



Proceedings of the
Ninth International Workshop on
Graph Transformation and
Visual Modeling Techniques
(GT-VMT 2010)

Efficient Analysis of Permutation Equivalence of
Graph Derivations Based on Petri Nets

Frank Hermann , Andrea Corradini , Hartmut Ehrig , Barbara König

14 pages

Efficient Analysis of Permutation Equivalence of Graph Derivations Based on Petri Nets

Frank Hermann¹, Andrea Corradini², Hartmut Ehrig¹, Barbara König³

¹ [frank,ehrig]@cs.tu-berlin.de, Institut für Softwaretechnik und Theoretische Informatik, Technische Universität Berlin, Germany

² andrea(at)di.unipi.it, Dipartimento di Informatica, Università di Pisa, Italy

³ barbara_koenig(at)uni-due.de, Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen, Germany

Abstract: In the framework of graph transformation systems with Negative Application Conditions (NACs) the classical notion of switch equivalence of derivations is extended to *permutation equivalence*, because there are intuitively equivalent derivations which are not switch-equivalent if NACs are considered. By definition, two derivations are permutation-equivalent, if they respect the NACs and disregarding the NACs they are switch equivalent. A direct analysis of permutation equivalence is very complex in general, thus we propose a much more efficient analysis technique. For this purpose, we construct a Place/Transition Petri net, called dependency net, which encodes the dependencies among rule applications of the derivation, including the inhibiting effects of the NACs.

The analysis of permutation equivalence is important for analysing simulation runs within development environments for systems modelled by graph transformation. The application of the technique is demonstrated by a graph transformation system within the context of workflow modelling. We show the effectiveness of the approach by comparing the minimal costs of a direct analysis with the costs of the efficient analysis applied to a derivation of our example system.

Keywords: graph transformation, Petri nets, process analysis, adhesive categories

1 Introduction

Given a workflow of a system, it is often interesting to know whether the workflow can be improved, by executing the tasks in a different order, which might be more convenient for the user or preferable from an efficiency point of view. If the workflow is modelled by a Petri net, representing a deterministic process, these questions can be fairly easily answered: processes incorporate a notion of concurrency that can be exploited to rearrange the tasks, while still respecting causality.

In this paper we consider workflow models with two further dimensions, which considerably complicate the problem: first, we work in the general setting of (weak) adhesive categories [LS05, EEPT06] where we can model systems with an evolving topology, such as (attributed) graph transformation systems, in contrast to systems with a static structure. For the sake of con-

ciseness, the definitions and results in this paper are presented for *graph* transformation systems, and we refer to the companion technical report [HCEK10] for the general notions based on adhesive categories. As a second dimension, we take into account Negative Application Conditions (NACs) that are used to ensure the “absence” of forbidden structures when executing a transformation step: NACs significantly improve the specification formalisms based on transformation rules leading to more compact and concise models as well as increased usability and as a matter of fact they are widely used in non-trivial applications. The presence of NACs leads to more complex interdependencies of tasks.

For this reason, we introduce a notion of permutation equivalence on derivations with NACs, which is coarser and more adequate than the switch equivalence in the double-pushout (DPO) approach including NACs. As defined in [Her09] two derivations are called permutation-equivalent, if they respect the NACs and disregarding the NACs they are switch-equivalent. Using the notion of switch equivalence with NACs directly does not lead to all permutation-equivalent derivations of a given derivation in general. The main remaining problem is how to derive the complete set of all permutation-equivalent derivations to a given one. For this purpose, we construct a subobject transformation system (STS) via a standard colimit construction and from this STS we construct a dependency net, given by a standard P/T Petri net, which includes a complete account of the inhibiting effects of the NACs. The main result shows that complete firing sequences of this net are one-to-one with derivations that are permutation-equivalent to the given derivation, allowing us to derive the complete set of permutation-equivalent derivations. Finally, for a given derivation of a simple example system with NACs, we perform a detailed complexity analysis of the cost of identifying all permutation equivalent derivations using the reduction to a Petri net and its reachability graph, and compare it with a lower bound of the costs for a direct analysis, i.e. for computing all shift-equivalent derivations first, and then filtering out the ones which do not respect the NACs. We obtain a significant improvement in speed, which shows that the proposed technique can be efficient for many applications which involve the generation of permutation-equivalent derivations. Furthermore, the constructed P/T Petri net can be used to derive specific permutations without generating the complete set first. In the context of workflow analysis, both goals are of central interest for the modelling of a system.

The structure of the paper is as follows. Sec. 2 reviews the main concepts of permutation equivalence for graph transformation systems. The construction of the dependency net is presented in Sec. 3 and in our main result it is shown to be sound and complete for computing the set of permutation-equivalent derivations. Sec. 4 validates the efficiency of the analysis based on an extended version of the running example. Finally, Sec. 5 sums up the main results, discusses related work, and points out aspects of future work.

2 Permutation Equivalence

In this section we review the standard switch equivalence and the recently introduced permutation equivalence [Her09] for graph transformation systems based on the double pushout (DPO) approach. The running example of this paper illustrates that there are derivations which are intuitively equivalent and also permutation-equivalent but not switch-equivalent.

Definition 1 (Graph Transformation System with NACs) A rule $p = (L_p \xleftarrow{l} K_p \xrightarrow{r} R_p)$ is a pair of injective graph morphisms. A *Negative Application Condition (NAC)* for a rule p is an injective graph morphism $n : L_p \hookrightarrow N$, having the left-hand side of p as source. A *rule with NACs* is a pair $\langle p, \mathbf{N} \rangle$ where p is a rule and $\mathbf{N} = \{n_i : L_p \hookrightarrow N_i\}_{i \in I}$ is a finite set of NACs for p . A *match* of a rule p in a graph G is an injective graph morphism¹ $m : L_p \hookrightarrow G$; match m satisfies the NAC $n : L_p \hookrightarrow N$ for p , written $m \models n$, if there is no arrow $g : N \rightarrow G$ such that $g \circ n = m$.² We say that there is a *direct derivation* $G \xrightarrow{p,m} H$ from an object G to H using a rule with NACs $\langle p, \mathbf{N} \rangle$ and a match $m : L_p \rightarrow G$, if there are two pushouts (1) and (2) in **Graphs**, as depicted. A derivation *respects the NACs*, if $m \models n$ for each NAC $(n : L_p \hookrightarrow N) \in \mathbf{N}$. A *typed graph transformation system (GTS) with NACs* is a tuple $\mathcal{G} = \langle Q, \pi_N \rangle$ where Q is a set of rule names, and π_N maps each name $q \in Q$ to a rule with NACs $\pi_N(q) = \langle \pi(q), \mathbf{N}_q \rangle$ in the category **Graphs**_{TG} of graphs typed over a given type graph TG. A *derivation (respecting NACs)* of \mathcal{G} is a sequence $G_0 \xrightarrow{q_1, m_1} G_1 \cdots \xrightarrow{q_n, m_n} G_n$, where $q_1, \dots, q_n \in Q$ and $d_i = G_{i-1} \xrightarrow{\pi(q_i), m_i} G_i$ are direct derivations (respecting NACs) for $i \in 1, \dots, n$. Sometimes we denote a derivation as a sequence $d = d_1; \dots; d_n$ of direct derivations.

$$\begin{array}{ccccc}
 N & \xleftarrow{n} & L_p & \xleftarrow{l} & K_p & \xrightarrow{r} & R_p \\
 & \searrow & \downarrow m & \text{(1)} & \downarrow & \text{(2)} & \downarrow \\
 & & G & \longleftarrow & D & \longrightarrow & H
 \end{array}$$

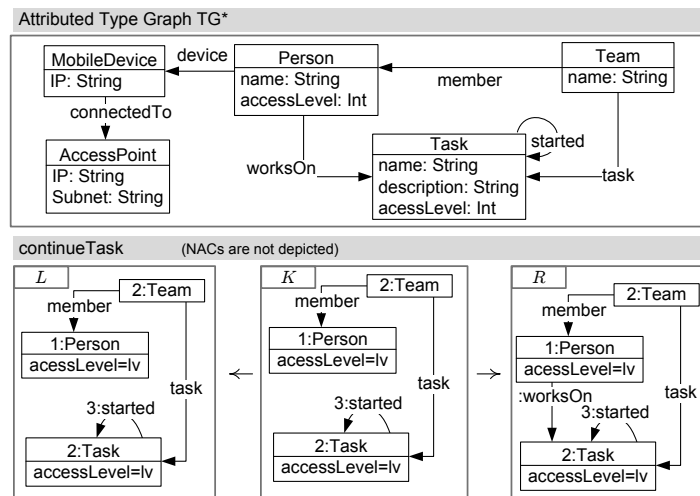
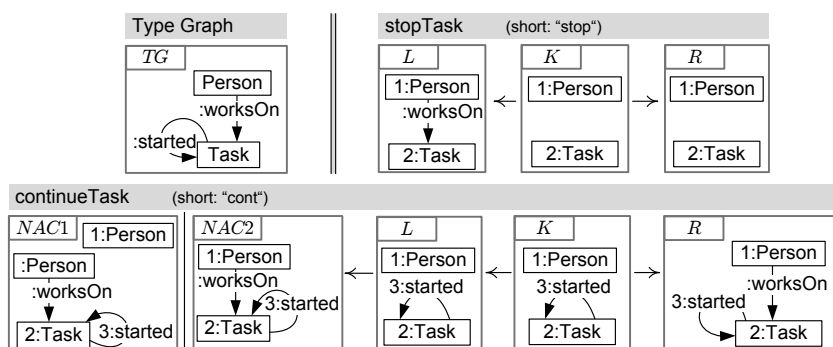


Figure 1: Part of attributed transformation system GS^* , modeling mobile adhoc networks

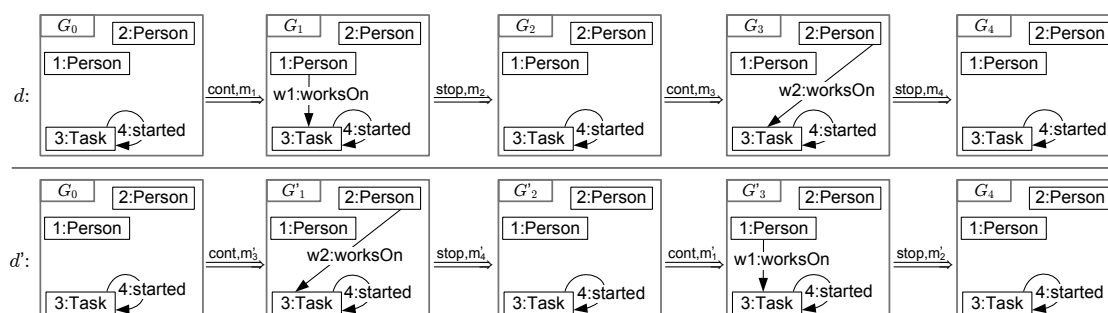
Example 1 (Graph Transformation System with NACs) Fig. 1 shows a part of an attributed graph grammar for modelling a workflow system in mobile adhoc networks, where persons can be assigned to teams and tasks and they can change their location implying that their mobile communication devices may need to reconnect to new access points. In order to simplify the further constructions we will use the reduced version of this grammar in Fig. 2. The type graph TG shows that nodes in the system represent either persons or tasks: a task is active if it has a

¹ In the general case NAC-morphisms $(n : L \rightarrow N)$ and matches are not required to be injective. For the general case, our technique can be extended by the results in [HE08].

² Intuitively, the image of L_p in G cannot be extended to an image of the “forbidden context” N .

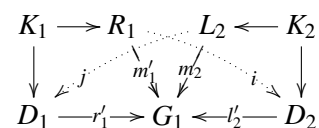

 Figure 2: Reduced transformation system GS as running example

“:started” loop, and it can be assigned to a person with a “:worksOn” edge. Rule “stopTask” cancels the assignment of a task to a person; rule “continueTask” instead assigns the task, and it has two NACs to ensure that the task is not assigned to a person already. Fig. 3 shows two derivations respecting NACs of GS . In derivation d the only task is first continued by “1:Person”, and then, after being stopped, by “2:Person”. In d' the roles of the two Persons are inverted.


 Figure 3: Derivation d (respecting NACs) of GS and permutation-equivalent derivation d'

The classical theory of the DPO approach (without NACs) introduces an equivalence among derivations which relates derivations that differ only in the order in which independent direct derivations are performed (see [Kre86, BCH⁺06]). The *switch equivalence* is based on the notion of *sequential independence* and on the *Local Church-Rosser theorem*. This is briefly summarised in the next definition.

Definition 2 (Switch Equivalence on Derivations) Let $d_1 = G_0 \xrightarrow{p_1, m_1} G_1$ and $d_2 = G_1 \xrightarrow{p_2, m_2} G_2$ be two direct derivations. Then they are *sequentially independent* if there exist arrows $i: R_1 \rightarrow D_2$ and $j: L_2 \rightarrow D_1$ such that $l'_2 \circ i = m'_1$ and $r'_1 \circ j = m_2$ (see the diagram on the right, which shows part of the derivation diagrams). If d_1 and d_2 are sequentially independent, then according to the Local Church Rosser Theorem (Thm. 5.12 in [EEPT06]) they can be “switched” obtaining direct derivations $d'_2 = G_0 \xrightarrow{p_2, m_2} G'_1$ and $d'_1 = G'_1 \xrightarrow{p_1, m_1} G_2$, which apply the two rules in the opposite order.



Now, let $d = (d_1; \dots; d_k; d_{k+1}; \dots; d_n)$ be a derivation, where d_k and d_{k+1} are two sequentially independent direct derivations, and let d' be obtained from d by switching them according to

the Local Church Rosser Theorem. Then, d' is a *switching of d* , written $d \overset{sw}{\sim} d'$. The *switch equivalence*, denoted $\overset{sw}{\approx}$, is the smallest equivalence on derivations containing both $\overset{sw}{\sim}$ and the relation \cong for isomorphic derivations.

Corresponding notions of parallel and sequential independence have been proposed for graph transformation systems with NACs [HHT96, LEO06]. However, the derived notion of switch equivalence does not identify all intuitively equivalent derivations. The reason is that, in presence of NACs, there might be an equivalent permutation of the direct derivations that cannot be derived by switch equivalence. Looking at d in Fig. 3 there is no pair of consecutive direct derivations which is sequentially independent if NACs are considered. However, the derivation d' should be considered as equivalent. There are also examples in which even the switching of blocks of several steps would not lead to all permutation-equivalent derivations. This brings us to the following, quite natural notion of permutation equivalence of derivations respecting NACs, first proposed in [Her09]. Note that for permutation-equivalent derivations $d \overset{\pi}{\approx} d'$ the sequence of rules used in d' is a permutation of those used in d .

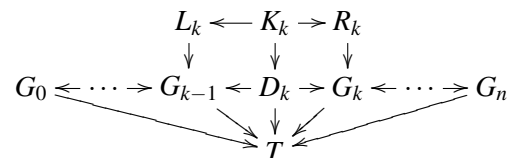
Definition 3 (Permutation Equivalence of Derivations) Two derivations d and d' respecting NACs are *permutation equivalent*, written $d \overset{\pi}{\approx} d'$ if, disregarding the NACs, they are switch equivalent as for Def. 2.

3 Dependency Net of a Derivation

In order to efficiently analyse permutation equivalence of derivations we introduce the construction of the dependency net of a given derivation with NACs. This Place/Transition Petri net purely encodes the dependencies between the derivation steps. The reachability graph of this net with initial marking determines the class of derivations which are permutation-equivalent to a given one.

The construction of the dependency net is based on the construction of the subobject transformation system (STS) of a given derivation d according to [CHS08] and its extension to NACs in [Her09]. Subobjects of a graph G form the category of subobjects $\mathbf{Sub}(G)$, which contains subgraphs of G^3 as objects and injective graph morphisms $m : G_1 \rightarrow G_2$ between subgraphs as morphisms, where m is required to respect the injective embeddings of G_1 and G_2 to G . We will write $G_1 \cap G_2$ for the componentwise intersection and $G_1 \cup G_2$ for the componentwise union of subgraphs of G , where items are identified with respect to the injective embeddings of G_1 and G_2 into G .

In order to construct the STS for a derivation $d = (d_1; \dots; d_n)$ we compute the colimit T of the sequence of DPO diagrams, where all morphisms are injective. Thus, all objects and morphisms of this diagram are in the category $\mathbf{Sub}(T)$. The NACs of the rules do not occur in this diagram.



³ More formally, a subgraph is given by an equivalence class of injective graph morphisms to G , such that the image of all morphism is equal.

Definition 4 (STS of a derivation) A *subobject transformation system* $\mathcal{S} = \langle T, Q, \pi \rangle$ consists of a super object T , a set of rule names Q , and a function π , which maps a name $q \in Q$ to a *rule*, i.e., to a triple $\pi(q) = \langle L_q, K_q, R_q \rangle$ of subobjects of T such that $K_q = L_q \cap R_q$.

Now, let $\mathcal{G} = \langle Q, \pi \rangle$ be a graph transformation system, and let $d = (G_0 \xrightarrow{q_1, m_1} \dots \xrightarrow{q_n, m_n} G_n)$ be a derivation of \mathcal{G} . The *STS generated from d* is defined as $STS(d) = \langle T, P, \hat{\pi} \rangle$, where T is the colimit object of the diagram underlying the derivation d , $P = \{k \mid d_k = (G_{k-1} \xrightarrow{q_k, m_k} G_k) \text{ is a step of } d\}$, and $\hat{\pi}(k) = \langle L_k, K_k, R_k \rangle$, where $q_k = (L_k \leftarrow K_k \rightarrow R_k)$.

For the rest of the paper, we consider only derivations such that the colimit T is a *finite object*, i.e. $\mathbf{Sub}(T)$ is a finite lattice. This is guaranteed if each rule of \mathcal{G} has finite left- and right-hand sides, and if the start object of the derivation is finite. The generation of an STS with NACs from a given derivation works as in Definition 4, but additionally each rule will be equipped with a list of NACs, i.e., those obtained as “instances” of the original NACs in the colimit object T . Note that one original NAC can have several instances, but also not a single one.

Definition 5 (Instantiated NACs) Let $d = d_1; \dots; d_k; \dots; d_n$ be a derivation respecting NACs and let T be the colimit object of the derivation. Let $\langle p, \mathbf{N} \rangle$ be the rule with NACs used in direct derivation d_k and let $NACS(p) = \{n : L_p \hookrightarrow N \mid n \in \mathbf{N}\}$. The set of all instantiated NACs in T of the NACs of a rule p is given by $NACS_T(p) = \{N \xrightarrow{t_N} T \text{ in } \mathbf{Sub}(T) \mid n \in \mathbf{N}, \text{ s.t. } t_N \circ n = t_L\}$ for $L_p \xrightarrow{t_L} T$ in $\mathbf{Sub}(T)$.

Definition 6 (STS of a Derivation with NACs) Let \mathcal{G} be a GTS with NACs and let d be a derivation of \mathcal{G} respecting NACs. The *STS with NACs generated by d* is given by $STS_N(d) = \langle T, P, \hat{\pi}_N \rangle$, where T and P are as in Def. 4, $\hat{\pi}_N(k) = \langle \hat{\pi}(k), \mathbf{N}_k \rangle$, $\hat{\pi}(k)$ is as in Def. 4, and \mathbf{N}_k is an arbitrary but fixed linearisation of the instantiated NACs $NACS_T(p_k)$ as in Def. 5, where p_k is the rule of \mathcal{G} used in d_k .

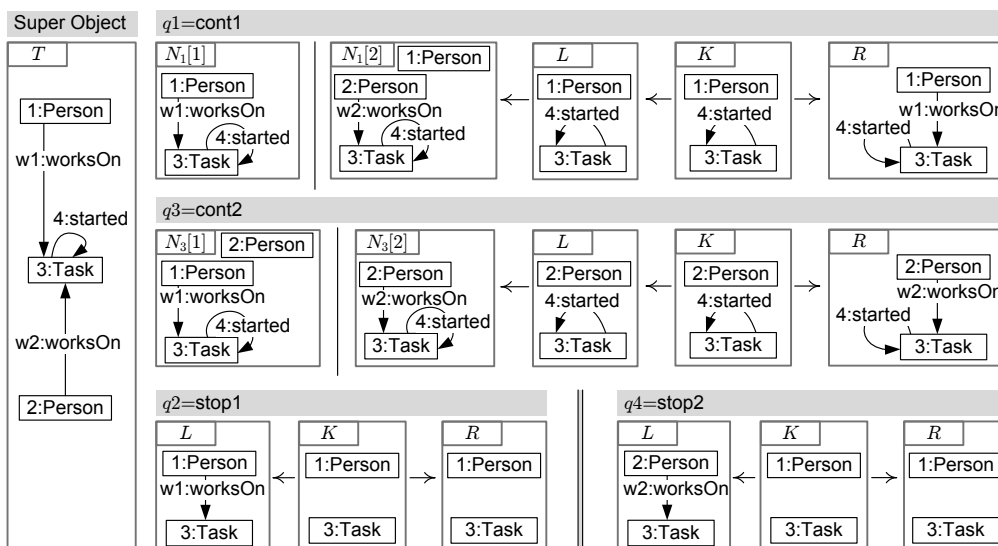


Figure 4: Derived Subobject Transformation System $STS_N(d)$

Example 2 (Derived STS $STS_N(d)$) For the derivation d in Ex. 1 we derive the STS as shown in Fig. 4. The super object T is derived by taking the first graph of the derivation and adding the items, which are created during the transformation, i.e. the two edges of type “worksOn”. The derivation d involves the rules “continueTask” and “stopTask” and thus, the derived STS contains the rule occurrences “1 \triangleq cont1”, “2 \triangleq stop1”, “3 \triangleq cont2” and “4 \triangleq stop2”, where the NACs of the rule “continueTask” are instantiated.

The following relations between the rules of an STS with NACs specify the possible dependencies among them: the first four relations are discussed in [CHS08], while the last two are introduced in [Her09]. In our case the STS with NACs is generated from a derivation d according to Def. 6.

Definition 7 (Relations on Rules) Let q_1 and q_2 be two rules in an STS with NACs $S = (T, P, \pi_N)$ with $\pi_N(q_j) = (\langle L_j, K_j, R_j \rangle, \mathbf{N}_j)$ for $j \in \{1, 2\}$ and $\mathbf{N}_j = (N_j[i])_{i=1..n_j}$. The relations on rules are defined on P as follows:

Name	Notation	Condition
Read Causality	$q_1 <_{rc} q_2$	$R_1 \cap K_2 \not\subseteq K_1$
Write Causality	$q_1 <_{wc} q_2$	$R_1 \cap L_2 \not\subseteq K_1 \cup K_2$
Deactivation	$q_1 <_d q_2$	$K_1 \cap L_2 \not\subseteq K_2$
Independence	$q_1 \diamond q_2$	$(L_1 \cup R_1) \cap (L_2 \cup R_2) \subseteq K_1 \cap K_2$
Weak NAC Enabling	$q_1 <_{wen[i]} q_2$	$1 \leq i \leq \mathbf{N}_2 \wedge L_1 \cap N_2[i] \not\subseteq K_1 \cup L_2$
Weak NAC Disabling	$q_1 <_{wdn[i]} q_2$	$1 \leq i \leq \mathbf{N}_1 \wedge N_1[i] \cap R_2 \not\subseteq L_1 \cup K_2$

Read causality specifies that rule q_1 produces an item that is read by q_2 , but not deleted by q_2 and in the case of write causality we have that q_2 also deletes such an item. Deactivation occurs when rule q_2 deletes an item that is read by q_1 , but not created and two rule occurrences are independent if they overlap only on items that are neither produced nor deleted by one of the rules. Rule q_1 weakly enables the rule q_2 at i if q_1 deletes a forbidden part q_2 , i.e. an item of the i -th NAC of q_2 that is not contained in L_2 . The rule q_2 weakly disables q_1 at i if q_2 produces a piece of the i -th NAC of q_1 . It is worth stressing that the relations introduced above are not transitive in general.

Example 3 (Relations on Rules) The rules of $STS_N(d)$ in Fig. 4 are related by the following dependencies. For write causality we have “cont1 $<_{wc}$ stop1” and “cont2 $<_{wc}$ stop2”. Weak enabling/disabling are shown in the table below, while read causality and deactivation are empty.

Weak Enabling		Weak Disabling	
$stop1 <_{wen[1]} cont1$	$stop2 <_{wen[2]} cont1$	$cont1 <_{wdn[1]} cont1$	$cont2 <_{wdn[2]} cont2$
$stop1 <_{wen[1]} cont2$	$stop2 <_{wen[2]} cont2$	$cont2 <_{wdn[1]} cont1$	$cont1 <_{wdn[2]} cont2$

Based on the STS of a derivation, we now present the construction of its “dependency net”,

given by a P/T Petri net which specifies only the dependencies between the derivation steps. All details about the internal structure of the graphs and the transformation rules are excluded, allowing us to improve the efficiency of the analysis of permutation equivalence.

Definition 8 (Dependency Net $DNet$ of a derivation) Let $d = (d_1; \dots; d_n)$ be a derivation respecting NACs of a GTS with NACs, let $STS_N(d) = (T, P, \hat{\pi})$ be the generated STS with NACs and let $s = seq(d) = \langle q_1, \dots, q_n \rangle = \langle 1, \dots, n \rangle$ denote the sequence of rule names in P according to the steps in d . The *dependency net* of d is given by the marked Petri net $DNet(d) = \langle N, M \rangle$, $N = \langle PL, TR, pre, post \rangle$, constructed by the steps in Fig. 5, where the steps are performed in the order they appear in the table.

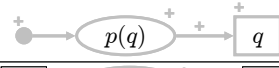
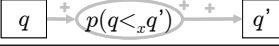
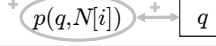
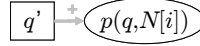
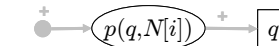
$STS(d) = (T, P, \hat{\pi})$		$DNet(d) = ((PL, TR, pre, post), M)$
1. For each $q \in P$		
2. For all $q, q' \in P$, $q <_x q'$, $x \in \{rc, wc, d\}$		
3. For all $q \in P$ with $q \not<_{wdn[i]} q$, $i \in \mathbb{N}$		
a)	$N[i]$ of q	
b)	For all $q' \in P$: $q' <_{wen[i]} q$	
c)	For all $q' \in P$: $q <_{wdn[i]} q'$	

Figure 5: Construction of the Dependency Net

Fig. 5 shows the steps of the construction of the dependency net. The steps are given as visualized rules, where gray line colour and plus-signs mark the inserted elements. In the first step they are created without context, but e.g. in step two the new place “ $p(q <_x q')$ ” is inserted between the already existing transitions q and q' . The tokens of the marked Petri net are represented by bullets that are connected to their places by arcs. The first step creates a transition for each rule and the transition is connected to a marked place for ensuring that it cannot fire twice. In step 2, between each pair of transitions in each of the relations $<_{rc}$, $<_{wc}$ and $<_d$, a new place is created in order to enforce the corresponding dependency. The rest of the construction is concerned with places which correspond to NACs and can contain several tokens in general. Each token in such a place represents *the absence* of a piece of the NAC; therefore if the place is empty, the NAC is complete. In this case, by step (3a) the transition cannot fire. Consistently with this intuition, if $q' <_{wen[i]} q$, i.e. transition q' consumes part of the NAC $N[i]$ of q , then by step (3b) q' produces a token in the place corresponding to $N[i]$. Symmetrically, if $q <_{wdn[i]} q'$, i.e. q' produces part of NAC $N[i]$ of q , then by step (3c) q' consumes a token from the place corresponding to $N[i]$. Notice that each item of a NAC is either already in the start graph of the derivation or produced by a single rule. Furthermore, if a rule generates a part of one of its NACs, say $N[i]$ ($q <_{wdn[i]} q$), then by the acyclicity of $STS_N(d)$ the NAC $N[i]$ cannot be completed before the firing of q : therefore we ignore it in the third step of the construction of the dependency net.

A more formal definition of the construction, which explicitly defines the sets PL, TR , the pre and post mappings as well as the marking $M \in PL^\oplus$, is given by Def. 12 in [HCEK10]. Note

that the constructed net in general is not a safe one, because the places for the NACs can contain several tokens. Nevertheless it is a bounded P/T net, where the bound to the number of tokens is given by the maximum, taken over places representing NACs, of the number of rules that either weakly disable or weakly enable the specific NAC.

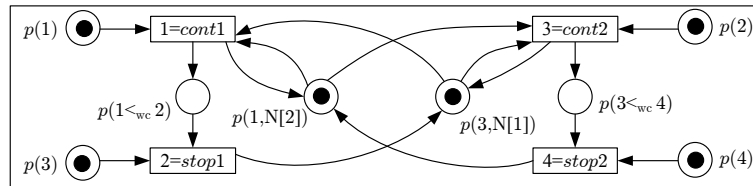


Figure 6: Dependency Net $DNet(d)$ as Petri Net

Example 4 (Dependency Net) Consider the derivation d from Ex. 1 and its derived STS in Ex. 2. The marked Petri net shown in Fig. 6 is the dependency net $DNet(d)$ according to Def. 8. The places encoding the write causality relation are “ $p(1 <_{wc} 2)$ ” and “ $p(3 <_{wc} 4)$ ”. For the NAC-dependencies we have the places “ $p(1, N[2])$ ” for the second instantiated NAC in the first derivation step of d and “ $p(3, N[1])$ ” for the third derivation step and its first instantiated NAC. The other two instantiated NACs are not considered, because the corresponding rules are weakly self-disabling ($q <_{wdn[i]} q$). At the beginning the transitions $cont1$ and $cont2$ are enabled. The firing sequences according to the derivations d and d' in Fig. 3 can be executed and they are the only firing sequences of this net. Thus, the net specifies exactly the derivations which are permutation-equivalent to d .

We now show by Thm. 1 below that we can exploit the constructed Petri net $DNet(d)$ for a derivation d to characterise all derivations that are permutation-equivalent to d , by analysing the firing behaviour of $DNet(d)$. Note that according to Def. 8 each sequence s of rule names in the STS $STS_N(d)$ can be interpreted as a sequence of transitions in the derived marked Petri net $DNet(d)$, and vice versa. This correspondence allows us to transfer the results of the analysis back to the STS. More precisely, we can generate the set of all permutation-equivalent sequences by constructing the reachability graph of $DNet(d)$, which therefore can be considered as a compact representation of this equivalence class.

Recall that a *transition complete firing sequence* of a Petri net is a firing sequence where each transition of the net occurs at least once; notice also that in a dependency net according to Def. 8, each transition can fire at most once by construction. This means in our case each transition fires exactly once. The following Thm. 1 presents a sound and complete analysis of permutation equivalence by complete firing sequences in the corresponding dependency net.

Theorem 1 (Analysis of Permutation Equivalence of Derivations) *Let d be a derivation respecting NACs of a GTS with NACs, and let $DNet(d)$ be its dependency net. Then a derivation d' is permutation equivalent to d ($d' \approx_{\pi} d$) if and only if the sequence of names $s_{d'}$, which contains all the direct derivations of d in the order they are actually fired in d' , is a transition complete firing sequence of the marked P/T Petri net $DNet(d)$.*

Proof (Sketch). Let d be a derivation with NACs, $STS_N(d)$ be its derived STS and $DNet(d)$ be the

constructed dependency net. We can interpret a transition complete firing sequence s of $DNet(d)$ within the STS $STS_N(d)$ and show that it corresponds to a valid derivation in $STS_N(d)$. This allows us to use Thm. 1 in [Her09] showing that the derivation derived from s is permutation-equivalent to d . Vice versa, given a derivation d' , which is permutation-equivalent to d , we can show that the corresponding sequence $s_{d'}$ is a transition complete firing sequence in $DNet(d)$. For a complete proof see Cor. 1 in [HCEK10]. \square

4 On the Cost of Analysis

Besides soundness and completeness of the analysis as presented in Thm. 1 we now focus on its efficiency. Therefore, we extend the previous example and compare the analysis efforts of the new technique with those of a direct analysis of the derivation. This comparison shows a significant advantage of the technique and the effect is not limited to specific examples. The benefit is high for transformation sequences, where many steps overlap on matches and include dependencies because of NACs. Clearly, if NACs are not involved permutation equivalence is equal to switch equivalence and in this case the reachability graph of the Petri net specifies all switch-equivalent derivations.

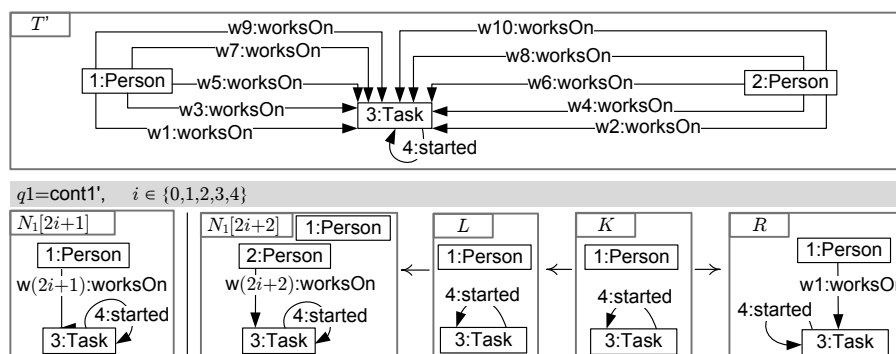


Figure 7: Part of the Derived STS $STS_N(\tilde{d})$

Example 5 (Extended Derivation) We extend the derivation d of Ex. 1 to a derivation \tilde{d} , which specifies that the two persons are working on the same task, but they continue and stop their work five times, i.e. $\tilde{d} = (d; d; d; d; d)$ is a derivation with 20 steps. The derived STS $STS(\tilde{d})$ contains 20 rule occurrences and Fig. 7 shows its super object T' and the rule occurrence “cont1” for the first step of \tilde{d} . This rule occurrence has 10 NACs, one for each possible edge of type “worksOn” in T' . These NACs are visualised in the figure by two NACs with a parameter i ranging from zero to four. The derivation consists of 10 blocks of the form “cont x ; stop x ”. Each permutation-equivalent derivation of \tilde{d} has to preserve these blocks, otherwise a NAC would not be fulfilled or the causality relation would be violated. Thus there are $10! = 3.628.800$ permutation-equivalent derivations.

Based on the dependency net $DNet(\tilde{d})$ we can construct the reachability graph $RG(DNet(\tilde{d}))$ for this marked Petri net with 20 transitions and 120 places. Each path in this graph specifies a permutation-equivalent derivation. An upper bound for the effort eff of constructing

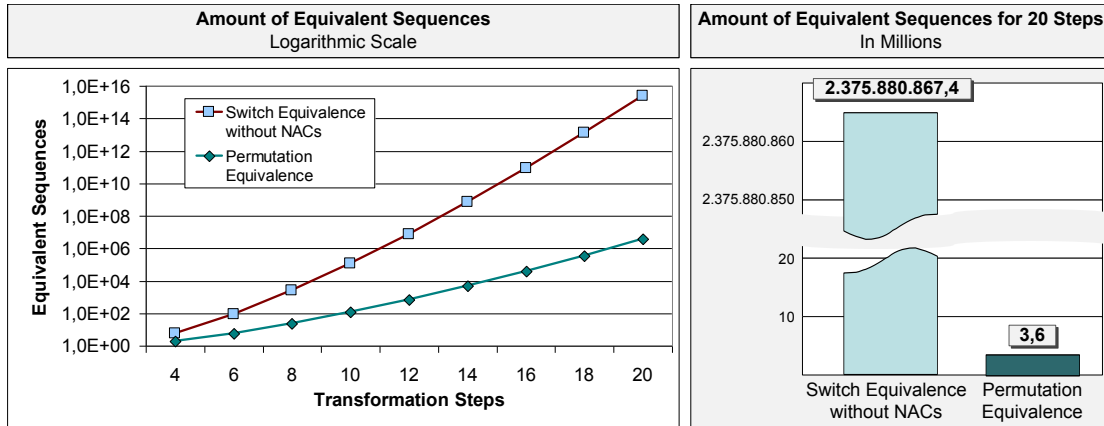


Figure 8: Comparison of the Amount of Equivalent Sequences

$RG(DNet(\tilde{d}))$ is given by: $eff < 9 \cdot n$, where n is $n = 20 \cdot 10! = 72.576.000$, which is the number of derivation steps in the set of all permutation-equivalent derivations. The details of these and the following numbers are given in [HCEK10]. A direct generation of the permutation-equivalent derivations based on Def. 3 (brute force method) starts with a computation of the complete set of switch-equivalent derivations disregarding the NACs and thereafter the invalid ones are filtered out. This would lead to $F = 654.729.075$ times as many derivations as the number of permutation-equivalent derivations, because the blocks “ $cont(x);stop(x)$ ” are split and many steps “ $cont(x)$ ” can be shifted backwards. Thus, the lower bound for the brute force effort EFF is given by $F \cdot n \leq EFF$. In comparison we have for the effort eff of constructing the reachability graph of the dependency net:

$$eff < 1,4 \times 10^{-8} EFF.$$

Fig. 8 shows how the different amounts of equivalent sequences develop for 2 up to 10 blocks of “continue;stop” steps. Both analyses are fairly brute-force, since we did not integrate reductions, such as symmetry or partial-order reduction. However, the figures show that a definite gain in efficiency can be obtained, which we expect similarly also with additional reductions, which are mainly orthogonal to the reduction technique studied in this paper.

Of course, the effort for constructing the Petri net has also to be taken into account, but it does not significantly change the result. In general, the construction of the STS $STS(d)$ with its relations is shown to be of polynomial time complexity with respect to the length of the derivation d [Her09]. Furthermore, the construction of the dependency net is linear with respect to $STS_N(d)$ equipped with the derived relations and for this example contains only 120 places. Note that still all steps in \tilde{d} are sequentially dependent with NACs and therefore, no direct switching is possible.

5 Conclusion

In the framework of adhesive high-level replacement (HLR) systems there are many instantiations, such as graph transformation systems scaling up to typed attributed graph transformation systems with node type inheritance, and Petri net transformation systems - in particular for the

modelling of workflows of reconfigurable mobile adhoc networks [EHP⁺07, HEP07]. Each of them has its specific features, which support the modelling of systems in the concrete application domain. Negative Application Conditions (NACs) are an important control structure for these techniques and they are widely used for applications. However, the analysis of processes of such systems, i.e. the study of equivalence of derivations in the presence of NACs going beyond switch equivalence including NACs as studied in [HHT96, LEO06], was introduced only recently in [Her09]. This new notion of equivalence, called permutation equivalence, is studied in this paper. More precisely, we study the problem how to obtain, in an efficient way, all derivations d' which are permutation-equivalent to a given derivation d .

In order to provide a sound, complete and efficient analysis technique for permutation equivalence we have shown how the dependency net for the derivation can be constructed, which purely specifies the dependencies between the transformation steps including the inhibiting effects of the NACs. Based on the reachability graph of the dependency net we derive all valid permutations of the derivation steps of a given derivation d , i.e. the order of the applied rules together with the new matches. The derived derivations are exactly the permutation-equivalent derivations of d . While the example in this paper was kept compact, the overall approach can be applied to adhesive HLR systems in general, if suitable side conditions are fulfilled [HE08], which is the case for e.g. typed attributed graph transformation systems.

The efficiency of the Petri net approach is based on two advantages. First of all, the constructed Petri net only specifies the dependencies among the steps of the derivation, ignoring the concrete structure of the involved graphs: This advantage is independent of the presence of NACs. The second advantage is that NACs are respected during the generation of the permutation-equivalent sequences. Thus, the number of generated sequences during the analysis is reduced significantly if NACs are involved, as shown by the presented example.

Some of the problems addressed in this paper are similar to those considered in the process semantics [KK04] and unfolding [Bal00, BKS04] of Petri nets with inhibitor arcs, and actually we could have used some sort of inhibitor arcs to model the inhibiting effect of NACs in the dependency net of a derivation. However, we would have needed some kind of “generalised” inhibitor nets, where a transition is connected to several (inhibiting) places and can fire if at least one of them is unmarked. To avoid the burden of introducing yet another model of nets, we preferred to stick to a direct encoding of the process of a derivation into a standard marked P/T nets, leaving as a topic for future research the possible use of different models of nets for our dependency net.

Future work will encompass the extension of the presented technique to general application conditions in the form of nested application conditions [HP05, HP09], for which we already have concrete ideas. Further improvements of efficiency could be obtained by observing the occurring symmetries in the P/T Petri net, and applying symmetry reduction techniques on it. Additionally, the space complexity of the analysis could be reduced by unfolding the net and then representing all permutation-equivalent derivations in a more compact, partially ordered structure. We already implemented the construction of a dependency net from a given graph transformation derivation with NACs based on a recently developed graph transformation engine in Mathematica called AGT (Algebraic Graph Transformation) [BHE09].

Acknowledgements: The research was supported by the DFG project Behaviour-GT.

References

- [Bal00] P. Baldan. *Modelling Concurrent Computations: from Contextual Petri Nets to Graph Grammars*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 2000. Available as technical report n. TD-1/00.
- [BCH⁺06] P. Baldan, A. Corradini, T. Heindel, B. König, P. Sobocinski. Processes for Adhesive Rewriting Systems. In *FoSSaCS'06*. LNCS 3921, pp. 202–216. Springer, 2006.
- [BHE09] C. Brandt, F. Hermann, T. Engel. Security and Consistency of IT and Business Models at Credit Suisse realized by Graph Constraints, Transformation and Integration using Algebraic Graph Theory. In *Proc. Int. Conf. on Exploring Modeling Methods in Systems Analysis and Design 2009 (EMMSAD'09)*. LNBIP 29, pp. 339–352. Springer Verlag, Heidelberg, 2009.
- [BKS04] P. Baldan, B. König, I. Stürmer. Generating Test Cases for Code Generators by Unfolding Graph Transformation Systems. In *Proc. of ICGT '04*. LNCS 3256, pp. 194–209. Springer, 2004.
- [CHS08] A. Corradini, F. Hermann, P. Sobociński. Subobject Transformation Systems. *Applied Categorical Structures* 16(3):389–419, June 2008.
- [EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theoretical Computer Science. Springer, 2006.
- [EHP⁺07] H. Ehrig, K. Hoffmann, J. Padberg, U. Prange, C. Ermel. Independence of Net Transformations and Token Firing in Reconfigurable Place/Transition Systems. In Kleijn and Yakovlev (eds.), *Proc. of ATPN'07*. LNCS 4546, pp. 104–123. Springer, 2007.
- [HCEK10] F. Hermann, A. Corradini, H. Ehrig, B. König. Efficient Process Analysis of Transformation Systems Based on Petri nets. Technical report 2010-3, Fak. IV, Technische Universität Berlin, 2010.
<http://www.eecs.tu-berlin.de/menue/forschung/forschungsberichte/2010>
- [HE08] F. Hermann, H. Ehrig. Process Definition using Subobject Transformation Systems. *EATCS Bulletin* 95:153–163, 2008.
- [HEP07] K. Hoffmann, H. Ehrig, J. Padberg. Flexible Modeling of Emergency Scenarios using Reconfigurable Systems. In *Proc. of IDPT'07*. Society for Design and Process Science, 2007.
- [Her09] F. Hermann. Permutation Equivalence of DPO Derivations with Negative Application Conditions based on Subobject Transformation Systems. In *Proc. of the Doctoral Symposium of ICGT'08. Electronic Communications of the EASST* 16, 2009.
- [HHT96] A. Habel, R. Heckel, G. Taentzer. Graph Grammars with Negative Application Conditions. *Fundamenta Informaticae* 26(3,4):287–313, 1996.

- [HP05] A. Habel, K.-H. Pennemann. Nested constraints and application conditions for high-level structures. In *Proc. of Formal Methods in Software and System Modeling*. LNCS 3393, pp. 293–308. Springer, 2005.
- [HP09] A. Habel, K.-H. Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science* 19(2):245–296, 2009.
- [KK04] H. C. M. Kleijn, M. Koutny. Process semantics of general inhibitor nets. *Information and Computation* 190(1):18–69, 2004.
- [Kre86] H.-J. Kreowski. Is parallelism already concurrency? Part 1: Derivations in graph grammars. In *Graph-Grammars and Their Application to Computer Science*. LNCS 291, pp. 343–360. Springer, 1986.
- [LEO06] L. Lambers, H. Ehrig, F. Orejas. Conflict Detection for Graph Transformation with Negative Application Conditions. In *ICGT'06*. LNCS 4178, pp. 61–76. Springer, 2006.
- [LS05] S. Lack, P. Sobociński. Adhesive and quasiadhesive categories. *Theoretical Informatics and Applications* 39(2):511–546, 2005.