Technische Universität Berlin

Forschungsberichte
der Fakultät IV – Elektrotechnik und Informatik

Efficient Process Analysis of Transformation Systems Based on Petri nets

Frank Hermann, Andrea Corradini, Hartmut Ehrig and
Barbara König

Bericht-Nr. 2010-03
ISSN 1436-9915

Efficient Process Analysis of Transformation Systems Based on Petri Nets

Frank Hermann[1], Andrea Corradini[2], Hartmut Ehrig[1] and Barbara König[3]

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

[2] andrea@di.unipi.it, Dipartimento di Informatica, Università di Pisa, Italy

[3] barbara_koenig@uni-due.de, Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen, Germany

Abstract

In the framework of adhesive transformation systems with Negative Application Conditions (NACs), we show how the problem of computing the set of equivalent derivations to a given one can be reduced to the analysis of the reachability graph of a generated Place/Transition Petri net. This net encodes the dependencies among rule applications of the derivation, including the inhibiting effects of the NACs. We show the effectiveness of this approach by comparing the cost of a brute force-approach with the cost of the presented analysis applied to a derivation of a simple system, showing a significant improvement in speed.

1 Introduction

Given a workflow of a system, it is often interesting to know whether the workflow can be rearranged, 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. We are here considering workflow models with two further dimensions, which considerably complicate the problem: first, we work in the general setting of adhesive categories where we can model systems with an evolving topology, such as graph transformation systems, in contrast to systems with a static structure like Petri nets. The analysis can be further extended to the class of weak adhesive categories using the results in [10]. Second, 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. Both dimensions are needed e.g. for the analysis of workflows in mobile ad-hoc networks.

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 [9] 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.

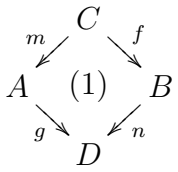
Finally, for a given derivation of a simple example system with NACs, we perform a detailed complexity analysis of the cost of identifying all permutation-equivalent derivations using the reduction to a Petri net and its reachability graph, and compare it with a brute force approach computing all shift-equivalent 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.

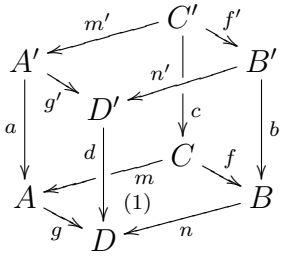
The structure of the paper is as follows. Sec. 2 reviews the main concepts of transformation systems over an adhesive category, permutation equivalence for derivations and the process construction based on subobject transformation systems, elaborating on the results of [9]. The construction of the process skeleton given by a Petri net is presented in Sec. 3 and it is shown to be sound and correct for computing the set of permutation-equivalent derivations. Thereafter, 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 Transformation Systems and Permutation Equivalence

Most definitions and results of the DPO approach, originally developed for graphs [7] and similar structures, have been generalized to *adhesive categories* [13, 6]: these are categories where pushouts along monos "behave well" with respect to pullbacks. Because of this, it is quite natural to present our contribution at this level of generality, by referring all definitions to an arbitrary but fixed adhesive category $\mathbf{C}$; the reader unfamiliar with adhesive categories can safely identify $\mathbf{C}$ with a standard category of graphs, like those used in the examples.

Definition 1 (Adhesive categories)**.** *A category is called* adhesive *if (1) it has pullbacks; (2) it has pushouts along monos; and (3) pushouts along monos are* Van Kampen squares.





(2)

Referring to the diagram above, a Van Kampen square is a pushout (1) which satisfies the following property: if we draw a commutative cube (2) which has (1) as its bottom face and whose back faces are pullbacks then the front faces of the cube are pullbacks if and only if its top face is a pushout.

For the rest of the section, let $\mathbf{C}$ be an arbitrary but fixed adhesive category: all objects and arrows are assumed to belong to $\mathbf{C}$. We recall some basic definitions of the DPO approach with NACs.

Definition 2 (Transformation System with NACs)**.** *A* rule *is a pair of monos with the same source in* $\mathbf{C}$, $p = (L_p \overset{l}{\hookleftarrow} K_p \overset{r}{\hookrightarrow} R_p)$. *A* Negative Application Condition (NAC) *for a rule* p *is a mono* $n : L_p \hookrightarrow N$, *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 an object* G *is a mono* $m : L_p \hookrightarrow G$; *match* m satisfies the NAC $n : L_p \hookrightarrow N$ for p, *written* $m \models n$, *if there is no arrow* $g : N \to G$ *such that* $g \circ n = m$.[1] *We say that there is a* direct derivation respecting NACs *from an object* G *to* H *using a rule with NACs* $\langle p, \mathbf{N} \rangle$ *and a match* $m : L_p \to G$, *if (a) there are two pushouts (1) and (2) in* $\mathbf{C}$, *as depicted; and (b)* $m \models n$ *for each NAC* $(n : L_p \hookrightarrow N) \in \mathbf{N}$. *If condition (a) above is satisfied (and (b) possibly not, thus NACs are ignored) we say that there is a* direct derivation *from* G *to* H. *In both cases we write* $G \overset{p,m}{\Longrightarrow} H$.

[1] Intuitively, the image of L_p in G cannot be extended to an image of the "forbidden context" N.

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

A transformation system (TS) with NACs over $\mathbf{C}$ *is a pair* $\mathcal{G} = \langle Q, \pi_N \rangle$ *where* Q *is a set of rule names, and* π_N *maps each name* $q \in Q$ *to a rule with NACs* $\pi_N(q) = \langle \pi(q), \mathbf{N}_q \rangle$. *A* derivation (respecting NACs) *of* $\mathcal{G}$ *is a sequence* $G_0 \overset{q_1,m_1}{\Longrightarrow} G_1 \cdots \overset{q_n,m_n}{\Longrightarrow} G_n$, *where* $q_1, \ldots, q_n \in Q$ *and* $d_i = G_{i-1} \overset{\pi(q_i),m_i}{\Longrightarrow} G_i$ *are direct derivations (respecting NACs) for* $i \in 1, \ldots, n$. *Sometimes we denote a derivation as a sequence* $d = d_1; \ldots; d_n$ *of direct derivations.*

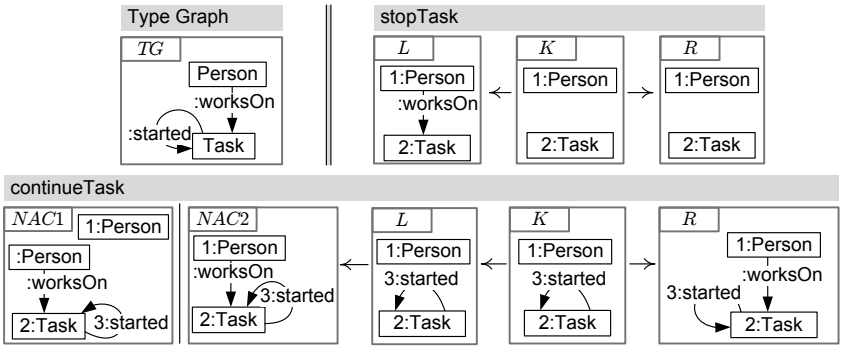


Figure 1: Part of transformation system *GS*, modeling task assignment

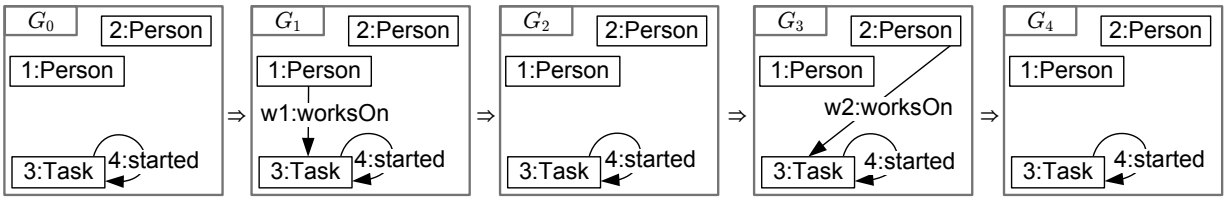


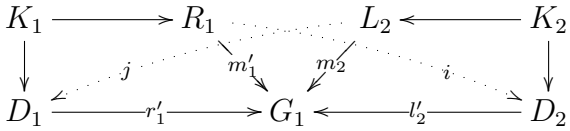
Figure 2: Derivation d (respecting NACs) of *GS*

Example 1 (Graph Transformation System with NACs)**.** *The adhesive category used in the examples of this paper is* (**Graph**$\downarrow TG$)*, i.e., the slice category of directed graphs over the graph TG of Fig. 1. Thus, objects are graphs with a* typing morphism *to TG, and arrows are graph morphisms preserving the typing. Fig. 1 shows a part of GS, a* TS *with NACs for specifying the assignment of tasks to persons, which is a little fragment of a workflow modeling system. The* type graph *TG shows that nodes in the system represent either persons or tasks: a task is active if it has a ":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. Other rules are omitted, because they are not*

used in the paper. Fig. 2 shows a derivation respecting NACs of GS. The only task is first assigned to node "1:Person", and then, after being stopped, to node "2:Person".

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. The *switch equivalence* is based on the notion of *sequential independence* and on the *Local Church-Rosser theorem*. This is briefly summarized in the next definition.

Definition 3 (Switch Equivalence on Derivations)**.** *Let* $d_1 = G_0 \overset{p_1,m_1}{\Longrightarrow} G_1$ *and* $d_2 = G_1 \overset{p_2,m_2}{\Longrightarrow} G_2$ *be two direct derivations. Then they are* sequentially independent *if there exist arrows* $i : R_1 \to D_2$ *and* $j : L_2 \to D_1$ *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 [6]) they can be "switched" obtaining direct derivations* $d'_2 = G_0 \overset{p_2,\overline{m_2}}{\Longrightarrow} G'_1$ *and* $d'_1 = G'_1 \overset{p_1,\overline{m_1}}{\Longrightarrow} G_2$*, which apply the two rules in the opposite order.*

Now, let $d = (d_1; \ldots; d_k;\ d_{k+1}; \ldots; d_n)$ *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.*[2]

Corresponding notions of parallel and sequential independence have been proposed for graph transformation systems with NACs [8, 14]. 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. 2 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 [9]. 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 4 (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. 3.*

[2]Informally, $d \cong d'$ if they have the same length and there are isomorphisms between the corresponding objects of d and d' compatible with the involved morphisms.

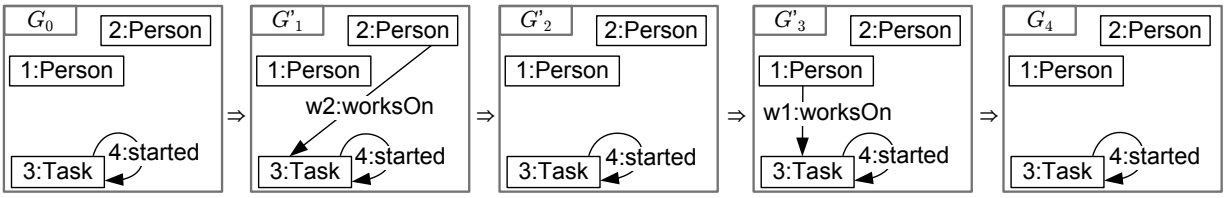


Figure 3: Permutation-Equivalent Derivation d' (respecting NACs) of GS

In the theory of Petri nets [15], from a given firing sequence one can build a *deterministic process*, which is a net which records all the transitions fired in the sequence, together with their causal dependencies. Similar constructions have been proposed for graph transformation [5] and for transformation systems based on adhesive categories [2, 4]. In particular, in [4] it is shown that starting with a derivation, with a suitable colimit construction in $\mathbf{C}$ one can build a *Subobject Transformation System (STS)*. An STS can be considered as a double-pushout transformation system over the (distributive) lattice of subobjects $\mathbf{Sub}(T)$ of a given object $T \in \mathbf{C}$. Informally, a subobject of T is an equivalence class of monos with target T, and in this framework rewriting can be defined with a set-theoretical notation: $A \subseteq B$ means that there is a mediating arrow from subobject A to B, the meet $A \cap B$ in $\mathbf{Sub}(T)$ is obtained as a pullback in $\mathbf{C}$, and the join $A \cup B$ is a suitable pushout (thanks to adhesivity of $\mathbf{C}$ [13]). We briefly recall the basic theory of STSs and then extend it to systems with NACs.

Definition 5 (STS of a derivation)**.** *A* subobject transformation system $\mathcal{S} = \langle T, Q, \pi \rangle$ *over an adhesive category* $\mathbf{C}$ *consists of a super object* $T \in \mathbf{C}$*, 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* $K_q = L_q \cap R_q$*.*

Given objects $G, H \in \mathbf{Sub}(T)$ *and a rule* $\pi(q) = \langle L_q, K_q, R_q \rangle$*, there is a* direct derivation $G \overset{q}{\Longrightarrow} H$ *if there exists an object* $D \in \mathbf{Sub}(T)$ *such that (a)* $G = L_q \cup D$*, (b)* $K_q = L_q \cap D$*, (c)* $H = R_q \cup D$*, and (d)* $K_q = R_q \cap D$*.*

Now, let $\mathcal{G} = \langle Q, \pi \rangle$ *be a transformation system over* $\mathbf{C}$*, and let* $d = (G_0 \overset{q_1,m_1}{\Longrightarrow} \ldots \overset{q_n,m_n}{\Longrightarrow} G_n)$ *be a derivation of* $\mathcal{G}$*. The* STS generated from d *is defined as* $Prc(d) = \langle T, P, \hat{\pi} \rangle$*, where* T *is the colimit object in* $\mathbf{C}$ *of the diagram underlying the derivation* d*,* $P = \{d_k \mid d_k = (G_{k-1} \overset{p_k,m_k}{\Longrightarrow} G_k)$ *is a step of* $d\}$*, and* $\hat{\pi}(d_k) = \langle [in_T(L_{p_k})], [in_T(K_{p_k})], [in_T(R_{p_k})] \rangle$*, where* $in_T(X)$ *is the injection of* X *in the colimit* T*.*

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. In the STS generated from a derivation d we can identify all derivations which are switch equivalent to d simply checking how the rules overlap in $Prc(d)$. We summarise some relevant facts presented in [4].

Definition 6 (Switch Equivalence of Derivations in STS). *Given an STS $\mathcal{S} = (T, Q, \pi)$, two rules $\pi(q_i) = \langle L_i, K_i, R_i \rangle$, $i \in \{1, 2\}$, are called* independent *(written $q_1 \Diamond q_2$) if $(L_1 \cup R_1) \cap (L_2 \cup R_2) \subseteq K_1 \cap K_2$. Let d be a derivation in $\mathcal{S}$ and let $s = \langle q_1, \ldots, q_n \rangle$ be its corresponding sequence of rule names. If $q_k \Diamond q_{k+1}$, 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 $s \overset{sw}{\sim}_S s'$. The switch equivalence $\overset{sw}{\approx}_{\mathcal{S}}$ on sequences of rule names is the reflexive and transitive closure of $\overset{sw}{\sim}_S$.*

Proposition 1 (Analysis of Switch Equivalence using $Prc(d)$). *Let $d = d_1; \ldots; d_k; d_{k+1}; \ldots; d_n$ be a derivation of a TS over $\mathbf{C}$, and let $Prc(d)$ be the generated STS. Then d_k and d_{k+1} are sequential independent if and only if $d_k \Diamond d_{k+1}$ in $Prc(d)$. As a consequence, a derivation d' is shift equivalent to d ($d' \overset{sw}{\approx} d$) if and only if in $Prc(d)$ the sequence of names of d, $s_d = \langle d_1, \ldots, d_n \rangle$ is shift equivalent to the sequence $s_{d'}$ ($s_d \overset{sw}{\approx}_{Prc(d)} s_{d'}$), which contains all the direct derivations of d in the order they are actually fired in d'.*

We discuss now how to extend this result to TSs with NACs, and how $Prc(d)$ can be used to identify the derivations which are permutation equivalent to d.

Definition 7 (STS with NACs). *Let $p = \langle L_p, K_p, R_p \rangle$ be a rule in $\mathbf{Sub}(T)$, with $T \in \mathbf{C}$. A* negative application condition *for p is an object $N \in \mathbf{Sub}(T)$ such that $L_p \subseteq N$. A* rule with NACs *is a pair $\langle p, \mathbf{N} \rangle$, where p is a rule and $\mathbf{N} = \langle N[1], N[2], \ldots, N[k] \rangle$ is an ordered list of NACs for p (we denote by $|\mathbf{N}|$ the length of $\mathbf{N}$). An* STS with NACs *is an STS $\mathcal{S} = \langle T, Q, \pi_N \rangle$ such that $\pi_N(q) = \langle \pi(q), \mathbf{N}_q \rangle$ is a rule with NACs. A direct derivation $G \overset{q}{\Longrightarrow} H$ as in Def. 5* respects the NACs $\mathbf{N}_q$ *if it holds: (e) for all $0 < i \leq |\mathbf{N}_q|$, $N[i] \not\subseteq G$.*

The generation of an STS with NACs from a given derivation works as in Definition 5, 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.

Definition 8 (Instantiated NACs). *Let $\mathcal{G}$ be a TS with NACs and let $d = d_1; \ldots; d_k; \ldots; d_n$ be a derivation respecting NACs. Let $\langle p, \mathbf{N} \rangle$ be the rule with NACs used in direct derivation d_k, let T be the colimit object of the derivation, and let $in_T(L_p)$ be the injection in T of the left-hand side of p. Let $n : L_p \hookrightarrow N$ be a NAC of p; an* instantiated NACs of n in T *is a subobject $[j : N \hookrightarrow T] \in \mathbf{Sub}(T)$ such that $j \circ n = in_T(L_p)$. The set of all instantiated NACs in T of all NACs of a rule p is denoted by $NACS_T(p)$.*

Definition 9 (STS of a Derivation with NACs). *Let $\mathcal{G}$ be a TS with NACs and let d be a derivation of $\mathcal{G}$ respecting NACs. The* STS with NACs generated by d *is given by $Prc_N(d) = \langle T, P, \hat{\pi}_N \rangle$, where T and P are as in Def. 5, $\hat{\pi}_N(d_k) = \langle \hat{\pi}(d_k), \mathbf{N}_k \rangle$, $\hat{\pi}(d_k)$ is as in Def. 5, and $\mathbf{N}_k$ is an arbitrary but fixed linearisation of $NACS_T(p_k)$, where p_k is the rule of $\mathcal{G}$ used in d_k.*

Example 2 (Derived Process $Prc(d)$). *For the derivation in Ex. 1 the process construction leads to the STS as shown in Fig. 4. The derivation d involves the rules "continueTask" and "stopTask" and thus, the derived STS contains the rule occurrences "cont1", "cont2", "stop1" and "stop2", where the NACs of the rule "continueTask" are instantiated.*

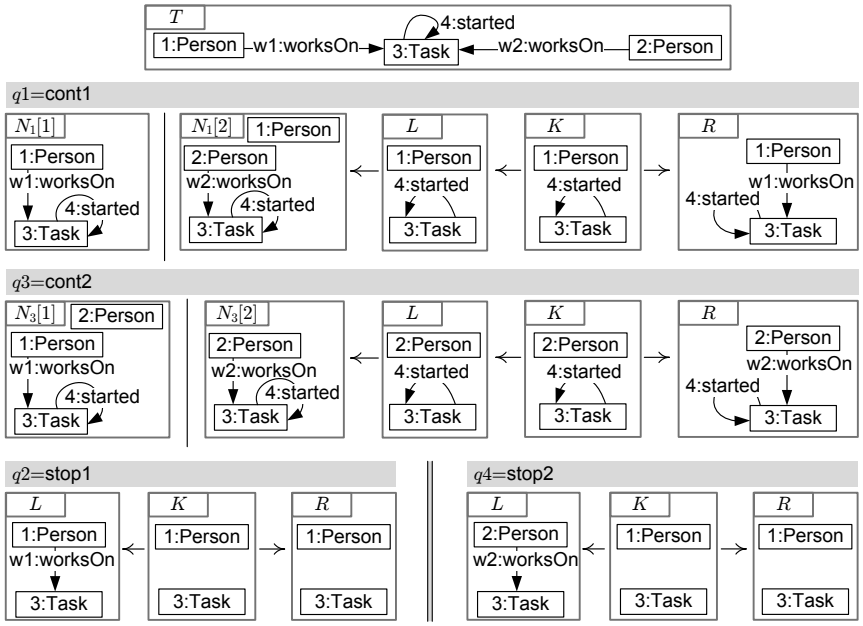


Figure 4: Derived Process $Prc(d)$ as Subobject Transformation System

The following relations between the rules of an STS with NACs specify the possible dependencies among them: the first four relations are discussed in [4], while the last two are introduced in [9].

Definition 10 (Relations on Rules)**.** *Let q_1 and q_2 be two rules in an STS with NACs $\mathcal{S} = (T, P, \pi_N)$ with $\pi_N(q_i) = (\langle L_i, K_i, R_i \rangle, \mathbf{N}_i)$ for $i \in \{1, 2\}$. 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	$q_1 \lozenge q_2$	$(L_1 \cup R_1) \cap (L_2 \cup R_2) \subseteq K_1 \cap K_2$
Weak NAC Enabling	$q_1 <_{wen[i]} q_2$	$0 < i \leq \lvert\mathbf{N}_2\rvert \ \wedge \ L_1 \cap N_2[i] \not\subseteq K_1 \cup L_2$
Weak NAC Disabling	$q_1 <_{wdn[i]} q_2$	$0 < i \leq \lvert\mathbf{N}_1\rvert \ \wedge \ N_1[i] \cap R_2 \not\subseteq L_1 \cup K_2$

In words, $q_1 <_{wen[i]} q_2$ (read: "q_1 weakly enables q_2 at i") if q_1 deletes a piece of the i-th NAC of q_2; instead $q_1 <_{wdn[i]} q_2$ ("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 of an STS)**.** *The rules of Prc(d) in Fig. 4 are related by the following dependencies. For write causality we have "cont1* $<_{wc}$ *stop1" and "cont2* $<_{wc}$ *stop2". The further dependencies are shown in in the table below.*

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

The following notion of legal sequences of rule names builds the basis for the analysis of permutation equivalence of derivations with NACs within the constructed STS. It requires that for every NAC $N[i]$ of a rule q_k of $Prc(d)$, either there is a rule which *deletes* part of $N[i]$ and was fired *before* q_k, or there is a rule which *produces* part of $N[i]$ and is fired *after* q_k: In both cases $N[i]$ cannot be present when firing q_k, because the STS $Prc(d)$ is a sort of "unfolding" of the derivation, and every subobject is created at most once and deleted at most once (see [4]).

Definition 11 (Legal Sequence)**.** *Let* $d = (d_1; \ldots; d_n)$ *be a derivation respecting NACs in a* TS*, and let* $Prc(d) = (T, P, \pi_N)$ *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. $s \overset{sw}{\approx}_{Prc(d)} seq(d)$
2. $\forall$ *NAC* $N_k[i]$ *of* q_k : $\left(\begin{array}{rl} \exists & e \in \{1, \ldots, k-1\} : \quad q_e <_{wen[i]} q_k \text{ or} \\ \exists & l \in \{k, \ldots, n\} : \quad q_k <_{wdn[i]} q_l. \end{array} \right)$

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

Theorem 1 (Analysis of Permutation Equivalence using $Prc(d)$)**.** *Let* $d = d_1; \ldots; d_n$ *be a derivation respecting NACs of a TS with NACs over* $\mathbf{C}$*, and let* $Prc(d)$ *be the generated STS with NACs. Then a derivation* d' *is permutation equivalent to* d ($d' \overset{\pi}{\approx} d$) *if and only if in* $Prc(d)$ *the sequence of names* $s_{d'}$*, which contains all the direct derivations of* d *in the order they are actually fired in* d'*, is legal with respect to* d*.*

By Thm. 1 we can transfer the analysis of permutation equivalence from derivations to sequences of rule names. Thus, given a derivation d respecting NACs we construct the process model $Prc(d)$ according to Def. 9, compute the relations for the STS, and then generate the legal sequences w.r.t. d according to Def. 11, which identify the derivations permutation equivalent to d.

3 Construction of the Process Skeleton

Based on the process of a derivation given by an STS, we now present the construction of its "process skeleton", given by a P/T Petri net which specifies only the dependencies between the derivation steps. All details about the internal structure of the objects and the transformation rules are excluded, allowing us to further increase the efficiency of the analysis of permutation equivalence.

Definition 12 (Process skeleton $\underline{Prc}$ of a derivation)**.** *Let d be a derivation respecting NACs of a TS with NACs over* **C***, let $Prc(d)$ be the generated STS with NACs and let $s = seq(d) = \langle q_1, \ldots, q_n \rangle$ denote the sequence of rule names in $Prc(d)$ according to the steps in d. The process skeleton of d is given by the marked Petri net $\underline{Prc}(d) = \langle N, M \rangle$, $N = \langle PL, TR, pre, post \rangle$, defined as follows:*

- $TR = \{q_k \mid k \in \{1, \ldots, n\}\}$
- $PL = \{p(k) \mid q_k \in TR\} \cup \{p(j <_x k) \mid q_j <_x q_k \wedge x \in \{rc, wc, d\}\}$
  $\cup \{p(k, N[i]) \mid N_k[i] \text{ is a NAC of } q_k \text{ in } Prc(d) \wedge q_k \not<_{wdn[i]} q_k\}$
- $pre(q_k) = p(k) \oplus \sum_{\substack{q_j <_x q_k \\ x \in \{rc,wc,d\}}} p(j <_x k) \oplus \sum_{\substack{q_j <_{wdn[i]} q_k \\ j \neq k}} p(j, N[i]) \oplus \sum_{p(k,N[i]) \in PL} p(k, N[i])$
- $post(q_k) = \sum_{\substack{q_k <_x q_l \\ x \in \{rc,wc,d\}}} p(k <_x l) \oplus \sum_{q_k <_{wen[i]} q_l} p(l, N[i]) \oplus \sum_{p(k,N[i]) \in PL} p(k, N[i])$
- $M = \sum_{q_k \in TR} p(k) \oplus \sum_{\substack{q_j <_{wdn[i]} q_k \\ p(j,N[i]) \in PL}} p(j, N[i])$

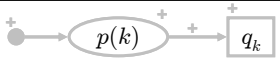
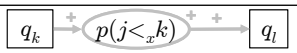
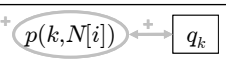
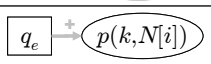
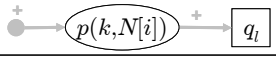
$Prc(d) = (S,T,P,\pi)$		$\underline{Prc}(d) = ((PL,TR,pre,post),M)$
1. For each $q_k \in P$		
2. For all $q_k <_x q_l$, $x \in \{rc,wc,d\}$		
3. For all i, q_k with $q_k \not<_{wdn[i]} q_k$		
	a) N[i] of q_k	
	b) For all $q_e <_{wen[i]} q_k$	
	c) For all $q_k <_{wdn[i]} q_l$	

Figure 5: Visualization of the Construction of the Petri net

Fig. 5 presents an intuitive view of the construction in Def. 12. Gray line colour and plus-signs mark the inserted elements. The tokens of the initial marking are represented by bullets that are connected to their places by arcs. In the first step each rule is encoded as a transition and it 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 $q <_{wen[i]} p$, i.e. transition q consumes part of the NAC $N[i]$ of p, then by step (3b) q produces a token in the place corresponding to $N[i]$. Symmetrically, if $q <_{wdn[i]} p$, i.e. p produces part of NAC $N[i]$ of q, then by step (3c) p consumes a token from the place corresponding to $N[i]$. Notice that if a rule generates part of one of its NACs, say $N[i]$ ($q_k <_{wdn[i]} q_k$), then by the acyclicity of $Prc(d)$ the NAC $N[i]$ cannot be completed before the firing of q_k: therefore we ignore it in the third step of the construction of the process skeleton.

Note that the constructed net is a true (bounded) P/T net, and not a safe one, because the places for the NACs can contain several tokens. A bound 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.

We now show that we can exploit the constructed Petri net $\underline{Prc}(d)$ to characterize the derivations that are permutation equivalent to d, by analysing its firing behaviour. Note that according to Def. 12 each sequence s of rule names in the STS $Prc(d)$ can be interpreted as a sequence of transitions in the derived marked Petri net $\underline{Prc}(d)$, 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 $\underline{Prc}(d)$, which therefore can be considered as a compact representation of this equivalence class.

For the following theorem, 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 process skeleton according to Def. 12, each transition can fire at most once by construction.

Theorem 2 (Analysis based on Petri Nets)**.** *Let d be a derivation respecting NACs of a TS with NACs over* $\mathbf{C}$*, then:*

$$s \overset{\pi}{\approx}_{Prc(d)} seq(d) \textbf{ iff } s \text{ is a transition complete firing sequence of } \underline{Prc}(d),$$

i.e. s is a legal sequence with respect to d iff s is a firing sequence of the skeleton process of d given by the marked P/T Petri net $\underline{Prc}(d)$ and each transition occurs at least once in s.

In order to prove Thm. 2 we use Lemma 1 below, which we prove first. The lemma states that switch equivalence without NACs of rule sequences in an STS respects the partial order of the relations "$<_{rc}, <_{wc}$" and "$<_d$", and vice versa: if the order is respected then the two sequences are switch equivalent.

Lemma 1 (Linearisation)**.** *Let d be a derivation respecting NACs of a TS with NACs over* **C***, let $Prc(d)$ be the generated STS with NACs, and let $s = \langle\, q_1, \ldots, q_n \,\rangle$ be a permutation of $seq(d)$. Then,*

$$s \overset{sw}{\approx}_{\mathcal{S}} seq(d) \textit{ iff } \forall\, i,j \in \{1,\ldots,n\},\, x \in \{rc, wc, d\} : q_i <_x q_j \Rightarrow i < j.$$

Proof. Let $(*) : \forall\, i,j \in \{1,\ldots,n\},\, x \in \{rc, wc, d\} : q_i <_x q_j \Rightarrow i < j$.
Direction "⇒"
Let $s \overset{sw}{\approx}_{\mathcal{S}} seq(d)$ and $seq(d) = \langle\, q'_1, \ldots, q'_n \,\rangle$.
We show that $(*)$ holds.

- We first show the property for $s = seq(d)$, i.e.
  $(**) : \forall\, i,j \in \{1,\ldots,n\}, x \in \{rc, wc, d\} : q'_i <_x q'_j \Rightarrow i < j$.
  $\Leftrightarrow \forall\, i,j \in \{1,\ldots,n\}, x \in \{rc, wc, d\} : i \geq j \Rightarrow q'_i \not<_x q'_j$.
  Let $\pi(q'_i) = (\langle L_i, K_i, R_i \rangle, N_i)$ and $\pi(q'_j) = (\langle L_j, K_j, R_j \rangle, N_j)$.
  For $i = j$ the condition is fulfilled directly.

  Now, consider $i > j$.

  - Case $x = rc$:
    By definition we have that $q'_i \not<_{rc} q'_j \Leftrightarrow R_i \cap K_j \subseteq K_i$.

    We can build up the colimit of the derivation d by stepwise pushouts. Let T_{i-1} be the colimit of the steps $d_1, \ldots, d_{i-1}$. Then we have that $(1) : K_j \subseteq T_{i-1}$. Let T'_i be the colimit of the single derivation step d_i, and therefore, T'_i is given by the pushout (2) of $G_i \leftarrow D_i \rightarrow G_{i+1}$. We perform a pushout (3) of T_{i-1} and T'_i and obtain T_i. Now consider the category $\mathbf{Sub}(T_i)$. We compose the pushouts (2) and (3) with the pushout $(4) : D_i \leftarrow K_i \rightarrow R_i \rightarrow G_{i+1}$ of the derivation step d_i. This is also a pullback and thus, $R_i \cap T_{i-1} \cong K_i$. Using (1) this implies $R_i \cap K_j \subseteq K_i$.
  - Case $x = wc$:
    By definition we have that $q'_i \not<_{wc} q'_j \Leftrightarrow R_i \cap L_j \subseteq K_i \cup K_j$.

    Considering the construction from before, we additionally derive $L_j \subseteq T_{i-1}$ and thus, the equation holds.
  - Case $x = d$:
    By definition we have that $q'_i \not<_{wc} q'_j \Leftrightarrow K_i \cap L_j \subseteq K_j$.

    Considering the construction from before, we can additionally compose the pushout $(5) : D_j \leftarrow K_j \rightarrow L_j \rightarrow G_j$ of the derivation step d_j with the pushouts of the stepwise construction of T_{i-1} and finally derive $L_j \cap T_{i-1} \cong K_j$. Furthermore, we have $K_i \subseteq T_{i-1}$ from (1) and thus, the above equation holds.

- We now show that the condition $(*)$ holds for every sequence s that is switch-equivalent to $seq(d)$ without considering the NACs. By $(**)$ we know that the condition holds for $seq(d)$. Furthermore, each sequence s is derived from $seq(d)$ by switchings according to $\overset{sw}{\approx}_{\mathcal{S}}$. It remains to show that each switching preserves the condition $(*)$. Now, switch equivalence of sequences $\overset{sw}{\approx}_{\mathcal{S}}$ is based on $(q_i \Diamond q_j)$, which is equivalent to $(q_i \not<_{rc} q_j \wedge q_i \not<_{wc} q_j \wedge q_i \not<_{d} q_j)$ according to Thm. 32.2 in [4]. Thus, the condition is not affected by any switching.

Direction "$\Leftarrow$":
We have to show that the condition $(*)$ for the sequence s implies $s \overset{sw}{\approx}_{\mathcal{S}} seq(d)$. This is equivalent to $\neg(s \overset{sw}{\approx}_{\mathcal{S}} seq(d)) \Rightarrow \neg(*)$. Since s is a permutation of $seq(d)$ the condition $\neg(s \overset{sw}{\approx}_{\mathcal{S}} seq(d))$ means that s can be derived by switching neighbouring steps of $seq(d)$, where at least on switching is performed on a pair $(q_i; q_j)$ of steps that is dependent, i.e. $\neg(q_i \Diamond q_j)$, which is equivalent to $(q_i \not<_x q_j)$ for one $x \in \{rc, wc, d\}$ according to Thm. 32.2 in [4]. Thus, this pair would violate the condition $(*)$ in the new order. Since s is assumed to be not switching equivalent to $seq(d)$ based on $\overset{sw}{\approx}_{\mathcal{S}}$ there is at least one such pair, where the final position of q_j is in front of q_i in s.

□

Using Lemma 1 we now prove Thm. 2.

Proof of Thm. 2. Let $seq(d) = \langle\, q_1, \ldots, q_n \,\rangle$ and $s = \langle\, \hat{q}_1, \ldots, \hat{q}_n \,\rangle$.
Direction "$\Rightarrow$":
Assume that s is a legal sequence with respect to d in $Prc(d)$. We have to show that s is a transition complete firing sequence of $\underline{Prc}(d)$. First of all, each transition occurs exactly once, because s is a permutation of $seq(d)$ in $Prc(d)$. Consider the rule name $\hat{q}_m = tr$ in s, thus the claimed firing step $M_a \xrightarrow{tr} M_b$ with $tr = q_a$. We check the activation of tr in M_a, i.e. $M_a \geq pre(tr)$ according to Def. 12. Now, let $pre(tr) = \sum_{pl \in PL} \lambda_{pl} \cdot pl$. For each pl we have:

1. **case** $pl = p(k)$**:**

   this implies that $k = a$ and $\lambda_{pl} = 1$. By definition this place is initially marked with one token and there is no other transition connected to this place. Since each transition occurs exactly once in s this token is available in M_a.

2. **case** $pl = p(j <_x k), x \in \{rc, wc, d\}$**:**

   this implies that $a = k$ and $\lambda_{pl} = 1$, thus $tr = q_k$ and $q_j <_x q_k$. By Def. 12 we then have $post(q_j) \geq pl$ and pl is not in the pre domain of any other transition than $tr = q_k$. By Lemma 1 we have that q_j occurs before q_k in s and thus, $M_a \geq pl$.

3. **case** $pl = p(j, N[i])$**:**

   By Def. 12, the marking $M \geq m \cdot pl$ with m being the amount of weak disabling causes, i.e. $m = |DC|, DC = \{q'_k \mid q_j <_{wdn[i]} q_{k'}\}$.

(a) **case** $j \neq a$**:**
By Def. 12 we have that $\lambda_{pl} = 1$, $q_j <_{wdn[i]} q_k$ and $a = k$. The only transition tr' in $TR \setminus DC$ with $pre(tr') \geq pl$ is q_j and q_j consumes and produces one token.
Each of the transitions in DC consumes exactly one token and in sum they consume exactly m tokens. Therefore, $M_a \geq pl$, because q_k has not fired at this point.

(b) **case** $j = a$**:** Thus, $\lambda_{pl} = 1$. By Def. 11 there is one preceding rule occurrence q in s with $q <_{wen[i]} \hat{q}_j$ or there is one subsequent rule occurrence q in s with $\hat{q}_j <_{wdn[i]} q$. This means that for the first case $M_a \geq m \cdot pl + 1 - m \cdot pl = pl$ and for the second case: $M_a \geq mpl - (m-1)pl = pl$.

Direction "$\Leftarrow$":
Assume that s is a transition complete firing sequence of $\underline{Prc}(d)$. We have to show that s is a legal sequence with respect to d in $Prc(d)$. First of all, s is a transition complete firing and for each transition q_k the initial marking M contains exactly one token for the corresponding place $p(k) \in PL$ and q_k is consuming exactly one token from $p(k)$. Therefore, each rule name q_k in $seq(d)$ occurs exactly once in s. Now, we consider an arbitrary rule name q_k in $seq(d)$. We show that the two conditions in Def. 11 hold:

- condition 1: $s \overset{sw}{\approx}_{\mathcal{S}} seq(d)$
By Lemma 1 this condition is equivalent to
$(*)$: $\forall\ i,j \in \{1,\ldots,n\},\ x \in \{rc, wc, d\}$: $q_i <_x q_j \Rightarrow i < j$. According to Def. 12 there is exactly one place with initially no token for each pair (q_i, q_j) with $q_i <_x q_j, x \in \{rc, wc, d\}$. The transition q_i produces exactly one token and q_j consumes exactly one token from this place and there is no other transition connected to this place. Therefore, the condition is ensured.

- condition 2: $\forall$ NACs $N_k[i]$ of q_k : $\left(\begin{array}{rl} \exists & e \in \{1,\ldots,k-1\} : \quad q_e <_{wen[i]} q_k \text{ or} \\ \exists & l \in \{k,\ldots,n\} : \quad q_k <_{wdn[i]} q_l. \end{array} \right)$

Consider a NAC $N[i]$ of q_k.

1. case $q_k <_{wdn[i]} q_k$: Thus, we have a $l = k$ for the above condition.
2. case $q_k \not<_{wdn[i]} q_k$:
Thus, there is the place $p(k, N[i])$, such that the transition q_k consumes exactly one token from that place. Consider the firing step $M_k \xrightarrow{q_k} M_{k+1}$ according to s. Since q_k has fired there was a token on $p(k, N[i])$ in the marking M_k. The initial marking contains m tokens for this place, where m is the amount of weak disabling causes, i.e. $m = |DC|, DC = \{q_{k'} \mid q_k <_{wdn[i]} q_{k'}, k \neq k'\}$. Let $EC = \{q_e \mid q_e <_{wen[i]} q_k\}$ be the set of weak enabling causes of of q_k for $N_k[i]$. Assume that condition 2 of Def. 11 does not hold. We then have that all $q_{k'}$ in DC occur before q_k in s and there is no q_e in EC that occurs before q_k in s.

This implies that each transition of DC has consumed a token from $p(k, N[i])$ and none of the transitions that precede q_k have produced a token on this place. Therefore, there is no token left on $p(k, N[i])$, which is a contradiction to the firing of q_k and thus, condition 2 holds.

□

Coming back to the original challenge of computing the set of all permutation-equivalent derivations for a given one, we can now state by the following corollary that the analysis can be completely performed on the process skeleton $\underline{Prc}(d)$.

Corollary 1 (Analysis of Permutation Equivalence of Derivations)**.** *Let d be a derivation respecting NACs of a TS with NACs over $\mathbf{C}$, and let $\underline{Prc}(d)$ be its process skeleton. Then a derivation d' is permutation equivalent to d ($d' \overset{\pi}{\approx} d$) 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 $\underline{Prc}(d)$.*

Proof. This is a direct consequence of Thms. 1 and 2. □

Example 4 (Process Skeleton)**.** *Consider the derivation d from Ex. 1 and its derived STS in Ex. 2. The marked Petri net shown in Fig. 6 is the process skeleton $\underline{Prc}(d)$ according to Def. 12. As expected, there is a one-to-one correspondence between its firing sequences and the set of permutation-equivalent derivations of d. At the beginning the transitions cont1 and cont2 are enabled. The firing sequences according to the derivations d and d' of Figures 2 and 3 can be executed, and are the only firing sequences of this net.*

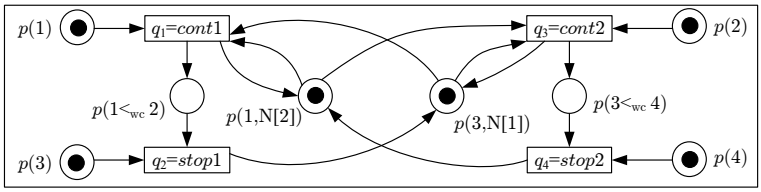


Figure 6: Process Skeleton $\underline{Prc}(d)$ as Petri net

4 On the Cost of Analysis

Besides soundness and completeness of the analysis as presented before 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.

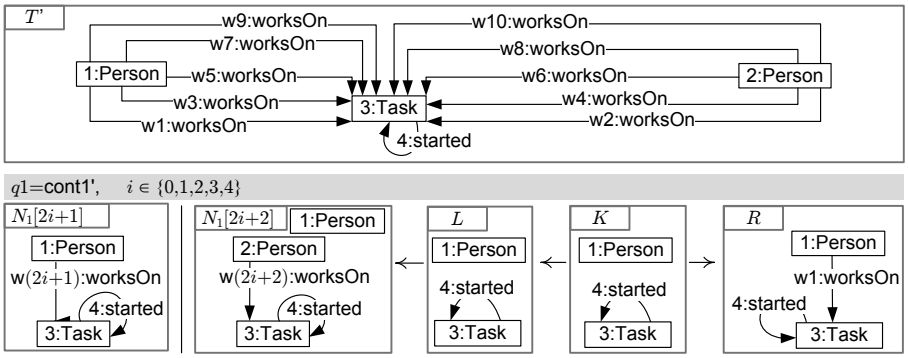


Figure 7: Part of the Dervied Process $Prc(\widetilde{d})$

Example 5 (Extended Derivation)**.** *In order to evaluate the efficiency we extend the derivation of Ex. 1 to a derivation $\widetilde{d}$, which specifies that the two persons are working on the same task, but they continue and stop their work five times, i.e. $\widetilde{d} = (d; d; d; d; d)$. The construction of $Prc(\widetilde{d})$ leads to an STS with 20 rule occurrences. Fig. 7 shows its super object T' and the rule occurrence "cont1' " for the first step of $\widetilde{d}$. 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 "contx; stopx". Each permutation-equivalent derivation of $\widetilde{d}$ 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.*

Let us consider to perform a direct analysis based on the definition of permutation equivalence. We call this the brute force variant, where first all switch-equivalent derivations are generated without considering the NACs, and then those which do not respect the NACs are filtered out. This means that we only have to respect the causality between the first and the second step of each of the 10 blocks. Therefore, we can always switch neighbouring steps of different blocks. For each permutation-equivalent sequence we can move the rule occurrences of the rule "stop" forward, i.e. at later positions. Therefore, we have $F = 19 \times 17 \times \cdots \times 1 = 654.729.075$ switch-equivalent sequences for each permutation in the order of the rule occurrences "cont$x$", i.e. for each single permutation-equivalent sequence. This leads to a number of $20!/2^{10} = 2.375.880.867.360.000 \approx 2,4 \times 10^{15}$ switch equivalent sequences.

Fig. 8 shows how the different amounts of equivalent sequences develop for 2 up to 10 blocks of "continue;stop" steps. Since the complexity of a function, which is dominated by a factorial expression, is super-exponential, the calculation of invalid sequences should be avoided in general. In Fig. 9 we present the reachability graph of the dependency net for an analogous transformation sequence with 10 steps, i.e. 5 "continue;stop" blocks. The graph was generated using our implementation based on Mathematica. As presented before, the reachability graph for the complete sequence contains $10! = 3.628.800$ leaf nodes and would be far to big for normal paper dimensions. The figure already shows the symmetric nature

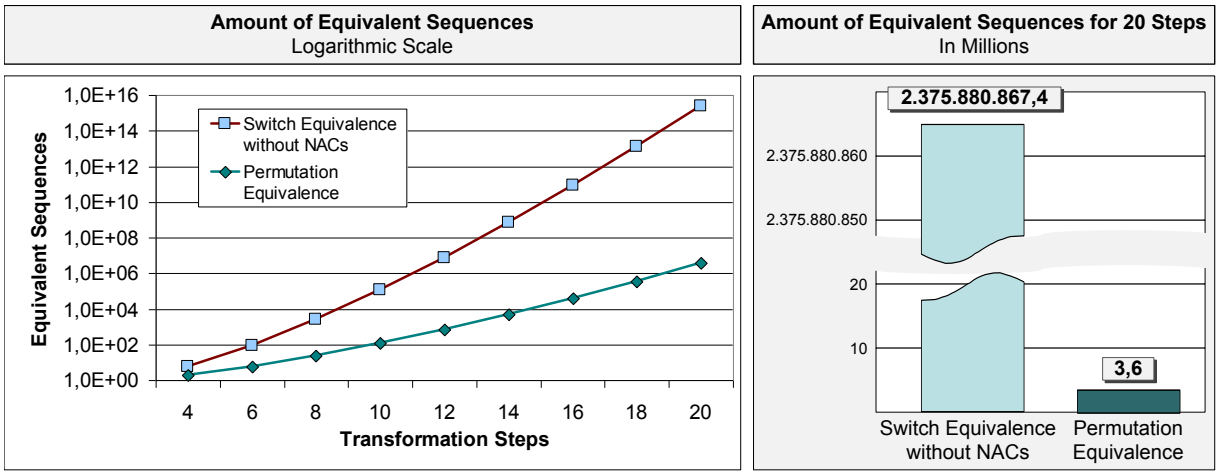


Figure 8: Comparison of the Amount of Equivalent Sequences

of the specific example and future work will encompass symmetry reduction techniques, such that the size can be reduced for examples with symmetry.

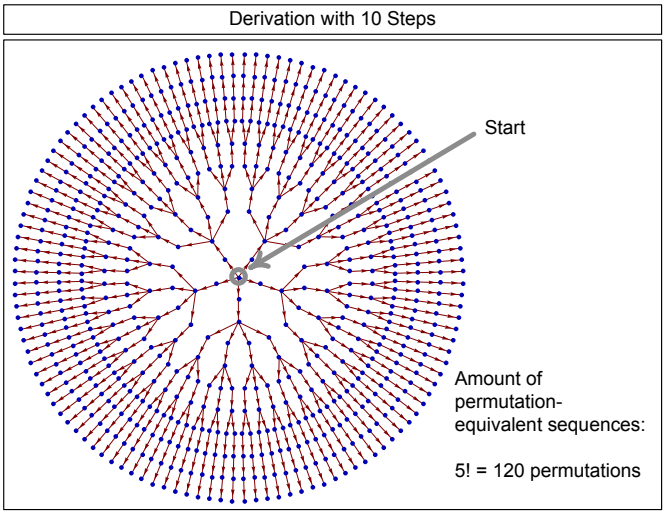


Figure 9: Reachability Graph of the Dependency Net for 10 Steps

Obviously, the generation of the permutation-equivalent sequences involves several computation steps. But let us compare the effort of generating the set of permutation-equivalent derivations using the process skeleton $\underline{Prc}(d)$ with the effort of a brute force generation directly based on Def. 4. The result is that a lower bound for the effort of the brute force variant is 8 orders of magnitude higher than an upper bound for the analysis based on the process skeleton.

Example 6 (Analysis Efforts)**.** *Based on the process skeleton* $\underline{Prc}(\widetilde{d})$ *we can construct the reachability graph* $RG(\underline{Prc}(\widetilde{d}))$ *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 $RG(\underline{Prc}(\widetilde{d}))$ is given by: $eff \leq 651.030.320 < 9 \cdot n$, 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. Considering the brute force variant we will construct $F = 654.729.075$ times as many derivations as the number of permutation-equivalent derivations. Thus, the lower bound for the brute force effort EFF is given by $F \cdot n \leq EFF$. In comparison we have:

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

Details for the Efficiency Results

- Size of the Petri net $\underline{Prc}(\widetilde{d}) = (PN, M), PN = (PL, TR, pre, post)$:
  $|TR| = 20$
  $|PL| = 20 + 10 + 10 \cdot 9 = 120$
  Amount of arcs $= 20 + 2 \cdot 10 + (1 + 2 + 1) \cdot 10 \cdot 9 = 400$
  Amount of all elements: 540

- Size of $RG(\underline{Prc}(\widetilde{d})) = (V, E, s, t)$:
  Branching number for the successors: root node: 10 (for each "cont$x$"), then 1 ("stop$x$"), then 9 , ... , then 1, then 1
  $|V| = 1 + 2 \cdot (\sum_{i=0..10} 10!/(10 - i)!) = 19.728.201$
  $|E| = |V| - 1 = 19.728.200$

- Effort for calculating $RG(\underline{Prc}(\widetilde{d}))$:
  Store the transitions that have fired, since the maximum of one time firing is ensured by definition
  At each node: check each transition that has not fired for activation, i.e. for the "cont$x$" transitions $1 + 9 = 10$ pre arcs and for the "stop$x$" transitions $1 + 1$ pre arcs. If one place in the pre domain is found empty, the remaining ones do not have to be checked. Then, continue and update the marking (1+9+1=11 for "cont$x$" and 1+1+9=11 for "stop$x$").
  Effort eff measured in binary operations: $eff \leq 651.030.320 < 33 \cdot |V| < 9 \cdot n$

Of course, the effort for constructing the Petri net has also to be taken into account, but it does not change the result. In general, the construction of the process $Prc(d)$ with its relations is shown to be of polynomial time complexity with respect to the length of the derivation d [9]. Furthermore, the construction of the process skeleton is linear with respect to $Prc(d)$ with its dependency relations and in this this example there are only 120 places in the constructed Petri net. Note that still all steps in $\widetilde{d}$ are sequentially dependent with the 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 system - in particular for the modelling of workflows of reconfigurable mobile adhoc networks. 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, was introduced only recently in [9].

While switch-equivalence [2] for systems without NACs leads to the complete set of equivalent derivations, this is not the case in the presence of NACs if the notion of equivalence proposed in [8, 14] is considered. Similarly, the notion of shift equivalence [16, 12] for derivations cannot be extended appropriately to the case with NACs, because it is also based on sequential independence of neighbouring steps. The problem is that rule applications may be possible in an equivalent way at several positions of the derivation, which are not situated next to each other, as shown with the presented example.

In order to provide a sound, complete and efficient analysis technique for permutation equivalence we have shown how the generated process given by an STS [4, 9] can be transformed to a process skeleton given by a marked P/T Petri net. The construction is shown to be sound and complete with respect to the computation of the set of permutation-equivalent derivations to a given one. Furthermore, the constructed Petri net shows significant advantages with respect to efficiency. 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 [10].

The efficiency of the 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 objects: 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. The construction of the Petri net can be performed in polynomial time with respect to the size of the initial derivation [9], and thus it does not affect the efficiency of the analysis.

Some of the problems addressed in this paper are similar to those considered in the process semantics [11] and unfolding [1, 3] 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 process skeleton 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 process skeletons.

Future work will also include the study of non-deterministic processes of transformation systems with NACs, which will be based on incomplete firings of the constructed P/T Petri net and suitable side conditions. 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. An implementation of the analysis is planned and will be based on a recently developed graph transformation engine in Mathematica.

References

[1] 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.

[2] P. Baldan, A. Corradini, T. Heindel, B. König, and P. Sobocinski. Processes for Adhesive Rewriting Systems. In *FoSSaCS'06*, volume 3921 of *LNCS*, pages 202–216. Springer, 2006.

[3] P. Baldan, B. König, and I. Stürmer. Generating Test Cases for Code Generators by Unfolding Graph Transformation Systems. In *Proc. of ICGT '04*, volume 3256 of *LNCS*, pages 194–209. Springer, 2004.

[4] A. Corradini, F. Hermann, and P. Sobociński. Subobject Transformation Systems. *Applied Categorical Structures*, 16(3):389–419, June 2008.

[5] A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26:241–265, 1996.

[6] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theoretical Computer Science. Springer, 2006.

[7] H. Ehrig, M. Pfender, and H. Schneider. Graph-grammars: an algebraic approach. In R. Book, editor, *Switching and Automata Theory*, pages 167–180. IEEE Computer Society Press, 1973.

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

[9] F. Hermann. Permutation Equivalence of DPO Derivations with Negative Application Conditions based on Subobject Transformation Systems. *Electronic Communications of the EASST*, 16, 2009.

[10] F. Hermann and H. Ehrig. Process Definition using Subobject Transformation Systems. *EATCS Bulletin*, 95:153–163, 2008.

[11] H. C. M. Kleijn and M. Koutny. Process semantics of general inhibitor nets. *Information and Computation*, 190(1):18–69, 2004.

[12] H.-J. Kreowski. Is parallelism already concurrency? Part 1: Derivations in graph grammars. In *Graph-Grammars and Their Application to Computer Science*, volume 291 of *LNCS*, pages 343–360. Springer, 1986.

[13] S. Lack and P. Sobociński. Adhesive and quasiadhesive categories. *Theoretical Informatics and Applications*, 39(2):511–546, 2005.

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

[15] W. Reisig. *Petri Nets: An Introduction*. EACTS Monographs on Theoretical Computer Science. Springer Verlag, 1985.

[16] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.