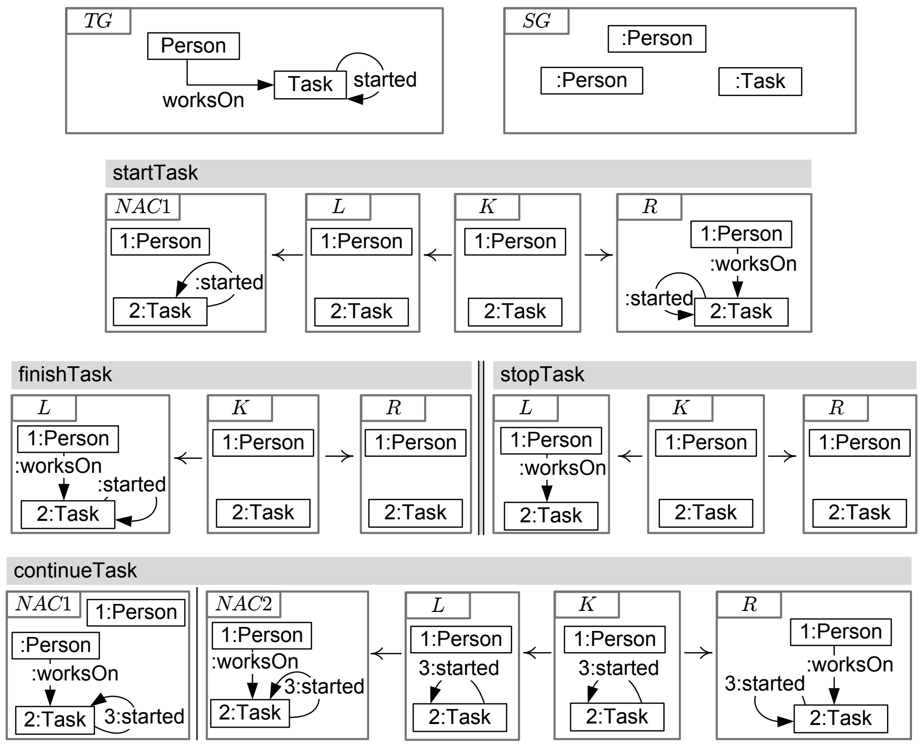


Figure 1: The graph grammar GG

Example 1 (Graph Grammar GG) *Figure 1 shows the graph grammar $GG = (SG, TG, P, \pi)$ for mobile agents in reconfigurable networks. The mappings of the rule morphisms are specified by numbers. Rule "startTask" assigns a person to a task via an edge of the type "worksOn", but the rule is not applicable if the task was already started, as specified by the NAC "NAC1". Rule "finishTask" is inverse to "startTask" and removes the assignment and the edge of the type* *"started", while rule "stopTask" also deletes the assignment, but not the flag "started". Finally, rule "continueTask" specifies that a person may continue the work, which possibly was started by another person and stopped meanwhile. The NACs* NAC1 *and* NAC2 *of this rule require that neither the person itself nor another person is working on the task to be assigned.*

Switch equivalence of derivations without NACs is based on sequential independence.

Definition 2 (Sequential Independence without NACs) Let $d = (G_0 \xRightarrow{p_1,m_1} G_1 \xRightarrow{p_2,m_2} G_2)$ be a derivation without NACs in a grammar GG. Then, $d_1 = G_0 \xRightarrow{p_1,m_1} G_1$ and $d_2 = G_1 \xRightarrow{p_2,m_2} G_2$ are two sequentially independent derivation steps, if there exist $i : R_1 \to D_2, j : L_2 \to D_1$ in the diagram on the right, which shows parts of the derivation diagrams, s.t. $l_2' \circ i = m_1'$ and $r_1' \circ j = m_2$.

$$\begin{tikzcd}
K_1 \arrow[r] \arrow[d] & R_1 \arrow[drr, dotted, "i"] \arrow[d, "m_1'"'] & L_2 \arrow[dll, dotted, "j"'] \arrow[d, "m_2"] & K_2 \arrow[l] \arrow[d] \\
D_1 \arrow[r, "r_1'"] & G_1 & & D_2 \arrow[ll, "l_2'"']
\end{tikzcd}$$

Remark 1 (Local Church Rosser) *Two sequentially independent derivation steps without NACs can be switched by the Local Church Rosser Theorem (Thm. 5.12 in [EEPT06]). By $\overline{m_k} : L_k \to G_k'$ we denote the match for rule p_k in the new order of the steps.*

Given a derivation without NACs, switch equivalence leads to the complete set of its equivalent derivations [BCH$^+$06]. Note that DPO derivation diagrams are unique up to isomorphism only, thus we relate isomorphic derivation diagrams by "$\cong$" meaning that there are isomorphisms between the objects compatible with the involved morphisms.

Definition 3 (Switch Equivalence without NACs) Let $d = (d_1; \ldots; d_k; d_{k+1}; \ldots d_n)$ be a derivation without NACs in grammar GG, where $d_k; d_{k+1}$ are two sequentially independent derivation steps. Let d' be derived from d by switching $(d_k; d_{k+1})$ to $(d_{k+1}'; d_k')$ according to the Local Church Rosser Theorem. Then, d' is a switching of d, written $d \overset{sw}{\sim} d'$. Switch-equivalence $\overset{sw}{\approx}$ is the union of the transitive closure of $\overset{sw}{\sim}$ and the relation $\cong$ for isomorphic derivations.

We now extend the notion of switch equivalence to derivations with NACs using sequential independence for derivations with NACs according to [HHT96, LEO06].

Definition 4 (Sequential Independence with NACs) Let $d = (G_0 \xRightarrow{p_1,m_1} G_1 \xRightarrow{p_2,m_2} G_2)$ be a derivation with NACs in a grammar GG. Suppose that $i : R_1 \to D_2, j : L_2 \to D_1$ exist in the two derivation diagrams on the right, s.t. $l_2' \circ i = m_1', r_1' \circ j = m_2$. Suppose also that the derived match $\overline{m}_1 : L_1 \to G_1'$ by the Local Church Rosser Theorem and the match $\overline{m}_2 = l_1' \circ j : L_2 \to G_0$ fullfill all NACs, i.e. $\overline{m}_2 \models N_2$ for each NAC $(n_2 : L_2 \to N_2)$ of p_2 and $\overline{m}_1 \models N_1$ for each NAC $(n_1 : L_1 \to N_1)$ of p_1. Then, $G_0 \xRightarrow{p_1,m_1} G_1, G_1 \xRightarrow{p_2,m_2} G_2$ are two sequentially independent derivation steps with NACs.

$$\begin{tikzcd}
L_1 \arrow[d, "m_1"'] & K_1 \arrow[l] \arrow[r] \arrow[d] & R_1 \arrow[drr, dotted, "i"] \arrow[d, "m_1'"'] & L_2 \arrow[dll, dotted, "j"'] \arrow[d, "m_2"] & K_2 \arrow[l] \arrow[r] \arrow[d] & R_2 \arrow[d, "m_2'"] \\
G_0 & D_1 \arrow[l, "l_1'"] \arrow[r, "r_1'"'] & G_1 & & D_2 \arrow[ll, "l_2'"] \arrow[r, "r_2'"'] & G_2
\end{tikzcd}$$

Definition 5 (Switch Equivalence with NACs) Let $d = (d_1; \ldots; d_k; d_{k+1}; \ldots d_n)$ be a derivation with NACs in grammar GG, where $d_k; d_{k+1}$ are two sequentially independent derivation steps with NACs. Let d' be derived from d by switching $d_k; d_{k+1}$ to $d_{k+1}'; d_k'$ according to the Local

Church Rosser Theorem. Then, d' is a switching with NACs of d, written $d \overset{swN}{\sim} d'$. Switch equivalence with NACs $\overset{swN}{\approx}$ is the union of the transitive closure of $\overset{swN}{\sim}$ and the relation $\cong$ for isomorphic derivations.

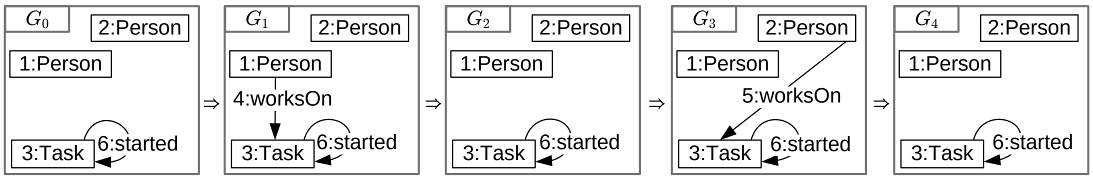


Figure 2: Derivation d in the grammar GG

Example 2 (Derivation in GG) *Derivation* $d = (d_1;d_2;d_3;d_4) = (G_0 \xRightarrow{continueTask,m_1} G_1 \xRightarrow{stopTask,m_2} G_2 \xRightarrow{continueTask,m_3} G_3 \xRightarrow{stopTask,m_4} G_4)$ *in Fig.* 2 *describes that at first person "1" works on task "3" and afterwards person "2" works on the same task, but both of them stop without finishing the task. Derivation steps* d_2 *to* d_4 *are dependent from their preceding steps and thus, no switching of independent steps is possible: the second step deletes edge "4" produced by the first step, the NAC of the third step forbids the presence of "4", which is deleted by the second, and finally, the fourth step deletes edge "5" that was created by the third step. However, there is a permutation of the steps, which is conceptually equivalent to the depicted derivation. Consider* $d' = (d'_3;d'_4;d'_1;d'_2)$*, where the third and fourth steps are moved to the front, all NACs are respected and the rules of GG are applied at the same places of the graph that is transformed.*

Since switch equivalence is not general enough for transformation systems with NACs as shown and explained in Example 2, we introduce the notion of permutation equivalence for derivations with NACs, which relates all equivalent derivations that respect the NACs. In particular, the equivalent permutation of the example can be derived by the new notion.

Definition 6 (Permutation Equivalence of Derivations) Two derivations d and d' with NACs in a grammar GG are permutation-equivalent, written $d \overset{\pi}{\approx} d'$, if disregarding the NACs, they are switch-equivalent.

A direct analysis of permutation equivalence in the adhesive category of the derivations is possible, but can be quite complex, if e.g. the graphs of the derivation are much bigger than in the simple running example. For each possible switching disregarding the NACs we have to update the derivation diagrams and perform pattern matching for the NACs on the updated objects in order to check whether the new derivation is permutation-equivalent. Therefore, we suggest to first construct a process model based on Subobject Transformation Systems, and then to perform a more efficient analysis on it. Accordingly, the main results of this paper, given by Theorems 1 - 3 in Sec. 5, show that the analysis on the basis of STSs is sound and complete and that the process model can be constructed efficiently.

Next, we will show how subobject transformation systems can be used as process model for a derivation without NACs. Thereafter, we present in Sec. 4 how they can be extended to the case with NACs leading to an efficient analysis framework for permutation equivalence, which is shown and explained in Sec. 5.

3 Subobject Transformation Systems

Processes of a graph grammar are defined as occurrence grammars together with a mapping to the original grammar. The concept of occurrence grammars was extended to the more abstract setting of adhesive rewriting systems [BCH$^+$06] based on subobjects. This technique was further elaborated in [CHS08] introducing the general concept of subobject transformation systems (STSs). In this section we review STSs and their construction from a given derivation without NACs.

A subobject A of an object T of a category $\mathbf{C}$ is an equivalence class of monomorphisms $a : A \to T$. We write A for short to denote a representative of the equivalence class and we leave the monomorphism a implicit. The category of subobjects of T is called $\mathbf{Sub}(T)$ and its morphisms $f : A \to B$ are those monomorphisms in $\mathbf{C}$, which are compatible with the implicit monomorphisms to T, i.e. $b \circ f = a$ for $a : A \to T$ and $b : B \to T$. If such an f exists, we write $A \subseteq B$ for short.

Definition 7 (Subobject Transformation Systems) A *Subobject Transformation System* $\mathcal{S} = (S_0, T, P, \pi)$ over an adhesive category $\mathbf{C}$ consists of a super object $T \in \mathbf{C}$, a start object $S_0 \subseteq T$ in $\mathbf{Sub}(T)$, a set of *production names* P, and a function π, which maps a production name q to a *production* $\langle L_q, K_q, R_q \rangle$, where L_q, K_q, and R_q are objects in $\mathbf{Sub}(T)$, $K_q \subseteq L_q$ and $K_q \subseteq R_q$.

The application of a production in an STS is based on union and intersection, which are coproduct and product in category $\mathbf{Sub}(T)$ and they can be constructed in the underlying adhesive category $\mathbf{C}$ as follows: $A \cap B$ is given by the pullback of $A \to T \leftarrow B$ and $A \cup B$ is given by the pushout of $A \leftarrow A \cap B \to B$. The implicit monomorphism of $A \cap B$ is given by $A \cap B \to A \to T$ and the one of $A \cup B$ is the induced one by the pushout property [LS04].

Definition 8 (Direct Derivations) Let $\mathcal{S} = (S_0, T, P, \pi)$ be a Subobject Transformation System, $\pi(q) = \langle L, K, R \rangle$ be a production, and let G be an object of $\mathbf{Sub}(T)$. Then there is a *direct derivation from G to G' using q*, written $G \overset{q}{\Longrightarrow} G'$, if $G' \in \mathbf{Sub}(T)$ and if there exists an object $D \in \mathbf{Sub}(T)$ such that: (i) $L \cup D \cong G$; (ii) $L \cap D \cong K$; (iii) $D \cup R \cong G'$, and (iv) $D \cap R \cong K$.

According to Proposition 6 of [CHS08] a direct derivation in an STS induces a DPO diagram in the underlying adhesive category $\mathbf{C}$. Therefore, a derivation in an STS, specified by its sequence of rule names, gives rise to a derivation in $\mathbf{C}$.

Definition 9 (Derivation of an STS-sequence) Let $d_S = (G_0 \overset{q_1}{\Longrightarrow} G_1 \Rightarrow \ldots \overset{q_n}{\Longrightarrow} G_n)$ be a derivation in an STS $\mathcal{S}$. Let $s = \langle q_1; \ldots; q_n \rangle$ be the sequence of the rule occurrences according to d_S. Then, $drv(s)$ denotes the sequence of DPO diagrams in $\mathbf{C}$ for each derivation $G_{i-1} \overset{q_i}{\Longrightarrow} G_i$ in $\mathcal{S}$.

In order to derive all switch-equivalent derivations to a given one we can analyse its process, which is given by an isomorphism class of occurrence grammars together with a mapping to the original grammar as presented in [CMR96, BCH$^+$06]. An occurrence grammar directly corresponds to an STS and in fact, the construction of an STS in [CHS08] from a given derivation tree coincides with the above construction of an occurrence grammar in the case of a linear derivation, i.e. there is a one-to-one correspondence between both representations. In analogy to

[BCH+06], we explicitly define the start object S_0 of an STS derived from a derivation instead of leaving it implicit. Furthermore, this paper does not consider general derivation trees, but derivation sequences, for which we extend the analysis to the case with NACs in Sec. 5.

Definition 10 (STS of a Derivation) Let $d = (G_0 \xRightarrow{q_1,m_1} \ldots \xRightarrow{q_n,m_n} G_n)$ be a derivation in an adhesive category. The derived STS of d is denoted by $Prc(d)$ and constructed as follows: $Prc(d) = (S_0, T, P, \pi)$, where T is the colimit of the DPO-diagrams given by d, S_0 is given by the embedding $G_0 \subseteq T$, $P = \{(q_i, i) \mid i \in [n]\}$ is a set that contains a rule occurrence name for each rule occurrence in d and π maps each rule occurrence name (q_i, i) to the rule occurrence at the i(th) step in d extended by the embeddings into T. The sequence of rule occurrence names of P according to d is denoted by $seq(d)$.

In [BCH+06] the set of all linearisations of the process is derived using compound relations, which can be obtained from basic relations as shown in [CHS08]. Hence, we can use the following general relation of independence subsuming the basic relations for analysing switch equivalence.

Definition 11 (Independence of Productions in STSs) Let $\mathcal{S} = (S_0, T, P, \pi)$ be an STS and let $q_1, q_2 \in P$ be two production names, and $\pi(p_i) = \langle L_i, K_i, R_i \rangle$ for $i \in \{1,2\}$ be the corresponding productions. Then, q_1 and q_2 are *independent*, denoted $q_1 \diamondsuit q_2$, if

$$(L_1 \cup R_1) \cap (L_2 \cup R_2) \subseteq (K_1 \cap K_2).$$

In the following, we define the switch equivalence for sequences of rule occurrence names within an STS, which will be used in Sec. 5 for the analysis of permutation equivalence.

Definition 12 (Switch-Equivalence of Sequences) Let $\mathcal{S} = (S_0, T, P, \pi)$ be an STS, let d be a derivation in $\mathcal{S}$ and let $s = \langle q_1, \ldots, q_n \rangle$ be its corresponding sequence of rule occurrence names. Let q_k,q_{k+1} be independent in $\mathcal{S}$, then the sequence $s' = \langle q_1, \ldots, q_{k+1}, q_k, \ldots, q_n \rangle$ is switch-equivalent to the sequence s, written $s \overset{sw}{\sim}_{\mathcal{S}} s'$. Switch equivalence $\overset{sw}{\approx}_{\mathcal{S}}$ of sequences is the transitive closure of $\overset{sw}{\sim}_{\mathcal{S}}$.

4 Subobject Transformation Systems with NACs

In this section, we extend the definition of subobject transformation systems to STSs with NACs. Each rule in an STS is extended by an ordered list of NACs, which is later used by the dependency relations in Sec. 5 for specifying the NAC that is causing a dependency.

Definition 13 (STS with NACs) A *Subobject Transformation System with NACs* $\mathcal{S} = (S_0, T, P, \pi)$ over an adhesive category **C** consists of a super object $T \in \mathbf{C}$, a start object $S_0 \subseteq T$ in $\mathbf{Sub}(T)$, a set of production names P, and a function π, which maps a production name q to a *production with NACs* $(\langle L_q, K_q, R_q \rangle, N)$, where L_q, K_q, and R_q are objects in $\mathbf{Sub}(T)$, $K_q \subseteq L_q$, $K_q \subseteq R_q$ and N is an ordered list of negative application conditions with $L \subseteq N[i] \subseteq T$, where $N[i]$ denotes the i(th) element of N.

Direct derivations with NACs in an STS correspond to direct derivations with NACs in the underlying adhesive category, but the check of a NAC to be found in the intermediate object G is simplified, because a NAC in an STS cannot occur at several positions in G.

Definition 14 (Direct Derivations with NACs) Let $\mathcal{S} = \langle S_0, T, P, \pi \rangle$ be a Subobject Transformation System with NACs, $\pi(q) = (\langle L, K, R \rangle, N)$ be a production with NACs, and let G be an object of $\mathbf{Sub}(T)$. Then there is a *direct derivation with NACs* from G to G' using q, written $G \overset{q}{\Longrightarrow} G'$, if $G' \in \mathbf{Sub}(T)$ and for each $N[i]$ in N: $N[i] \nsubseteq G$ and if there exists an object $D \in \mathbf{Sub}(T)$ such that: (i) $L \cup D \cong G$; (ii) $L \cap D \cong K$; (iii) $D \cup R \cong G'$, and (iv) $D \cap R \cong K$.

The extension of the process mapping *Prc* for derivations with NACs in an adhesive category is based on the following instantiation of NACs applied for all derivation steps.

Definition 15 (Instantiated NACs) Let $(N_i, n_i) \in N$ be a NAC of a rule $p = (\underline{p}, N)$ of a grammar $GG = (SG, TG, P, \pi)$ in an adhesive category, with $\underline{p} = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$, let d be a derivation with NACs in GG, where p is applied at step k via the match m_k, and let T be the colimit of the derivation diagram of d. An instantiated NAC M of (N_i, n_i) is a subobject of T $(M \subseteq T)$, such that $M \cong N_i$, $L \subseteq M$ and M is compatible with the typing of N_i and with n_i, i.e. $type_{N_i} = type_T \circ inc_{M,T} \circ iso_M$ and $inc_{L,M} = iso_M \circ n_i$ as shown on the right. The set of all instantiated NACs of p is denoted by $Nacs_T(p, m_k)$.

$$\begin{tikzcd}
N_i \arrow[dr, "iso_M"] \arrow[drr, bend right=40, "type_{N_i}"'] & & L \arrow[ll, tail, "n_i"'] \arrow[dl, hook] \arrow[d, hook] \\
& M \arrow[r, hook] \arrow[ur, phantom, "(=)"] & T \arrow[d, "type_T"] \\
& \arrow[ur, phantom, "(=)"] & TG
\end{tikzcd}$$

Remark 2 *Note that given a NAC, then the set of its instantiated NACs may be empty, which means that the NAC cannot be found within T. Furthermore, a set of instantiated NACs may be infinite if T or d are infinite. In this case the analysis in Sec. 5 may not be decidable. However, in the case of finite derivation sequences and objects being finite in its structural part, e.g. finite graph structure of an attributed graph with an infinite algebra, we get a finite list for each NAC. Note further that $(N_i, inc_{M,T} \circ iso_M) \cong (M, inc_{M,T})$ in $\mathbf{Sub}(T)$.*

Definition 16 (Derived STS with NACs) Let GG be a grammar in an adhesive category $\mathbf{C}$ and let $d = (G_0 \xRightarrow{p_1, m_1} \ldots \xRightarrow{p_n, m_n} G_n)$ be a derivation in GG. The STS for d is given by $Prc(d) = (S_0, T, P, \pi)$, where T is the colimit of the derivation diagram of d, S_0 is given by G_0 and its embedding to T, P is a set of rule names, each distinguished name q_k corresponds to a derivation step $d_k = G_{k-1} \xRightarrow{p_k, m_k} G_k$ in d and the mapping π is given as follows: $\pi(q_k) = (\langle L_k \supseteq K_k \subseteq R_k \rangle, N_k)$, where N_k is an ordered list of the set $Nacs_T(q_k, m_k)$. The order of N_k is arbitrary but fixed. The sequence of rule names of P in d is denoted by $seq(d)$.

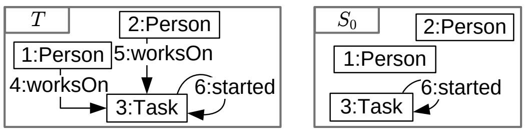


Figure 3: Super object T and start object S_0 of the STS $Prc(d)$

Example 3 (STS of a Derivation) *For the derivation $d = (G_0 \xRightarrow{continueTask, m_1} G_1 \xRightarrow{stopTask, m_2} G_2$*

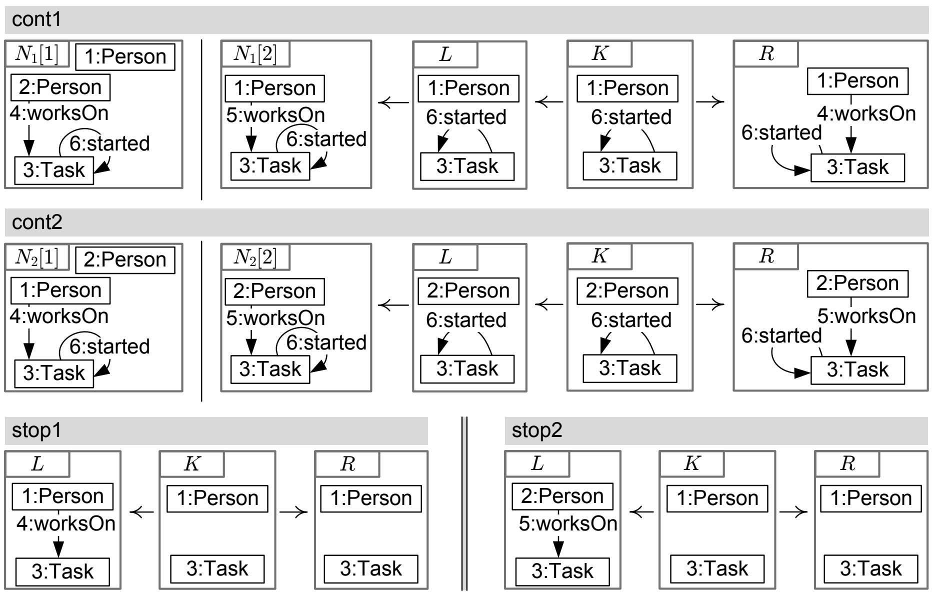


Figure 4: Rule occurrences of the STS $Prc(d)$

$\xrightarrow{continueTask,m_3}$ G_3 $\xrightarrow{stopTask,m_4}$ G_4) in Fig. 2 we construct the STS $\mathcal{S} = Prc(d) = (S_0, T, P, \pi)$ as shown in Fig. 3 and Fig. 4, where numbers denote the embeddings to the colimit T of the derivation diagram. The start object S_0 is given by G_0 in d and its embedding to T. Rule occurrences "cont1" and "cont2" correspond to the first and third derivation step, respectively. Each NAC of "continueTask" can be instantiated once, given by $N_1[1]$ and $N_1[2]$ for "cont1". For "cont2" they are instantiated to $N_2[1]$ and $N_2[2]$. The rule occurrences "stop1" and "stop2" correspond to the second and forth derivation step of d. As explained before in Example 2, no switching is possible in d.

5 Efficient Analysis

In order to analyse dependencies within the process $Prc(d)$ of a derivation d with NACs two new relations are introduced specifying weak enabling and weak disabling dependencies. Here we can use the fact that the NACs in an STS are given as ordered lists to indicate the concrete NAC instantiation that is causing a dependency. These relations are used to characterise the permutation equivalence of derivations with NACs by properties in the derived STS shown by the main technical result of the paper in Thm. 1.

Intuitively, q_1 weakly enables q_2 if it deletes an item that is part of the forbidden structure of the NAC $N_2[i]$ of q_2. This means that there is an element in $L_1 \cap N_2[i]$ that is not contained in $K_1 \cup L_2$. Analogously, q_1 is weakly disabled by q_2 if q_2 produces an item that is part of the forbidden structure of the NAC $N_1[i]$ of q_1.

Definition 17 (Weak NAC Enabling) Let q_1 and q_2 be two rules in an STS and let $N_2[i]$ be a NAC of q_2, i.e. $N_2[i] \in N_2$ for $\pi(q_2) = (\langle L_2, K_2, R_2 \rangle, N_2)$. The relation $<_{wen[i]}$ is defined on P as follows: $q_1 <_{wen[i]} q_2 \Leftrightarrow L_1 \cap N_2[i] \not\subseteq K_1 \cup L_2$.

Definition 18 (Weak NAC Disabling) Let q_1 and q_2 be two rules in an STS and let $N_1[i]$ be NAC of q_1, i.e. $N_1[i] \in N_1$ for $\pi(q_1) = (\langle L_1, K_1, R_1 \rangle, N_1)$. The relation $<_{wdn[i]}$ is defined on P as follows: $q_1 <_{wdn[i]} q_2 \Leftrightarrow N_1[i] \cap R_2 \nsubseteq K_2 \cup L_1$.

Note that the STS is an unfolding of the original derivation. Thus, e.g. for the category **Graphs** we have that elements can be created and deleted, but never re-created after they have been deleted in a derivation of the derived STS, because the colimit construction distinguishes each creation of an element. This implies that each item is either produced by exactly one rule or it is present in the start object and not produced by any rule. Thus, a NAC is satisfied, if an item of the elements it forbids has already been deleted (weak enabling) or such an item is created later (weak disabling). This condition is formalised by the following notion of legal sequences based on the new dependencies. It allows us to first characterise equivalent derivations with NACs within an STS. Afterwards we show that this characterisation is sound and complete for analysing the permutation equivalence of derivations with NACs in the original adhesive category.

Definition 19 (Legal Sequence) Let $d = (d_1; \ldots; d_n)$ be a derivation with NACs in an adhesive category and let $Prc(d) = \mathcal{S} = (S_0, T, P, \pi)$ be its derived STS with NACs. A sequence $s = \langle q_1; \ldots; q_n \rangle$ of rule names of P is locally legal at position $k \in \{1, \ldots, n\}$ with respect to d, if each rule name in P occurs exactly once in s and the following conditions hold:

1. $s \overset{sw}{\approx}_{\mathcal{S}} seq(d)$
2. $\forall$ NACs $N_k[i]$ of q_k : $\begin{pmatrix} \exists\ e \in \{1, \ldots, k-1\}: & q_e <_{wen[i]} q_k \text{ or} \\ \exists\ d' \in \{k, \ldots, n\}: & q_k <_{wdn[i]} q_{d'}. \end{pmatrix}$

The sequence s of rule names is legal with respect to d, if it is locally legal at all positions $k \in \{1, ..., n\}$ with respect to d.

The second condition of Def. 19 considers NACs and ensures that each NAC of a rule cannot be found in the subobject to which the rule is applied, which is a consequence of Thm. 1 and explained above. This subsumes the special case of $s = seq(d)$, where we have that $seq(d)$ is always legal with respect to d. The following definition of permutation equivalence of sequences is based on the notion of legal sequences and therefore, it suffices to evaluate the presented relations on rule occurrence names in order to analyse permutation equivalence of sequences.

Definition 20 (Permutation Equivalence of Sequences) Let d be a derivation with NACs and let $\mathcal{S} = Prc(d) = (S_0, T, P, \pi)$ be its derived subobject transformation system with NACs. Two sequences s, s' of rule names in $\mathcal{S}$ are permutation equivalent, written $s \overset{\pi}{\approx}_{\mathcal{S}} s'$, if they are legal sequences with respect to d.

Now, we are able to state by Thm. 1 below that the analysis of permutation equivalence within the framework of STSs is sound and complete.

Theorem 1 (Analysis of Permutation Equivalence of Derivations based on STSs) *Let d be a derivation with NACs in a grammar GG in an adhesive category and let $\mathcal{S} = Prc(d)$. Then the*

analysis of permutation-equivalence within $\mathcal{S}$ is sound and complete:
Let d' be a derivation with NACs in GG, then: $d \overset{\pi}{\approx} d' \Leftrightarrow seq(d) \overset{\pi}{\approx}_{\mathcal{S}} seq(d')$.

Proof. The proof is given in [Her09a]. □

Based on Thm. 1 we define for a given derivation d the set $EQU(d)$ of all canonical derivations, which are derived from permutation-equivalent sequences of $seq(d)$ in $Prc(d)$. Note that the set $EQU(d)$ is finite, if d is finite.

Definition 21 (Canonical Equivalent Derivations) Let d be a derivation with NACs in grammar GG in an adhesive category and let $\mathcal{S} = Prc(d)$. The set $EQU(d) = \{drv(s') \,|\, s' \overset{\pi}{\approx}_{\mathcal{S}} seq(d)\}$ is called set of canonical permutation-equivalent derivations of d.

Now we can state the second main result of this paper by Thm. 2, which shows that for each derivation d', wich is permutation-equivalent to d, there is an isomorphic representative in the set of canonical equivalent derivations $EQU(d)$.

Theorem 2 (Generation of all Permutation-Equivalent Derivations based on STSs) *Let d be a derivation with NACs in a grammar GG of an adhesive category* **C** *and let $\mathcal{S}$ be the derived STS from d. Then, $\forall\, d' : d \overset{\pi}{\approx} d' \;\exists\, d'' \in EQU(d) : d' \cong d''$.*

Proof. This is a consequence of Def. 21 and Thm. 1, where the derivation d'' is obtained by $drv(s')$ with s' being the sequence of rule occurrence names that correspond to the steps of d'. □

Furthermore, the construction of the process model of a derivation is efficient as stated by the next theorem. This ensures that the effort for the construction of the presented framework does not lead to efficiency problems of the overall analysis.

Theorem 3 (Efficient Construction of the Process Model) *Let d be a derivation with NACs in a grammar GG of the category* **Graphs** *and let $\mathcal{S}$ be the derived STS from d. If additionally the size of each NAC is bounded by the size of the left hand side of the corresponding rule plus an arbitrary but fixed c, then the complexity of the construction of the process model $\mathcal{S}$ and its dependency relations is in $\mathrm{O}(n^{c+4})$, where n is the length of the input $I = (GG, d)$.*

Proof. The super object T is constructed as colimit of d by incremental pushouts for each derivation step and the intermediate colimit object is extended by at most n elements at each step. Thus, this construction has a complexity of $\mathrm{O}(n^2)$. The size of T is at most n, because - in the worst case - T is given as disjoint union of the graphs in d. Furthermore, the construction of S_0 with its embedding to T and the construction of P is linear.

The mapping π is given by composing the morphisms in d with the embeddings to T and instantiation of the NACs. For each derivation step we have at most n NACs of the current rule p. The left hand side of p is already embedded into T and it remains to perform pattern matching for the additional elements (at most c) for each NAC. We derive complexity $\mathrm{O}(n^c \cdot n \cdot n) = \mathrm{O}(n^{2+c})$ and there are at most n^{1+c} NAC-instances for each rule occurrence.

The relations:For each pair of rules we store a Boolean value specifying whether the relation $\Diamond$ holds. We have at most n^2 pairs and use the embeddings to T to check whether they overlap only on interface elements of the rules. Thus, we have a complexity of $\mathsf{O}(n^3)$ and a Boolean array of size at most n^2. For each instantiated NAC (at most n^{1+c}) of a rule occurrence q we check the relation $<_{wdn[i]}$ against each other rule q', i.e. whether the rules overlap on L_q and $K_{q'}$ only. We derive the complexity $\mathsf{O}(n^{c+1} \cdot n^2 \cdot n) = \mathsf{O}(n^{c+4})$ and a Boolean array of size at most n^{c+3}. The same procedure is applied for $<_{wen[i]}$. Summing up all steps, we have complexity $\mathsf{O}(n^{c+4})$. $\square$

Remark 3 (Check for Permutation Equivalence) *If we are interested whether two given derivations d and d' with NACs are permutation-equivalent we can transfer this problem to the usual analysis of switch equivalence without NACs. The reason is that we already know that d and d' respect all NACs. But note that this check also involves isomorphism checks, because the modified structure of an intermediate object in d is not related to some structure in d'.*

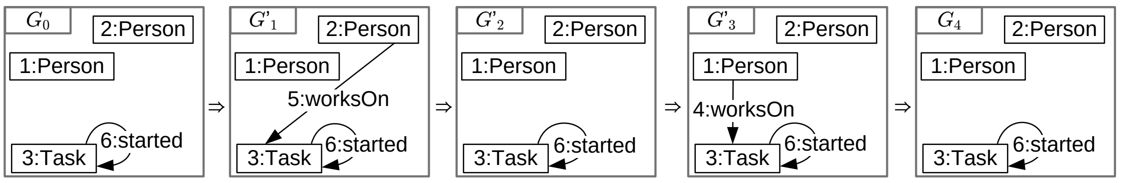
The following claim states that given a derivation d of a graph grammar, then for large intermediate graphs and long derivations the generation of a complete set of representatives for all permutation-equivalent derivations with NACs is more efficient using the derived STS than a direct generation based on the conditions of permutation equivalence.

Claim 1 (Efficiency of the Analysis of Permutation Equivalence of Derivations) *Given a long derivation d in category* **Graphs**, *where the intermediate graphs of d are large. Then, the generation of $EQU(d)$ using $Prc(d)$ is more efficient than a direct generation within* **Graphs**.

Justification of Claim 1: We explain, why the complexity of the direct analysis of permutation equivalence of derivations with NACs in the adhesive category **Graphs** is higher than in the derived STS for long derivations with large intermediate graphs. First of all, sequential independence has to be checked for each switching. Furthermore, we have to construct the new transformation steps according to the switching in order to derive the current intermediate objects. For checking that a switching is valid, we then have to perform pattern matching of the NACs of each derivation step that was involved in a switching. Pattern matching is of high complexity even for medium sized objects, e.g. graphs with more than 20 nodes and edges. Finally, many switchings can be possible at the same time, but not all of them will lead to an equivalent derivation, which implies many backtracking points and paths.

The main advantage of the analysis using STSs is that we do not need to update the graphs of the derivation and we do not need to perform pattern matching for the NACs after each switching. Permutation equivalence of sequences within an STS does only concern the introduced relations on rule occurrence names. This means that we only have to check whether rule components overlap in the specified way. Therefore, we do not need to perform any pattern matching after the STS is constructed. Furthermore, we can compute the relations once and for all and store the results in Boolean arrays. This allows us to efficiently check the relations. A more detailed explanation of the reasons for the claimed efficiency is presented in [Her09a].

Example 4 (Equivalent Sequence) *The presented derivation d' in Fig. 5 is permutation-equivalent to the derivation d in Fig. 2 in Section 2. The last two steps are moved to the beginning. Using the relations in an STS we derive the dependencies as listed in Fig. 5 for the rule occurrences*



$$d' = (G_0 \xRightarrow{continueTask,m'_3} G_1 \xRightarrow{stopTask,m'_4} G_2 \xRightarrow{continueTask,m'_1} G_3 \xRightarrow{stopTask,m'_2} G_4)$$

write causality	weak enabling	weak disabling
cont1 $<_{wc}$ stop1	stop1 $<_{wen[1]}$ cont1	cont1 $<_{wdn[1]}$ cont1
cont2 $<_{wc}$ stop2	stop1 $<_{wen[1]}$ cont2	cont1 $<_{wdn[1]}$ cont2
	stop2 $<_{wen[2]}$ cont1	cont2 $<_{wdn[2]}$ cont2
	stop2 $<_{wen[2]}$ cont2	cont2 $<_{wdn[2]}$ cont1

Figure 5: Derivation d' equivalent to d in GG and the table of its dependencies

"cont1, cont2, stop1, stop2" in Fig. 4, where the basic relation for write causality "$q_1 <_{wc} q_2$" as defined in [CHS08] is a special case of dependency given by $\neg(q_1 \Diamond q_2)$. This implies that "cont1" has to occur before "stop1" and "cont2" before "stop2". According to Thm. 1 and Def. 20 we can check, whether the sequence $seq(d')$ can be derived by the presented relations. First of all, the sequence is derived by switchings of independent rule occurrences without considering NACs: stop1 $\leftrightarrow$ cont2, stop1 $\leftrightarrow$ stop2, cont1 $\leftrightarrow$ cont2 and cont1 $\leftrightarrow$ stop2 leading to $s = \langle cont2;stop2;cont1;stop1\rangle = seq(d')$. Now, for each rule occurrence and NAC in s there is a weak enabling rule before or a weak disabling rule behind or the rule disables itself. For instance for $N_1[2]$ of "cont1" we have "stop2" with stop2 $<_{wen[2]}$ cont1 and therefore, $N_1[2]$ is not present in the intermediate object. All together $s = seq(d')$ is a legal sequence with respect to d, which implies that $seq(d') \overset{\pi}{\approx}_S seq(d)$ and hence, $d' \overset{\pi}{\approx} d$ according to Thm. 1.

Note that a pairwise switching of the example derivation with NACs is not possible, because each pair is sequentially dependent - either by causal relation or by NAC dependency. Therefore, this sequence cannot be derived by standard switching of completely independent derivation steps according to switch equivalence with NACs in Def. 5. This shows that switch equivalence with NACs based on sequential independence of derivations with NACs [HHT96, LEO06, LEOP08] only leads to a subclass of equivalent derivations and in general, many equivalent derivations cannot be derived. But as the example derivation shows, all permutation-equivalent derivations are of interest, because a certain person may not be available for a concrete time slot while another person could use the time and give some support for the task.

6 Conclusion and Future Work

Up to now, process analysis of transformation systems did not consider negative application conditions (NACs), which are widely used in practical case studies and applications. Switch equivalence based on sequential independence of derivations with NACs [HHT96, LEO06] is not sufficient, because rule applications may be possible in an equivalent way at several positions of the derivation, which are not situated next to each other and these permutation-equivalent derivations cannot be derived by switching sequentially independent steps as explained with the presented example. Furthermore, also critical pair analysis [LEPO08] for systems with NACs only leads to partial results in this context. Critical pairs specify the possible conflicts of productions without considering the current instance objects on which productions are applied to in a concrete derivation. But in most cases rules have the ability to conflict each other. Analogously, the criteria for the applicability of rule sequences in [LET08] cannot be applied, because they are only sufficient but not necessary, and furthermore, matches can be arbitrary and do not have to be equivalent.

For this reason, we introduced permutation equivalence for derivations with NACs. From a general point of view permutation equivalence is maximal in abstraction, because it relates two derivations, if they start at the same object, they end at the same object, both derivations are valid, each one can be obtained by the other by permuting the applied rules, and finally all matches are equivalent with respect to the gluing of all intermediate instances.

The main result of this paper is a framework for the efficient analysis of permutation equivalence, i.e. the efficient derivation of all derivations, which are permutation-equivalent to a given one. The presented construction of a process model for derivations with NACs is based on sub-object transformation systems (STSs) [CHS08], which are defined for the abstract setting of adhesive categories. Thus, the process analysis can be instantiated to several concrete categories.

The main benefit of using STSs in this context is that the construction of the process model can be performed in polynomial time and in advance, i.e. possibly before a user requests an analysis. Furthermore, the relations for the analysis are based on overlappings, which implies that there is no need for pattern matching and updates of the derivation in order to analyse permutation equivalence of derivations with NACs. A direct analysis in the adhesive category of the derivation would cause high complexity as explained in detail in the previous section. In particular, many of the possible permutations have to be constructed and checked including the high complexity of pattern matching for the NACs on the updated intermediate objects.

This paper is a fundamental contribution to the PhD project: Process Construction and Analysis for Workflows modelled by Adhesive HLR Systems with Application Conditions [Her08]. While this paper concerns the extension of the process construction for NACs in the setting of adhesive categories, also positive application conditions will be integrated in a further step, which allow the modeller to specify rules that are more compact, which additionally leads to a bigger class of permutation-equivalent derivations in general. Furthermore, the results will be transferred to the more general class of adhesive high level replacement systems (AHLR systems) [EEPT06] as already explained in [HE08, Her09b]. The developed techniques will be instantiated to typed attributed graph transformation systems and further more, to Petri net transformation systems, which are used for modelling mobile networks.

The benefits of the overall framework of process analysis will be elaborated in future case studies for reconfigurable workflow models as instances of AHLR systems, which shall show that analysis and execution are efficient and convenient.While the running example of this paper is simplified to show the results on compact instances, a full case study of mobile reconfigurable workflow systems will cover complex scenarios and in particular emergency scenarios. A motivation for the case study is to compute equivalent executions that show maximal parallelism or improved properties with respect to the application domain.

Acknowledgements: I am grateful to the anonymous reviewers, who gave many important suggestions on how to improve the paper.

Bibliography

[Bal00] P. Baldan. *Modelling Concurrent Computations: from Contextual Petri Nets to Graph Grammars*. PhD thesis, Computer Science Department - University of Pisa, 2000.
http://www.math.unipd.it/~baldan/Papers/Soft-copy-pdf/Tesi.pdf

[BCH+06] P. Baldan, A. Corradini, T. Heindel, B. König, P. Sobocinski. Processes for Adhesive Rewriting Systems. In Aceto and Ingólfsdóttir (eds.), *FoSSaCS*. Lecture Notes in Computer Science 3921, pp. 202–216. Springer, 2006.

[CHS08] A. Corradini, F. Hermann, P. Sobociński. Subobject Transformation Systems. *Applied Categorical Structures* 16(3):389–419, February 2008.
doi:10.1007/s10485-008-9127-6

[CMR96] A. Corradini, U. Montanari, F. Rossi. Graph Processes. *Fundamenta Informaticae* 26(3/4):241–265, 1996.

[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. World Scientific, 1997.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theoretical Computer Science. Springer Verlag, 2006.
http://www.springer.com/3-540-31187-4

[HE08] F. Hermann, H. Ehrig. Process Definition using Subobject Transformation Systems. *Bulletin of the EATCS* 95:153–163, 2008.
http://www.eatcs.org/index.php/eatcs-bulletin

[Her08] F. Hermann. Process Construction and Analysis for Workflows Modelled by Adhesive HLR Systems with Application Conditions. In Ehrig et al. (eds.), *Proc. International Conference on Graph Transformation (ICGT'08)*. LNCS 5214, pp. 496–498. Springer Verlag, Heidelberg, 2008.
doi:10.1007/978-3-540-87405-8_44

[Her09a] F. Hermann. Permutation Equivalence of DPO Derivations with Negative Application Conditions based on Subobject Transformation Systems (Long Version). Technical report 2009/10, TU Berlin, 2009. (to appear).
http://tfs.cs.tu-berlin.de/publikationen/Papers09/Her09a.pdf

[Her09b] F. Hermann. Process Definition of Adhesive HLR Systems. Technical report, Technische Universität Berlin,Fakultät IV, 2009. (to appear).

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

[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 Verlag, Natal, Brazil, September 2006.
http://tfs.cs.tu-berlin.de/publikationen/Papers06/LEO06a.pdf

[LEOP08] 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.), *Proceedings of the ACCAT workshop at ETAPS 2007*. ENTCS 203 / 6, pp. 43–66. Elsevier, 2008.
http://tfs.cs.tu-berlin.de/publikationen/Papers08/LEOP08.pdf

[LEPO08] L. Lambers, H. Ehrig, U. Prange, F. Orejas. Embedding and Confluence of Graph Transformations with Negative Application Conditions. In Ehrig et al. (eds.), *Proc. International Conference on Graph Transformation (ICGT'08)*. LNCS 5214, pp. 162–177. Springer Verlag, Heidelberg, 2008.
http://tfs.cs.tu-berlin.de/publikationen/Papers08/LEPO08.pdf

[LET08] L. Lambers, H. Ehrig, G. Taentzer. Sufficient Criteria for Applicability and Non-Applicability of Rule Sequences. In C. Ermel and Heckel (eds.), *Proc. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'08)*. Volume 10. EC-EASST, 2008.
http://eceasst.cs.tu-berlin.de/index.php/eceasst/issue/view/19

[LS04] S. Lack, P. Sobociński. Adhesive Categories. In *Proc. FOSSACS 2004*. LNCS 2987, pp. 273–288. Springer, 2004.