Math. Struct. in Comp. Science (2014), vol. 24, iss. 4, e240409, 47 pages. © Cambridge University Press 2014
doi:10.1017/S0960129512000382

Analysis of permutation equivalence in $\mathcal{M}$-adhesive transformation systems with negative application conditions

FRANK HERMANN[†], ANDREA CORRADINI[‡] and HARTMUT EHRIG[§]

[†]*Institut für Softwaretechnik und Theoretische Informatik, Technische Universität Berlin, Berlin, Germany*
and
Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg
Email: frank@cs.tu-berlin.de
[‡]*Dipartimento di Informatica, Università di Pisa, Pisa, Italy*
Email: andrea@di.unipi.it
[§]*Institut für Softwaretechnik und Theoretische Informatik, Technische Universität Berlin, Berlin, Germany*
Email: ehrig@cs.tu-berlin.de

Received 21 February 2011; revised 16 January 2012

$\mathcal{M}$-adhesive categories provide an abstract framework for a large variety of specification frameworks for modelling distributed and concurrent systems. They extend the well-known frameworks of adhesive and weak adhesive HLR categories and integrate high-level constructs such as attribution as in the case of typed attributed graphs.
In the current paper, we investigate $\mathcal{M}$-adhesive transformation systems including negative application conditions (NACs) for transformation rules, which are often used in applications. For such systems, we propose an original equivalence on transformation sequences, called *permutation equivalence*, that is coarser than the classical switch equivalence. We also present a general construction of deterministic processes for $\mathcal{M}$-adhesive transformation systems based on subobject transformation systems. As a main result, we show that the process obtained from a transformation sequence identifies its equivalence class of permutation-equivalent transformation sequences. Moreover, we show how the analysis of this process can be reduced to the analysis of the reachability graph of a generated Place/Transition Petri net. This net encodes the dependencies between rule applications of the transformation sequence, including the inhibiting effects of the NACs.

1. Introduction

The notion of $\mathcal{M}$-adhesive transformation systems provides an abstract framework for transformation systems based on the double pushout (DPO) approach originally developed for graphs (Ehrig *et al.* 1973) and extended to typed attributed graphs and a large variety of Petri nets based on the slightly more specific framework of weak adhesive transformation systems with suitable classes $\mathcal{M}$ of monomorphisms (Ehrig *et al.* 2006). While several analysis techniques for the crucial properties of termination and local confluence have been provided for the general setting, in the current paper, we present general techniques for the analysis of processes of such systems, that is, of equivalence classes of executions differing only in the interleaving of the same transformation steps.

The main problem in this context is to analyse whether a sequence of transformation steps can be rearranged in order to generate all possible equivalent executions, or some specific and possibly better ones. If the system is modelled by a Petri net, these questions can be fairly easily answered: processes (or occurrence nets) incorporate a notion of concurrency (represented as a partial order) that can be exploited to rearrange the tasks, while still respecting causality, so the equivalent computations (firing sequences) are all and only those obtained as linearisations of the process. However, in the current paper, we consider models with two further dimensions, and these considerably complicate the problem. First, we work in the general setting of $\mathcal{M}$-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 classical Petri nets. Second, we take account of negative application conditions (NACs), which are used to ensure the 'absence' of forbidden structures when executing a transformation step. It is well known that NACs significantly improve the specification formalisms based on transformation rules, and lead to more compact and concise models as well as increased usability – they are used widely in non-trivial applications.

In the case of systems with NACs, we propose an original equivalence on transformation sequences, called *permutation equivalence*, which is coarser than the classical switch equivalence based on the local Church–Rosser theorem in the DPO approach including NACs (Lambers 2009) because it might equate two transformation sequences that cannot be obtained one from the other by repeatedly switching independent consecutive steps. As defined in Hermann (2009), two transformation sequences are said to be permutation equivalent if they respect the NACs and, disregarding the NACs, they are switch equivalent.

In order to achieve greater generality, and also motivated by our case study based on typed attributed graph transformation systems, we consider transformation sequences with general (that is, possibly non-monic) matches, and we introduce a new kind of NAC, called *NAC-schemata*, which allow us to reduce the number of classical NACs significantly. Interestingly, we show in our first main result (Theorem 2.23) that permutation equivalence of transformation sequences using general matches and NAC-schemata can be reduced to permutation equivalence of sequences that only uses matches in $\mathcal{M}$ (called $\mathcal{M}$-matches) and classical NACs. This also allows us to reduce the analysis of permutation equivalence to the case of transformation sequences with $\mathcal{M}$-matches and classical NACs.

The main practical analysis problem for permutation equivalence is to construct, for a given transformation sequence, the set of all permutation-equivalent transformation sequences. The brute-force method would be to construct all switch-equivalent sequences disregarding NACs, and then filter out the NAC-consistent ones. However, our case study shows that this brute-force approach is in general very inefficient. In the current paper, we show how to analyse permutation equivalence using subobject transformation systems (STSs) and Petri nets leading to much more efficient solutions.

To this end, we exploit the notion of the process of a transformation sequence, which consists of an STS with an embedding into the original transformation system: this construction is based on and generalises results in Corradini *et al.* (2008) and Hermann (2009) for STSs over adhesive transformation systems with NACs. Our second main result (Theorem 3.18) shows that the constructed process of a given transformation sequence exactly characterises the equivalence class of permutation-equivalent transformation sequences.

To improve the efficiency of the analysis of permutation equivalence, we describe the construction of a dependency net for a given process of a transformation sequence with NACs. This net is given by a standard P/T Petri net that includes a complete account of the causal dependencies and NAC-dependencies among transformation steps. Our remaining main results (Theorems 4.4 and 4.7) show that complete firing sequences of the dependency net are one-to-one with transformation sequences that are permutation equivalent to the given one. This allows us to derive the complete set of permutation-equivalent sequences from a simple analysis of a Petri net. Furthermore, the constructed P/T Petri net can be used to derive specific permutations without generating the complete set first.

1.1. *Organisation of the paper*

The concepts and results of the current paper generalise those presented in Hermann *et al.* (2010) for graph transformation to the more abstract and general framework of $\mathcal{M}$-adhesive transformation systems with general matches. In Section 2, we review $\mathcal{M}$-adhesive categories and present the main concepts of transformation systems with NACs and of permutation equivalence. In Section 3, we introduce the framework of Subobject Transformation Systems (STSs) with NACs and the process construction for $\mathcal{M}$-adhesive transformation systems. In Section 4, we analyse the process by constructing the dependency net given by a Petri net. We discuss related work in Section 5, focusing on Petri nets with inhibitor arcs, and in Section 6 we present our conclusions and suggest some directions for future work. Finally, in Appendix A, we recall the technical details of the $\mathcal{M}$-adhesive category of typed attributed graphs, in Appendix B, we summarise the definitions related to P/T Petri nets, and in Appendix C, we provide the proofs of some auxiliary facts – the proofs of the main results (Theorems 2.23, 3.18, 4.4 and 4.7) are given in the main part of the paper.

2. Transformation systems and permutation equivalence

Most of the definitions and results of the Double Pushout (DPO) approach to transformation systems have been generalised to *adhesive categories* (Lack and Sobociński 2005), (weak) adhesive HLR categories (Ehrig *et al.* 2006), partial map adhesive categories (Heindel 2010) and $\mathcal{M}$-adhesive categories (Ehrig *et al.* 2010), which are the most general of these category types. These frameworks require that pushouts along monos (or along a distinguished subclass of monos, called $\mathcal{M}$-morphisms) 'behave well' with respect to pullbacks. Because of this, it is quite natural for us to work here at this level of generality by referring all definitions and results to an arbitrary but fixed $\mathcal{M}$-adhesive category **C**.

In this section, we review:

— $\mathcal{M}$-adhesive categories together with some additional properties in Section 2.1;
— $\mathcal{M}$-adhesive transformation systems with negative application conditions (NACs) in Section 2.2; and
— the notion of *permutation equivalence* on transformation sequences of such systems in Section 2.3.

Most of the definitions are illustrated using a running case study based on typed attributed graph transformation systems.

2.1. *$\mathcal{M}$-adhesive categories*

The abstract framework of $\mathcal{M}$-adhesive categories unifies several important modelling techniques for parallel and distributed systems. $\mathcal{M}$-adhesive categories are slightly more general than weak adhesive HLR categories (Ehrig *et al.* 2006), and thus include various kinds of graphs and Petri nets.

Definition 2.1 ($\mathcal{M}$-adhesive category). A pair $(\mathbf{C}, \mathcal{M})$ consisting of a category **C** and a class of morphism $\mathcal{M}$ is called an *$\mathcal{M}$-adhesive category* if:

(a) $\mathcal{M}$ is a class of monomorphisms of **C** closed under isomorphisms, composition, and decomposition ($g \circ f \in \mathcal{M}, g \in \mathcal{M} \Rightarrow f \in \mathcal{M}$).
(b) **C** has pushouts and pullbacks along $\mathcal{M}$-morphisms, and $\mathcal{M}$-morphisms are closed under pushouts and pullbacks.
(c) Pushouts in **C** along $\mathcal{M}$-morphisms are *'$\mathcal{M}$-Van Kampen' ($\mathcal{M}$-VK) squares*, that is, for any commutative cube like

$$\begin{tikzcd}
 & & A' \arrow[lld, "f'"'] \arrow[rd, "m'"] \arrow[dd, dotted, "a"] & \\
C' \arrow[r, "n'"] \arrow[dd, "c"'] & D' \arrow[dd, "d"] & & B' \arrow[ll, "g'"'] \arrow[dd, "b"] \\
 & & A \arrow[lld, dotted, "f"'] \arrow[rd, dotted, "m"] & \\
C \arrow[r, "n"'] & D & (1) & B \arrow[ll, "g"]
\end{tikzcd}$$

where the bottom face

$$\begin{tikzcd}
 & A \arrow[ld, "f"'] \arrow[rd, "m"] & \\
C \arrow[rd, "n"'] & (1) & B \arrow[ld, "g"] \\
 & D &
\end{tikzcd}$$

is a pushout along $m \in \mathcal{M}$ and the back faces are pullbacks, and $b, c, d \in \mathcal{M}$, we have that the top face is a pushout if and only if the front faces are pullbacks.

As mentioned above, beginning with Lack and Sobociński (2004), adhesivity has been defined in several variants and sometimes in subtly different ways – see Ehrig *et al.* (2010) for a review of such notions and comparisons between them.

Example 2.2 (the category of typed attributed graphs). The $\mathcal{M}$-adhesive category of our case study is the category of typed attributed graphs $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$, which is given by the slice category $(\mathbf{AGraph}\!\downarrow\! ATG, \mathcal{M})$ of directed attributed graphs over a type graph ATG. The distinguished class $\mathcal{M}$ contains all monomorphisms that are isomorphisms on the data part. According to Ehrig *et al.* (2006), an attributed graph consists of an extended directed graph for the structural part, called the E-graph, together with an algebra for the specification of the carrier sets of the value nodes (see Appendix A). The objects of $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ are attributed graphs with a *typing morphism* to a fixed attributed graph ATG (called the *type graph*), and the arrows are all the attributed graph morphisms preserving the typing. It follows from the results in Ehrig *et al.* (2006) that this category is $\mathcal{M}$-adhesive.

Several $\mathcal{M}$-adhesive categories and results for $\mathcal{M}$-adhesive transformation systems require the existence of epi-mono factorisations, or the more general $\mathcal{E}$-$\mathcal{M}$ pair factorisations. Similarly, we require in this paper that the underlying $\mathcal{M}$-adhesive categories provide extremal $\mathcal{E}$-$\mathcal{M}$ factorisations. This allows us to analyse transformation systems with general matches, that is, matches that are possibly not in $\mathcal{M}$.

Definition 2.3 (extremal $\mathcal{E}$-$\mathcal{M}$ factorisation). Given an $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M})$, the class $\mathcal{E}$ of all extremal morphisms with respect to $\mathcal{M}$ is defined by

$$\mathcal{E} := \{e \in \mathbf{C} \mid \text{ for all } m, f \text{ in } \mathbf{C} \text{ with } m \circ f = e : m \in \mathcal{M} \text{ implies } m \text{ isomorphism}\}.$$

For a morphism $f : A \to B$ in $\mathbf{C}$, an extremal $\mathcal{E}$-$\mathcal{M}$ factorisation of f is given by an object $\overline{B}$ and morphisms

$$(e : A \twoheadrightarrow \overline{B}) \in \mathcal{E}$$
$$(m : \overline{B} \rightarrowtail B) \in \mathcal{M}\ ,$$

such that

$$m \circ e = f.$$

Remark 2.4 (uniqueness of extremal $\mathcal{E}$-$\mathcal{M}$ factorisations). As shown by Braatz *et al.* (2010, Proposition 3), extremal $\mathcal{E}$-$\mathcal{M}$ factorisations are unique up to isomorphism. The class $\mathcal{E}$ is a generalisation of the notion of extremal epimorphisms (Adámek *et al.* 1990), which coincides with the notion of cover (Freyd and Scedrov 1990).

In the case of finitary $\mathcal{M}$-adhesive categories, the extremal factorisation can be performed by constructing all factorisations and stepwise pullbacks of them, as shown by Braatz *et al.* (2010, Proposition 4). An $\mathcal{M}$-adhesive category is finitary if each object A is finite in the sense that there are finitely many $\mathcal{M}$-subobjects

$$[b : B \rightarrowtail A],$$

that is, finitely many $\mathcal{M}$-morphisms up to isomorphism with target A. A typed attributed graph

$$AG = ((G, D), t)$$

in $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ with typing

$$t : (G, D) \to ATG$$

is finite if the graph part of G, that is, all vertex and edge sets except the set V_D of data vertices generated from D, is finite, though the attributed type graph ATG or the data type part D may be infinite because $\mathcal{M}$-morphisms are isomorphisms on the data type part. The restriction of $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ to finite objects forms a finitary category.

Example 2.5 (extremal $\mathcal{E}$-$\mathcal{M}$ factorisation). Given a morphism $f : G \to H$ in the finitary category of typed attributed graphs $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$, the factorisation

$$f = m \circ e$$

is constructed by performing the epi-mono-factorisation on the graph part, that is, on all nodes and edges except the data value nodes V_D, while for the data part f_D we derive

$$e_D : A_G \to A_H$$

with

$$\begin{aligned} e_D(x) &= f_D(x) \\ m_D &= id : A_H \to A_H. \end{aligned}$$

In order to analyse permutation equivalence efficiently in Sections 3 and 4, we will require effective unions for the underlying category $\mathbf{C}$, that is, that the join of two subobjects can be constructed as a pushout in $\mathbf{C}$.

Definition 2.6 (effective unions). Given an $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M})$ and two $\mathcal{M}$-morphisms

$$\begin{aligned} b &: B \to Z \\ c &: C \to Z, \end{aligned}$$

let (f, g) be obtained as the pullback of (b, c) as in

$$\begin{tikzcd}
 & B \arrow[dr, tail, "g'"] \arrow[drr, tail, bend left, "b"] & & \\
A \arrow[ur, tail, "f"] \arrow[dr, tail, "g"'] & (1) & D \arrow[r, "d"] & Z \\
 & C \arrow[ur, tail, "f'"'] \arrow[urr, tail, bend right, "c"'] & &
\end{tikzcd}$$

and let (f', g') be obtained as the pushout (1) of (f, g), with induced mediating morphism $d : D \to Z$. Pushout (1) is then said to be *effective* if $d \in \mathcal{M}$, and the $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M})$ has *effective unions* if for all pairs b, c of $\mathcal{M}$-morphisms, the pushout (1) is effective.

Remark 2.7 (effective unions in $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$). The $\mathcal{M}$-adhesive category $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ has effective unions because by the commutativity of the diagram in Definition 2.6, the morphism d is an isomorphism on the data part.

Assumption 2.8 (general assumption). In order to analyse transformation systems based on an $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M})$, we will base all our further constructions in the current paper on the general assumption that $(\mathbf{C},\mathcal{M})$ provides an extremal $\mathcal{E}$-$\mathcal{M}$ factorisation (Definition 2.3) and effective unions (Definition 2.6).

2.2. *$\mathcal{M}$-adhesive transformation systems*

We will begin this section by reviewing the basic notions of transformation steps and transformation systems. A transformation rule specifies how a given object G can be transformed into a resulting object H. Given a match $m : L \to G$ of the left-hand side of the rule

$$p = \left(L \xleftarrow{l} K \xrightarrow{r} R\right)$$

into the object G such that p is applicable, the resulting object H is intuitively derived by removing the parts that are in L but not in K and then adding those that are in R but not in K. Negative application conditions (NACs) extend a transformation rule to restrict the applicability of the rule by specifying forbidden contexts in which the rule shall not be applied. Intuitively, a match $m\colon L \to G$ satisfies a NAC $n : L \to N$ for a rule p if the image of the left-hand side L in G cannot be extended to an image of the 'forbidden context' N.

It is worth noting that transformation systems with NACs are closely related to Petri nets with *inhibitor arcs*, where inhibitor arcs play a role analogous to that of NACs – the relationship between the two computational models will be discussed further in Section 5. In the current paper, we do not consider nested application conditions (Habel and Pennemann 2009), but we plan to extend our results to this more general kind of application condition in the future.

Definition 2.9 (NAC-consistent transformation steps for $\mathcal{M}$-matches). Given an $\mathcal{M}$-adhesive category $(\mathbf{C},\mathcal{M})$, a *(transformation) rule*

$$p = \left(L \xleftarrow{l} K \xrightarrow{r} R\right),$$

which is also called a *production*, is a pair of $\mathcal{M}$-morphisms with the same source in $\mathbf{C}$. A *negative application condition* (NAC) for a rule p is an $\mathcal{M}$-morphism $n : L \rightarrowtail N$, having the left-hand side of p as source; and a *rule with NACs* is a pair $(p,\mathbf{N})$ where p is a rule and $\mathbf{N}$ is a set of NACs for p. Given an $\mathcal{M}$-morphism $m : L \rightarrowtail G$ into an object $G \in \mathbf{C}$, called a *match*, we say *m satisfies the NAC $n : L \rightarrowtail N$ for p*, written $m \models n$, if there is no $\mathcal{M}$-morphism $q : N \rightarrowtail G$ such that $q \circ n = m$. Furthermore, we say that there is a *NAC-consistent transformation step* from an object G to H using a rule with NACs $(p,\mathbf{N})$ and a match $m : L \rightarrowtail G$, if:

(a) there are two pushouts (1) and (2) in $\mathbf{C}$, as in

$$\begin{tikzcd}
N \arrow[dr, "q"'] & L \arrow[l, tail, "n"'] \arrow[d, tail, "m"'] \arrow[dr, phantom, "(1)"] & K \arrow[l, tail, "l"'] \arrow[r, tail, "r"] \arrow[d] \arrow[dr, phantom, "(2)"] & R \arrow[d] \\
& G & D \arrow[l] \arrow[r] & H
\end{tikzcd}$$

(b) $m \models n$ for each NAC $(n : L \rightarrowtail N) \in \mathbf{N}$.

If condition (a) is satisfied (though (b) may not be, in which case NACs are ignored), we say that there is a *transformation step disregarding NACs* from G to H. In both cases, we write $G \xRightarrow{p,m} H$.

This definition considers transformation steps for $\mathcal{M}$-matches only, but, as we note in the following Remark, this is too restrictive for transformations in our sample category of typed attributed graphs, and therefore in $\mathcal{M}$-adhesive categories in general.

Remark 2.10 (discussion on matches and NACs in $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$). Requiring that a match $m : L \to G$ is in $\mathcal{M}$ implies that the data part of L is isomorphic to that of G. But this is much too restrictive because it is usually the case (see, for example, Example 2.13) that the data algebra of L is given by a term algebra with variables $T_{OP}(X)$, while the data algebra of G is an arbitrary OP-algebra: in this situation the match m is determined, on the data part, by an assignment $ass : X \to A_G$, and it might be neither injective (for example, two variables could be mapped to the same element of A_G) nor surjective.

Therefore, in this general setting, we have to consider transformation steps with respect to arbitrary matches. But this requires us to revisit the basic definitions of NACs and their satisfaction. Indeed, if match $m : L \to G$ does not belong to $\mathcal{M}$, it follows from Definition 2.9 that m trivially satisfies n for any NAC $n : L \rightarrowtail N$: in fact, $n \in \mathcal{M}$ by definition, and if there were some $q \in \mathcal{M}$ such that

$$q \circ n = m,$$

then $m \in \mathcal{M}$ as well, which leads to a contradiction.

There are several options for a meaningful notion of NAC satisfaction in the presence of arbitrary matches. First, we could drop the requirement on q being in $\mathcal{M}$, saying that $m \models n$ if there is no morphism $q : N \to G$ such that

$$q \circ n = m.$$

However, as discussed in Habel *et al.* (1996) for the case of graph transformation, this notion of satisfaction has serious limitations in its expressive power because it cannot express natural constraints like those involving cardinality (for example, *'there must be at least two A-labelled nodes in G'*) or injectivity (for example, *'the match cannot identify two given nodes of L'*), so we prefer to avoid this solution.

Alternatively, we could drop the requirement that NAC $n : L \to N$ has to be in $\mathcal{M}$, but still require any $q : N \to G$ to be in $\mathcal{M}$. Indeed, this is the approach taken in, for example, Habel *et al.* (1996), but we do not consider it very satisfactory because it can lead to a combinatorial explosion in the number of NACs. In fact, let us suppose, for example, that L is a graph consisting of three B-labelled nodes, and that we want to forbid matches from L to any graph G that contains an additional node labelled with A; thus node A is a 'forbidden context'. It is easy to see that we need five distinct NACs, one for each possible different way of identifying subsets of the nodes of L with a match. Similarly, consider again the category of typed attributed graphs, a match

$$(m : L \to G) \notin \mathcal{M}$$

and a NAC

$$(n : L \to N) \notin \mathcal{M}.$$

If the data algebra A_N of N is not isomorphic to the data algebra A_G of G, there cannot exist any $q : N \to G$ in $\mathcal{M}$ making the triangle commute, so $m \models n$ holds trivially. This means that we need at least one different NAC for each distinct algebra (up to isomorphism) that could be the data algebra of an attributed graph to which the rule should not be applicable.

Motivated by this discussion, we will now introduce *NAC-schemata*, which form a new notion of NACs and NAC-consistency that was inspired by Kastenberg *et al.* (2006) and is at the same time both meaningful for general matches and avoids the combinatorial explosion in the number of NACs. A NAC-schema is simply an $\mathcal{M}$-morphism $n : L \rightarrowtail N$, but NAC-satisfaction does not require the absence of an $\mathcal{M}$-morphism $q : N \rightarrowtail G$, but of an $\mathcal{M}$-morphism $q : N' \rightarrowtail G$ with N' being obtained from N, intuitively, by performing the same identifications as in the match $f : L \to G$. This condition is formalised by a pushout over an extremal $\mathcal{E}$-$\mathcal{M}$-factorisation $L \xrightarrow{e} L' \xrightarrow{m} G$ of the match f (see Definition 2.3).

Definition 2.11 (NAC-schemata and Satisfaction). Let

$$p = \left(L \overset{l}{\leftarrowtail} K \overset{r}{\rightarrowtail} R \right)$$

be a rule. A *NAC-schema* for p is then an $\mathcal{M}$-morphism $n : L \rightarrowtail N$. Let $f : L \to G$ be a general match of p, $f = m \circ e$ be its extremal $\mathcal{E}$-$\mathcal{M}$-factorisation and diagram (1) in

$$\begin{tikzcd}
N \arrow[d] \arrow[dr, phantom, "(1)"] & L \arrow[l, tail, "n"'] \arrow[d, two heads, "e"] \arrow[dd, bend left=60, "f"] \\
N' \arrow[dr, tail, bend right, "q"'] & L' \arrow[l, tail, "n'"] \arrow[d, tail, "m"] \\
& G
\end{tikzcd}$$

be constructed as a pushout. Then f satisfies the NAC-schema $n : L \rightarrowtail N$, written $f \models n$, if there is no $q \in \mathcal{M}$ with $q \circ n' = m$. In this case, the match f is said to be NAC-consistent. If $p' = (p, \mathbf{N})$ is a rule with a set of NAC-schemata $\mathbf{N}$, a match satisfies $\mathbf{N}$ if it satisfies all $n \in \mathbf{N}$.

It is worth noting that if match $f : L \to G$ is an $\mathcal{M}$-morphism, then satisfaction of a NAC-schema $n : L \to N$ coincides with classical satisfaction because the factorisation is trivially $f = f \circ id$.

A set of named transformation rules forms a transformation system, and the naming is specified by a mapping

$$\pi : P \to RULES(\mathbf{C}, \mathcal{M})$$

from the set of rule names P to the set of rules in an $\mathcal{M}$-adhesive category $(\mathbf{C}, \mathcal{M})$.

Definition 2.12 ($\mathcal{M}$-adhesive transformation system). An *$\mathcal{M}$-adhesive transformation system (TS) over* $(\mathbf{C}, \mathcal{M})$ *for general matches* is a pair

$$TS = (P, \pi)$$

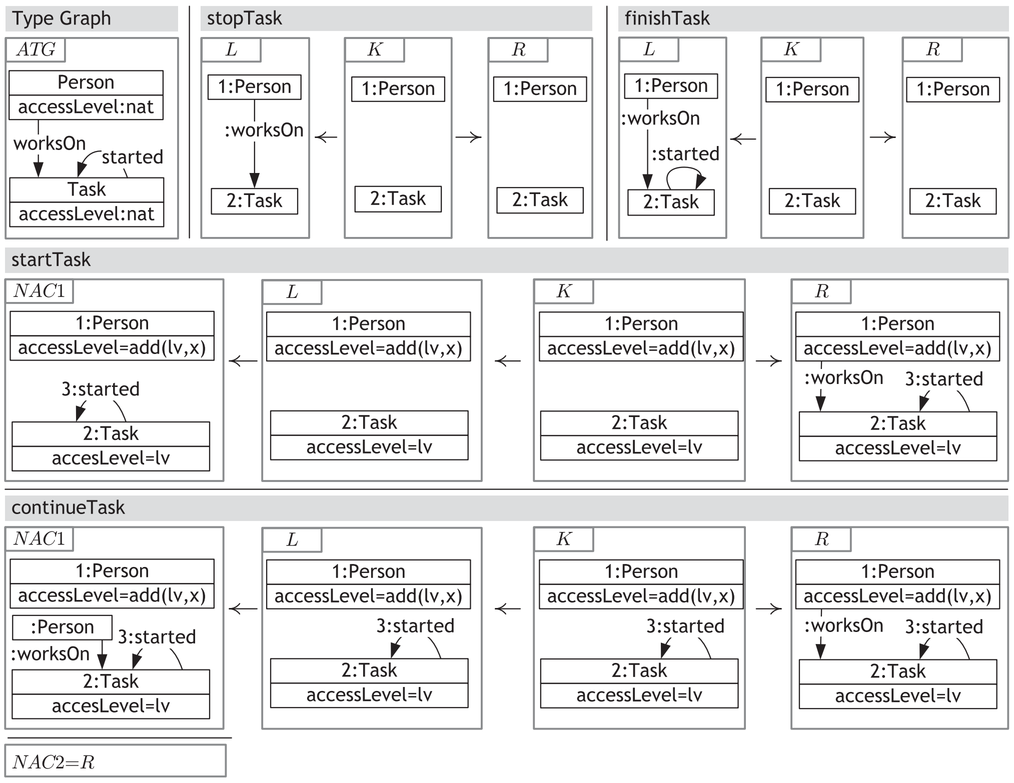


Fig. 1. Typed attributed graph transformation system *GTS*

where P is a set of rule names, and π maps each name $p \in P$ to a rule

$$\pi(p) = \left(\left(L \overset{l}{\hookleftarrow} K \overset{r}{\rightarrowtail} R\right), \mathbf{N}_S\right)$$

with NAC-schemata $\mathbf{N}_S$. A *NAC-consistent transformation sequence* of *TS* is a sequence

$$G_0 \xRightarrow{p_1,m_1} G_1 \cdots \xRightarrow{p_n,m_n} G_n,$$

where $p_1, \ldots, p_n \in P$ and

$$d_i = G_{i-1} \xRightarrow{\pi(p_i),m_i} G_i$$

is a transformation step with NAC-consistent match (see Definition 2.11) for $i \in 1, \ldots, n$. We will sometimes denote a transformation sequence by $d = (d_1; \ldots; d_n)$, where each d_i denotes a single transformation step.

An $\mathcal{M}$*-adhesive transformation system (TS) over* $(\mathbf{C}, \mathcal{M})$ *for* $\mathcal{M}$*-matches* is defined as above, where, however, the set $\mathbf{N}_S$ of NAC-schemata is replaced by a set of NACs $\mathbf{N}$ with NAC-consistency according to Definition 2.9.

Example 2.13 (typed attributed graph transformation system). The $\mathcal{M}$-adhesive transformation system for general matches of our case study is the typed attributed graph transformation system *GTS* in Figure 1. The *type graph ATG* specifies persons and

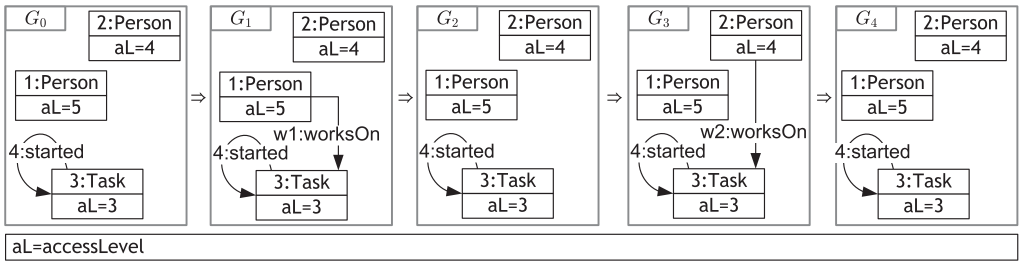


Fig. 2. Transformation sequence d of GTS

tasks: a task is active if it has a ':started' loop and it can be assigned to a person with a ':worksOn' edge. Moreover, the attribute 'accessLevel' specifies the required access level of tasks and the allowed maximal access level of persons. Rule 'startTask' is used to start a task, where the access level of the task can be at most equal to the access level of the considered person, and the NAC-schema ensures that the task has not already started. Rules 'stopTask' and 'finishTask' both remove the assignment of a person, and 'finishTask' also deletes the marker ':started' to specify that the task has been completed. Finally, rule 'continueTask' assigns an already started task to a person. This rule contains two NAC-schemata that forbid the assignment of persons to already assigned tasks: either if another person is already assigned to that task ('NAC1') or the selected person is already assigned ('NAC2'). Figure 2 shows a NAC-consistent transformation sequence

$$d = (G_0 \xRightarrow{continueTask,f_1} G_1 \xRightarrow{stopTask,f_2} G_2 \xRightarrow{continueTask,f_3} G_3 \xRightarrow{stopTask,f_4} G_4)$$

of GTS. The first graph of the transformation sequence contains exactly one task, which is first assigned to node '1:Person', and then, after being stopped, to node '2:Person'. The NAC-schemata of the rule '*continueTask*' are checked at graphs G_0 and G_2. The constructed pushouts according to Definition 2.11 yield instantiated NACs $n' : L \rightarrowtail N'$, with N' containing an edge of type *worksOn*. Since G_0 and G_2 do not contain an edge of this type, there is no embedding q from N' into these graphs such that the NAC-schemata are satisfied by the matches. Therefore, the transformation sequence is NAC-consistent because the remaining steps do not involve NACs. Note that the use of NAC-schemata and general matches is essential for our case study. If we used $\mathcal{M}$-matches or classical NACs, we would have to provide specific rules and NACs for each possible variable assignment for people with different actual access levels (see also Remark 2.10).

While general matches for $\mathcal{M}$-adhesive transformation systems lead to extended concepts for NACs and NAC satisfaction, we will now show that we can reduce the analysis of a concrete given transformation sequence to the case of $\mathcal{M}$-matches by instantiating the rules and transformation diagrams along the given matches. Note, in particular, that for transformation steps along $\mathcal{M}$-matches, the instantiated transformation steps coincide with the given ones.

$$\begin{tikzcd}
N \arrow[d] \arrow[rd, phantom, "(7)"] & L \arrow[l, tail, "n"'] \arrow[d, two heads, "e"'] \arrow[dd, bend left, "f"] \arrow[rd, phantom, "(3)"] & K \arrow[l, tail] \arrow[r, tail] \arrow[d, two heads, "k_e"'] \arrow[dd, bend left, "k"] \arrow[rd, phantom, "(4)"] & R \arrow[d, two heads, "e^*"'] \arrow[dd, bend left, "f^*"] \\
N' & L' \arrow[l, tail, "n'"] \arrow[d, tail, "m"'] \arrow[rd, phantom, "(5)"] & K' \arrow[l, tail] \arrow[r, tail] \arrow[d, tail] \arrow[rd, phantom, "(6)"] & R' \arrow[d, tail, "m^*"'] \\
 & G & D \arrow[l, tail] \arrow[r, tail] & H
\end{tikzcd}$$

Fig. 3. Construction of instantiated rules and transformation steps

Definition 2.14 (instantiated rules and transformation sequences). Let

$$G \xRightarrow{p,f} H$$

be a NAC-consistent transformation step through a rule

$$p = ((L \leftarrowtail K \rightarrowtail R), \mathbf{N}_S)$$

with NAC-schemata $\mathbf{N}_S$. Let $f = m \circ e$ be the extremal $\mathcal{E}$-$\mathcal{M}$ factorisation of match f. The *instantiated transformation step* is given by $G \xRightarrow{p',m} H$ with *instantiated rule* p' *derived via* e and constructed as follows according to Figure 3. First construct pullback (PB) (5) leading to pushouts (POs) (3) and (5) by PB splitting and the $\mathcal{M}$-PO-PB Decomposition Lemma (Ehrig *et al.* 2006, Theorem 4.26 (2)). Then construct PO (4), which leads to PO (6) by PO splitting. Now instantiate each NAC-schema $n : L \rightarrowtail N$ in $\mathbf{N}_S$ along morphism e (Definition 2.11) to give a new NAC $n' : L' \rightarrowtail N'$. Let $\mathbf{N}'$ be the new set of NACs consisting of all NACs $n' : L' \rightarrowtail N'$ obtained from all $n \in \mathbf{N}_S$. The *instantiated rule* is given by

$$p' = ((L' \leftarrowtail K' \rightarrowtail R'), \mathbf{N}')$$

and the *instantiated transformation step* is defined by $G \xRightarrow{p',m} H$ with $m \in \mathcal{M}$ through DPO diagram ((5) + (6)).

Let d be a transformation sequence. Then the instantiated transformation sequence d_I is derived by instantiating each transformation step as defined above.

The instantiation of rules ensures that transformation steps of the instantiated rule are in one-to-one correspondence with those of the original rule.

Fact 2.15 (compatibility of applicability and NAC-consistency with instantiation). Let

$$G_1 \xRightarrow{p,f_1} H_1$$

be a NAC-consistent transformation step and

$$G_1 \xRightarrow{p',m_1} H_1$$

be the instantiated step with extremal $\mathcal{E}$-$\mathcal{M}$-factorisation $f_1 = m_1 \circ e$ according to Definition 2.14. Let

$$m_2 : L' \to G_2$$

be a match with $m_2 \in \mathcal{M}$. Then, there is a NAC-consistent transformation step

$$G_2 \xRightarrow{p',m_2} H_2$$

through p' if and only if there is a NAC-consistent transformation step

$$G_2 \xRightarrow{p,f_2} H_2$$

through p with $f_2 = m_2 \circ e$.

Note that the proofs of all the ‘Facts’ stated in the paper are collected together in Appendix C.

Example 2.16 (instantiation of transformation sequence). In the case of typed attributed graphs, the instantiated rules are attributed through the algebra A of the transformed objects $G_0 \ldots G_n$. As happens in most cases, the algebra A in our case study is different from the term algebra $T_{OP}(X)$. The instantiation of the transformation sequence d in Figure 2 through the rules of Figure 1 is performed according to Definition 2.14. In this way, we derive an instantiated transformation sequence d_I. By definition, the lower line of the DPO diagrams coincides with the one of d in Figure 2. The instantiated rules for the four steps are shown in Figures 6 and 7 in Section 3.2 (rules ‘stop1’, ‘stop2’, ‘cont1’ and ‘cont2’), and they are used in the following sections for the analysis of permutation equivalence.

2.3. *Permutation equivalence of transformation sequences*

The classical theory of the DPO approach introduces an equivalence between transformation sequences, called *switch equivalence*, that relates the sequences that differ only in the order in which independent transformation steps are performed. More precisely, two sequences are switch equivalent if each of them can be obtained from the other by repeatedly exchanging consecutive transformation steps that are *sequentially independent*.

Definition 2.17 (sequential independence). Let

$$d_1 = G_0 \xRightarrow{p_1,f_1} G_1$$
$$d_2 = G_1 \xRightarrow{p_2,f_2} G_2$$

be two transformation steps disregarding NACs. Then they are *sequentially independent* if there exist arrows $i : R_1 \to D_2$ and $j : L_2 \to D_1$ such that

$$g_2 \circ i = f_1^*$$
$$h_1 \circ j = f_2.$$

The following diagram shows part of the transformation diagrams:

$$\begin{tikzcd}
K_1 \arrow[r, tail] \arrow[d] & R_1 \arrow[dr, "f_1^*"] \arrow[drrr, dotted, "i"] & & L_2 \arrow[dl, "f_2"'] \arrow[dlll, dotted, "j"'] & K_2 \arrow[l, tail] \arrow[d] \\
D_1 \arrow[rr, tail, "h_1"] & & G_1 & & D_2 \arrow[ll, tail, "g_2"']
\end{tikzcd}$$

Intuitively, two steps are not sequentially independent if the second one accesses (that is, reads or consumes) some resources produced by the first one, or if it consumes some resources accessed by the first one. In both cases we argue that there is an (implicit) 'information flow' from the first to the second transformation step, and this requires that the second step occurs after the first one.

If steps d_1 and d_2 are sequentially independent, then, according to the Local Church–Rosser theorem (Ehrig *et al.* 2006, Theorem 5.12), they can be 'switched' to give transformation steps

$$d'_2 = G_0 \xRightarrow{p_2,\overline{f_2}} G'_1$$
$$d'_1 = G'_1 \xRightarrow{p_1,\overline{f_1}} G_2,$$

which apply the two rules in the opposite order: this mechanism generates switch equivalence in the following sense.

Definition 2.18 (switch equivalence for transformation sequences). Let

$$d = (d_1 ; \ldots ; d_k ; d_{k+1} ; \ldots ; d_n)$$

be a transformation sequence, where d_k and d_{k+1} are two sequentially independent transformation steps, 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 transformation sequences containing both $\overset{sw}{\sim}$ and the isomorphism relation $\cong$[†].

Habel *et al.* (1996) and Lambers (2009) proposed a refined notion of sequential independence for graph transformation systems with NACs where two consecutive steps $d_1 ; d_2$ are sequential independent if, besides satisfying the conditions of Definition 2.17, the transformation steps $d'_2 ; d'_1$ obtained after switching them are NAC-consistent. The switch equivalence for NAC-consistent transformation sequences is then defined exactly as in Definition 2.18, but using the new definition of sequential independence, so this is a natural generalisation and conservative extension of the switch equivalence for sequences without NACs.

However, we believe the switch equivalence for NAC-consistent sequences is too restrictive for the following reason. Suppose $d_1 ; d_2$ are sequential independent according to Definition 2.17, but that after the switching $d'_2 ; d'_1$ is not NAC-consistent. Then either d'_2 does not satisfy the NACs, which means that d_2 can fire after d_1 because d_1 deletes some resource that would represent a forbidden context for d_2, or the NACs of d'_1 are not satisfied because d_2 creates a resource that matches (part of) a NAC of the transformation rule of d_1. In both cases, we argue that there is no information flow from d_1 to d_2, so there is no conceptual obstacle to the possibility that the two steps occur in the opposite order (even if not consecutively) in another equivalent transformation sequence.

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

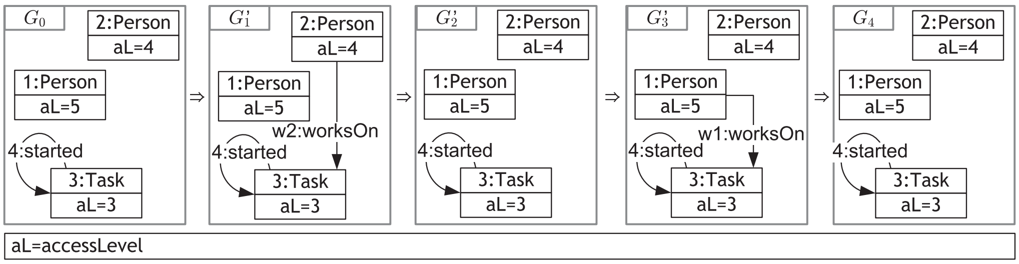


Fig. 4. Permutation equivalent transformation sequence d' of *GTS*

These considerations justify the following definition of *permutation equivalence* for NAC-consistent transformation sequences, which is coarser than the corresponding switch equivalence in the sense that it equates more sequences.

Definition 2.19 (permutation equivalence of transformation sequences). Two NAC-consistent transformation sequences d and d' are *permutation equivalent*, written $d \overset{\pi}{\approx} d'$ if, disregarding the NACs, they are switch equivalent as in Definition 2.18. The equivalence class of all permutation equivalent transformation sequences $\pi\text{-}Equ(d)$ of d is given by

$$\pi\text{-}Equ(d) = \{d' \mid d' \overset{\pi}{\approx} d\}.$$

It follows immediately from the definition that permutation equivalence coincides with the standard switch equivalence on derivations without NACs. We will see in the next section that all NAC-consistent transformation sequences that are permutation equivalent to a given sequence d can be obtained as suitable linearisations of a process-like structure generated from d, and this recovers in our framework here a result similar to the one presented in Corradini *et al.* (1996) for standard switch equivalence and graph processes. As far as we are aware, there is no similar result for the switch equivalence on NAC-consistent sequences defined in Habel *et al.* (1996) and Lambers (2009).

Example 2.20 (permutation equivalence). Figure 4 shows a NAC-consistent transformation sequence

$$d' = (G_0 \xRightarrow{continueTask,f_1'} G_1' \xRightarrow{stopTask,f_2'} G_2' \xRightarrow{continueTask,f_3'} G_3' \xRightarrow{stopTask,f_4'} G_4),$$

which is permutation equivalent to the transformation sequence d of Figure 2 by performing the following switchings of steps disregarding NACs (where we write $(d_i';d_j')$ to denote the result of switching $(d_j;d_i)$):

$$(d_2;d_3),(d_1;d_3'),(d_2';d_4),(d_1';d_4').$$

The equivalent transformation sequences are not switch equivalent with NACs because there is no pair of independent consecutive transformation steps in any of the transformation sequences.

Remark 2.21 (complexity of the analysis). The brute-force method for generating all permutation-equivalent sequences would be to first construct all switch-equivalent ones disregarding NACs, and then filter out the NAC-consistent sequences. But, as discussed in Hermann *et al.* (2010), this is far too inefficient for realistic examples: given the transformation sequence d of Figure 2, the sequence $d^3 = (d; d; d)$ consisting of twelve steps would lead to 7.484.400 switch-equivalent sequences disregarding NACs, out of which only 720 are NAC-consistent and therefore permutation equivalent. For this reason, we will describe a more efficient approach in Section 4 by generating the permutation-equivalent sequences directly. As shown in Hermann (2009) and Hermann *et al.* (2010), the construction of the derived Petri net has polynomial time complexity.

Given a transformation sequence d through general matches, we will now show (in Theorem 2.23) that we can reduce the analysis of permutation equivalence to $\mathcal{M}$-matches. To this end, we will first show that there is a one-to-one correspondence between sequential independence disregarding NACs for the instantiated steps and for the corresponding original steps.

Fact 2.22 (sequential independence disregarding NACs for instantiated steps). Let

$$(d_1; d_2) = (G_0 \xRightarrow{p_1, f_1} G_1 \xRightarrow{p_2, f_2} G_2)$$

be two transformation steps disregarding NACs and let

$$(d_{1,I}; d_{2,I}) = (G_0 \xRightarrow{p'_1, m_1} G_1 \xRightarrow{p'_2, m_2} G_2)$$

be their instantiated steps according to Definition 2.14. Then, d_1 and d_2 are sequentially independent disregarding NACs if and only if $d_{1,I}$ and $d_{2,I}$ are sequentially independent disregarding NACs.

Theorem 2.23 (reduction of permutation equivalence for general matches to $\mathcal{M}$-matches). Two transformation sequences d and d' with general matches are permutation equivalent if and only if their instantiated transformation sequences d_I and d'_I with $\mathcal{M}$-matches are permutation equivalent, that is,

$$d \overset{\pi}{\approx} d' \Leftrightarrow d_I \overset{\pi}{\approx} d'_I.$$

Proof. First, we have by Fact 2.22 and Definition 2.18 that switch equivalence disregarding NACs is implied for both directions. By Fact 2.15, we have that the transformation steps and hence, also the transformation sequences, are additionally NAC consistent. Therefore,

$$d \overset{\pi}{\approx} d' \Leftrightarrow d_I \overset{\pi}{\approx} d'_I. \qquad \square$$

Remark 2.24 (permutation equivalence for general matches). Theorem 2.23 means we can base our analysis techniques in the following sections on the derived transformation sequences with $\mathcal{M}$-morphisms only, as visualised in Figure 5. Given a transformation sequence d, we first instantiate d according to Definition 2.14 so that the lower transformation diagrams form a new transformation sequence d_I with $\mathcal{M}$-matches only. We can then analyse permutation equivalence for d_I and derive the analysis results for d using Theorem 2.23. In particular, the derived permutation-equivalent transformation

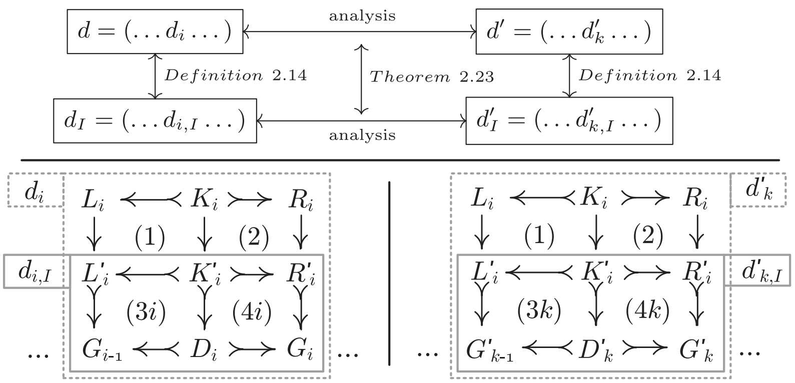


Fig. 5. Correspondence between transformation sequences and their instantiations

sequences d'_I of d_I can be composed with the upper DPO diagrams of the instantiation leading to permutation-equivalent transformation sequences d' of d.

Assumption 2.25 (general assumption). As a consequence of the above remark, in the following sections we will consider transformation sequences with $\mathcal{M}$-matches only. In fact, when analysing transformation sequences with general matches, it is sufficient to analyse their instantiated sequences and then lift the results back to the original sequences using Theorem 2.23.

3. From subobject transformation systems to processes of $\mathcal{M}$-adhesive transformation systems

In the theory of Petri nets (Reisig 1985), starting from a given firing sequence, we can build a *deterministic process*, which is a net that records all the transitions fired in the sequence, together with their causal dependencies. Similar constructions have been proposed for graph transformation (Corradini *et al.* 1996) and for transformation systems based on adhesive categories (Baldan *et al.* 2006; Corradini *et al.* 2008). In particular, Corradini *et al.* (2008) showed that starting with a transformation sequence (without NACs) in an adhesive transformation system, we can build a *Subobject Transformation System (STS)*, that is, a system where the sequence can be simulated and where it is possible to analyse the independence between steps of the sequence. In this section, we will generalise these results to transformation systems with NACs, and consider the more general framework of $\mathcal{M}$-adhesive categories.

3.1. *$\mathcal{M}$-subobject transformation systems*

Subobject transformation systems are essentially double-pushout transformation systems over the lattice of subobjects **Sub**(T) of a given object T of an adhesive category **C**. We will review here the main definitions of Corradini *et al.* (2008) in the case of $\mathcal{M}$-adhesive categories, starting with the notion of an $\mathcal{M}$-subobject. In the following, we will assume that **C** is an arbitrary but fixed $\mathcal{M}$-adhesive category, unless specified otherwise, and we will use $|\mathbf{C}|$ to denote the class of objects of **C**.

Definition 3.1 (category of $\mathcal{M}$-subobjects). Let T be an object of an $\mathcal{M}$-adhesive category **C**. Two $\mathcal{M}$-morphisms $a : A \rightarrowtail T$ and $a' : A' \rightarrowtail T$ are *equivalent* if there exists an isomorphism $\phi : A \to A'$ such that $a = a' \circ \phi$. An *$\mathcal{M}$-subobject* $[a : A \rightarrowtail T]$ of T is an equivalence class of $\mathcal{M}$-morphisms with target T. The *category of $\mathcal{M}$-subobjects of T*, denoted $\mathbf{Sub}_{\mathcal{M}}(T)$, has the $\mathcal{M}$-subobjects of T as objects. Furthermore, there is an arrow from $[a : A \rightarrowtail T]$ to $[b : B \rightarrowtail T]$ if there exists a morphism $f : A \to B$ such that $a = b \circ f$. In this case, f is an $\mathcal{M}$-morphism and it is unique (so $\mathbf{Sub}_{\mathcal{M}}(T)$ is a partial order), and we write

$$[a : A \rightarrowtail T] \subseteq [b : B \rightarrowtail T].$$

We will usually simply write A to denote an $\mathcal{M}$-subobject $[a : A \rightarrowtail T]$, leaving the $\mathcal{M}$-morphism a implicit, and, correspondingly, we will write

$$A \subseteq B$$

if

$$[a : A \rightarrowtail T] \subseteq [b : B \rightarrowtail T],$$

and write

$$f : A \hookrightarrow B$$

to denote the corresponding embedding.

If $\mathcal{M}$ is the class of all monomorphism of **C**, as for adhesive categories, then $\mathbf{Sub}_{\mathcal{M}}(T)$ for $T \in |\mathbf{C}|$ is the standard category of subobjects of T. The following notions of ‘intersection’ and ‘union’ will be used in the definition of direct derivations of an STS.

Definition 3.2 (intersection and union in $\mathbf{Sub}_{\mathcal{M}}(T)$). Let $A, B \in |\mathbf{Sub}_{\mathcal{M}}(T)|$ be two $\mathcal{M}$-subobjects with $T \in |\mathbf{C}|$. The product of A and B in $\mathbf{Sub}_{\mathcal{M}}(T)$ will be called their *intersection*, denoted $A \cap B$. The coproduct of A and B in $\mathbf{Sub}_{\mathcal{M}}(T)$ will be called their *union*, denoted $A \cup B$.

In the case of adhesive categories, Lack and Sobocinski (2005) showed that intersections and unions exist, unions are effective and $\mathbf{Sub}(T)$ is a distributive lattice for any $T \in \mathbf{C}$. We will show that for $\mathcal{M}$-adhesive categories, $\mathbf{Sub}_{\mathcal{M}}(T)$ is also a distributive lattice if unions are effective. Since unions are not effective in general, we require this property by Assumption 2.8.

Fact 3.3 (intersection in $\mathbf{Sub}_{\mathcal{M}}(T)$). Let $T \in |\mathbf{C}|$ and $A, B \in \mathbf{Sub}_{\mathcal{M}}(T)$. The intersection $A \cap B$ exists and it is given by the pullback

$$\begin{tikzcd}
A \cap B \arrow[r, hook, "p_A"] \arrow[d, hook, "p_B"'] \arrow[dr, phantom, "(1)"] & A \arrow[d, hook, "a"] \\
B \arrow[r, hook, "b"'] & T
\end{tikzcd}$$

in $\mathbf{C}$ with the $\mathcal{M}$-morphism

$$i : A \cap B \xrightarrow{a \circ p_A} T.$$

Remark 3.4 (unions in $\mathbf{Sub}_{\mathcal{M}}(T)$ for $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$). According to Remark 2.7, the category of typed attributed graphs $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ has effective unions, that is, the union $A \cup B$ of two $\mathcal{M}$-subobjects A and B can be constructed as the pushout over the intersection $A \cap B$ in $\mathbf{C}$.

In contrast to $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$, the category of simple graphs provides an example of an $\mathcal{M}$-adhesive category that has unions, but where unions are not effective. A simple graph is a pair (A, N) where N is a set of nodes and $A \subseteq N \times N$ is a set of arcs. A morphism

$$f : (N, A) \to (N', A')$$

is a function $f : N \to N'$ such that

$$(n_1, n_2) \in A \Rightarrow (f(n_1), f(n_2)) \in A'.$$

Such a morphism is *regular* if it is injective, and the opposite implication also holds.

Heindel (2010) showed that the category of simple graphs with the class $\mathcal{M}$ of all regular monomorphism is a partial-map adhesive category, and is thus $\mathcal{M}$-adhesive by the results in Ehrig *et al.* (2010). However, it is well known that unions are not effective in this category: given the graph

$$G = (\{n, n'\}, \{(n, n')\}),$$

the pushout built over the regular subobjects $(\{n\}, \emptyset)$ and $(\{n'\}, \emptyset)$ is $(\{n, n'\}, \emptyset)$, which is not a regular subobject of G.

Fact 3.5 (distributivity). Let $\mathbf{C}$ be an $\mathcal{M}$-adhesive category with effective unions and T be an object of $\mathbf{C}$, then the union and intersection constructions in $\mathbf{Sub}_{\mathcal{M}}(T)$ are distributive, that is,

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \tag{i}$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C). \tag{ii}$$

Using the notion of $\mathcal{M}$-subobjects and the distributivity law for intersection and union, we will now present subobject transformation systems (STSs) as a formal framework for the concurrent semantics of $\mathcal{M}$-adhesive transformation systems. This concept generalises the notion of elementary nets, which form the category of process nets for P/T Petri nets, in the same way that STSs form the category of process transformation systems for $\mathcal{M}$-adhesive transformation systems. The typical effect occurring in elementary nets, namely the situation of contact, also appears in the setting of STSs and forms an additional application condition for the transformation rules. Thus, we will first introduce the general setting of STSs on which we will then base the construction of the process of a transformation sequence.

Definition 3.6 (STS with NACs). A *Subobject Transformation System with NACs* $\mathcal{S} = (T, P, \pi)$ over an $\mathcal{M}$-adhesive category $\mathbf{C}$ with effective unions consists of:

— a super object $T \in \mathbf{C}$;
— a set of rule names P, which are also called productions; and
— a function π that maps each rule name $q \in P$ to a *rule with negative application conditions (NACs)*

$$((L, K, R), \mathbf{N}),$$

where

- L, K and R are objects in $\mathbf{Sub}_{\mathcal{M}}(T)$;
- $K \subseteq L$;
- $K \subseteq R$; and
- its NACs $\mathbf{N}$ are given by

$$\mathbf{N} = (N, \nu)$$

consisting of a set N of names for the NACs and a function ν mapping each NAC name $i \in N$ to a *NAC* $\nu(i)$, which is given by a subobject

$$\nu(i) = N_i \in \mathbf{Sub}_{\mathcal{M}}(T)$$

with

$$L \subseteq N_i \subseteq T.$$

The abbreviated notation $\mathbf{N}[i]$ refers to a NAC N_i of rule p with $\nu(i) = N_i$.

Direct derivations ($G \overset{q}{\Longrightarrow} G'$) with NACs in an STS correspond to transformation steps with NACs in an $\mathcal{M}$-adhesive TS, but the construction is simplified because morphisms between two subobjects are unique. There is no need for pattern matching, and for this reason, we use the notion of derivations within an STS in contrast to transformation sequences in an $\mathcal{M}$-adhesive TS, and we use names $\{p_1, \ldots, p_n\}$ for rules in an $\mathcal{M}$-adhesive TS and $\{q_1, \ldots, q_n\}$ for rules in an STS.

Definition 3.7 (direct derivations in an STS). Let $\mathcal{S} = (T, P, \pi)$ be a Subobject Transformation System with NACs and

$$\pi(q) = ((L, K, R), \mathbf{N})$$

be a production with NACs. Let $G \in |\mathbf{Sub}_{\mathcal{M}}(T)|$. Then there is a *direct derivation disregarding NACs* from G to G' using q if $G' \in |\mathbf{Sub}_{\mathcal{M}}(T)|$ and there is an object $D \in \mathbf{Sub}_{\mathcal{M}}(T)$ such that:

(i) $L \cup D = G$;
(ii) $L \cap D = K$;
(iii) $D \cup R = G'$;
(iv) $D \cap R = K$.

We say that there is a *direct derivation with NACs* from G to G' using q, if in addition to all the above conditions, $\mathbf{N}[i] \not\subseteq G$ for each $\mathbf{N}[i]$ in $\mathbf{N}$. In both cases we write $G \overset{q}{\Longrightarrow} G'$.

It is instructive to consider the relationship between a direct derivation in an STS and the usual notion of a DPO transformation step in an $\mathcal{M}$-adhesive category. We can make this comparison because we can consider a rule $L_q \supseteq K_q \subseteq R_q$ as the underlying span of $\mathcal{M}$-morphisms in **C**. However, given an $\mathcal{M}$-subobject $G \in \mathbf{Sub}_{\mathcal{M}}(T)$ such that $L \subseteq G$, we need an additional condition to be satisfied in order to guarantee that the result of a double-pushout transformation in **C** using rule $L_q \supseteq K_q \subseteq R_q$ and match $L \subseteq G$ is again an object in $\mathbf{Sub}_{\mathcal{M}}(T)$.

In fact, suppose $G \cap R \nsubseteq L$. Intuitively, this means that part of the $\mathcal{M}$-subobject G is created but not deleted by the rule: if we were allowed to apply the rule at this match using a DPO transformation step, the resulting object would contain the common part twice and consequently the resulting morphism to T would not be an $\mathcal{M}$-morphism: that is, the result would not be an $\mathcal{M}$-subobject of T.

By analogy with a similar concept for elementary Petri nets, we shall say that there is a *contact situation* for a rule (L, K, R) at an $\mathcal{M}$-subobject $G \supseteq L \in \mathbf{Sub}_{\mathcal{M}}(T)$ if $G \cap R \nsubseteq L$: as stated by the next result, STS direct derivations and DPO transformation steps coincide if there is no contact.

Proposition 3.8 (STS derivations are contact-free double pushouts). Let

$$\mathcal{S} = (T, P, \pi)$$

be an STS over an $\mathcal{M}$-adhesive category **C** with effective unions. Let

$$\pi(q) = (L, K, R)$$

be a rule and $G \in |\mathbf{Sub}_{\mathcal{M}}(T)|$. Then $G \overset{q}{\Longrightarrow} G'$ if and only if:

— $L \subseteq G$;
— $G \cap R \subseteq L$; and
— there is an object D in **C** such that diagrams (1) and (2) in

$$\begin{tikzcd}
L \arrow[d, tail, "m"'] \arrow[dr, phantom, "(1)"] & K \arrow[l, tail, "l"'] \arrow[r, tail, "r"] \arrow[d, tail, "k"] \arrow[dr, phantom, "(2)"] & R \arrow[d, tail, "n"] \\
G & D \arrow[l, tail, "f"] \arrow[r, tail, "g"'] & G'
\end{tikzcd}$$

are pushouts in **C**.

Proof. See the proof of Corradini *et al.* (2008, Proposition 6). □

As a consequence, every derivation

$$d = (G_0 \overset{q_1}{\Longrightarrow} \dots \overset{q_n}{\Longrightarrow} G_n)$$

in an STS $\mathcal{S}$ over an $\mathcal{M}$-adhesive category **C** determines a diagram in category **C**, consisting of a sequence of (conflict-free) double pushouts. We will write $\mathit{trafo}_{\mathcal{S}}(s)$ to denote this diagram in **C**, where $s = \langle q_1, \dots, q_n \rangle$.

3.2. *Processes of $\mathcal{M}$-adhesive transformation systems*

Using the notion and construction of processes for adhesive transformation systems without NACs in Baldan *et al.* (2006) and Corradini *et al.* (2008) as a basis, in this section we will present the construction of processes for a transformation sequence of an $\mathcal{M}$-adhesive transformation systems with NACs. The first step is to construct the STS for a given transformation sequence d with matches in $\mathcal{M}$ due to Assumption 2.25 based on Theorem 2.23.

Definition 3.9 (STS of a transformation sequence with $\mathcal{M}$-matches). Let

$$d = (G_0 \xRightarrow{p_1,m_1} \dots \xRightarrow{p_n,m_n} G_n)$$

be a NAC-consistent transformation sequence in an $\mathcal{M}$-adhesive TS with matches in $\mathcal{M}$. The *STS with NACs generated by d* is given by

$$STS(d) = (T, P, \pi),$$

and its components are constructed as follows:

— T is an arbitrarily chosen but fixed colimit of the sequence of DPO-diagrams given by d;

— $P = \{i \mid 0 < i \leqslant n\}$ is a set of natural numbers containing a canonical rule occurrence name for each rule occurrence in d.

— For each $k \in P$, we have $\pi(k)$ is defined by

$$\pi(k) = ((L_k, K_k, R_k), \mathbf{N}_k),$$

where each component X of the production p_k (with $X \in \{L_k, K_k, R_k\}$) is regarded as a subobject of T through the natural embedding $in_T(X)$.
And, for each $k \in \{1, \dots, n\}$, the NACs

$$\mathbf{N}_k = (N_k, v)$$

are constructed as follows. Let $J_{\mathbf{N}_k}$ be the set of subobjects of T that are possible images of the NACs of the production $(p_k, \mathbf{N}_k)$ with respect to the match

$$in_T : L_k \rightarrowtail T.$$

More precisely,

$$J_{\mathbf{N}_k} = \{[j : N \rightarrowtail T] \in \mathbf{Sub}_{\mathcal{M}}(T) \mid \exists\, (n : L_k \rightarrowtail N) \in \mathbf{N}_k \wedge j \circ n = in_T(L_k)\}.$$

Then the NAC names N_k are given by

$$N_k = \{i \mid 0 < i \leqslant |J_{\mathbf{N}_k}|\},$$

and the function v is an arbitrary but fixed bijective function $v : N_k \to J_{\mathbf{N}_k}$ mapping NAC names to the corresponding subobjects.

In order to ensure termination when analysing permutation equivalence in concrete case studies, we will only consider transformation sequences such that the colimit object T is finite, that is, has finitely many $\mathcal{M}$-subobjects. Finiteness is guaranteed if each rule of TS has finite left- and right-hand sides, and if the start object of the transformation

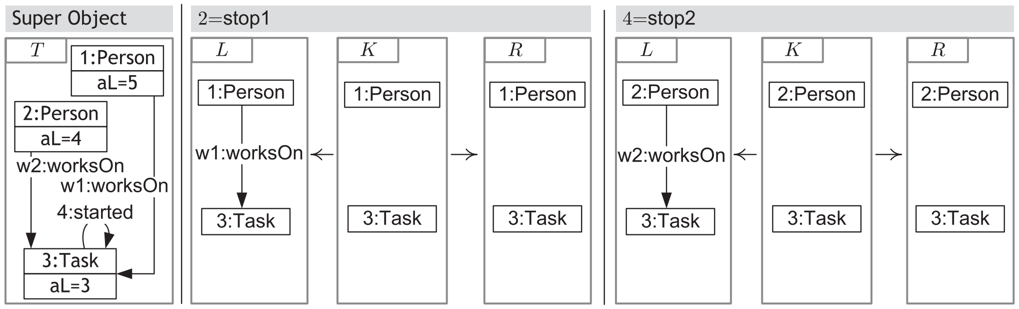


Fig. 6. Super object T and two rules of process $Prc(d)$

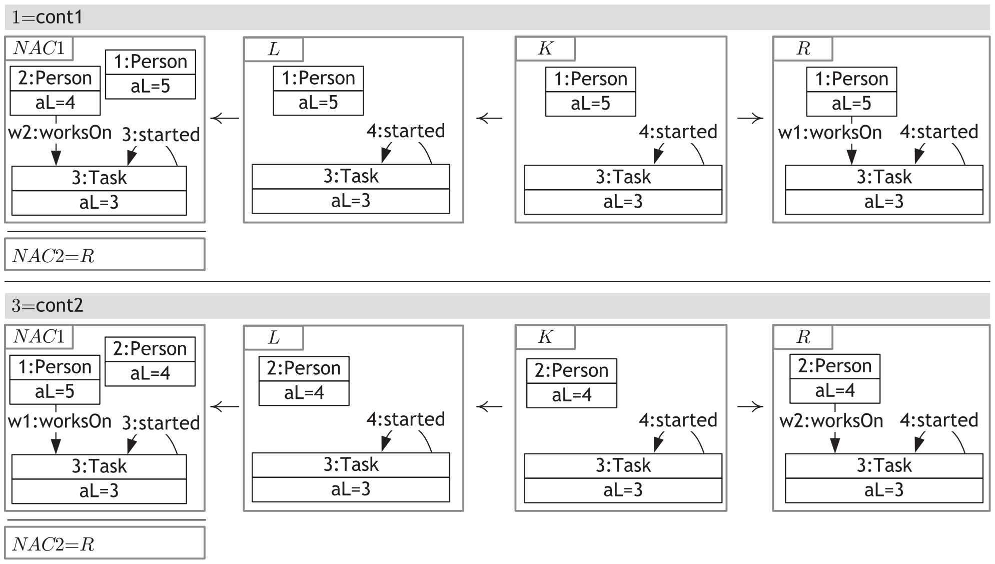


Fig. 7. Further rules of STS $STS(d)$

sequence is finite. For typed attributed graphs, this means that T is finite on the structural part, but the carrier sets of the data algebra for the attribution component may by infinite ($\mathcal{M}$-morphisms in $\mathbf{AGraphs}_{ATG}$ are isomorphisms on the data part).

Remark 3.10. Note that during the construction of $STS(d)$, the set of instantiated NACs for a NAC of a rule p applied in d may be empty, which means that the NAC n cannot be found within T. This would be the case for rule *continueTask* if we replaced the variable lv within the NACs by the constant 4, that is, the NAC pattern would never be present in the transformation sequence. Furthermore, if we require T to be finite, the sets of NACs in $STS(d)$ are finite.

Example 3.11 (derived STS $STS(d)$). For the transformation sequence in Figure 2, the construction of the STS leads to the STS shown in Figures 6 and 7. The transformation sequence d involves the rules ‘continueTask’ and ‘stopTask’, so the derived STS contains the rule occurrences ‘cont1’, ‘cont2’, ‘stop1’ and ‘stop2’.

The process of a transformation sequence d consists of the STS derived from d according to Definition 3.9 together with an embedding v relating the STS with the TS of the given transformation sequence. A process of d induces the complete equivalence class of transformation sequences with respect to permutation equivalence, which we will show in Theorem 3.18.

Definition 3.12 (process of a transformation sequence with NACs). Let

$$d = (G_0 \xRightarrow{q_1,m_1} \dots \xRightarrow{q_n,m_n} G_n)$$

be a NAC-consistent transformation sequence in an $\mathcal{M}$-adhesive transformation system

$$TS = (P_{TS}, \pi_{TS}).$$

The *process*

$$Prc(d) = (STS(d), \mu)$$

of d consists of the derived STS

$$STS(d) = (T, P, \pi)$$

of d together with the mapping

$$\mu : STS(d) \to TS$$

given by $\mu : P \to P_{TS}$,

$$\mu(i) = q_i$$

for each step i of d.

Note that the mapping μ induces a function

$$\mu_\pi : \pi(P) \to \pi_{TS}(P_{TS})$$

mapping each rule in $STS(d)$ to the corresponding rule in TS, where

$$\mu_\pi(\pi(q)) = \pi_{TS}(\mu(q)).$$

Given the process

$$Prc(d) = ((T, P, \pi), \mu)$$

of a derivation d, we will often write $seq(d) \in P^*$ to denote the sequence of production names of $Prc(d)$ that corresponds to the order in which productions are applied in d: it follows from the canonical choice of production names in P (see Definition 3.9) that

$$seq(d) = (1, 2, \dots, n),$$

where n is the length of d.

The notion of processes for transformation sequences corresponds to the notion of processes for Petri nets given by an occurrence net together with a Petri net morphism into the system Petri net. Moreover, as shown in Corradini *et al.* (2008), the process construction yields a *pure* STS, meaning that no rule deletes and produces the same part of a subobject again, that is,

$$L \cap R = K.$$

Table 1. *Relations on rules in an STS*

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	$q_1 <_{wen[i]} q_2$	$0 < i \leqslant \lvert\mathbf{N}_2\rvert \ \wedge\ L_1 \cap \mathbf{N}_2[i]$	$\not\subseteq$	$K_1 \cup L_2$
Weak NAC Disabling	$q_1 <_{wdn[i]} q_2$	$0 < i \leqslant \lvert\mathbf{N}_1\rvert \ \wedge\ \mathbf{N}_1[i] \cap R_2$	$\not\subseteq$	$L_1 \cup K_2$

This terminology is borrowed from the theory of Elementary Net Systems, where a system that does not contain transitions with a self-loop is said to be 'pure'. Therefore, the class of pure STSs can be seen as a generalisation of elementary nets to the setting of $\mathcal{M}$-adhesive transformation systems, and thus, as a generalisation of the Petri net class of occurrence nets.

The following relations between the rules of an STS with NACs specify the possible dependencies between them: the first four relations are discussed in Corradini *et al.* (2008); and the last two were introduced in Hermann (2009).

Definition 3.13 (relations on rules). Let q_1 and q_2 be two rules in an STS

$$\mathcal{S} = (T, P, \pi)$$

with

$$\pi(q_i) = ((L_i, K_i, R_i), \mathbf{N}_i)$$

for $i \in \{1, 2\}$. The relations on rules are defined on P as shown in Table 1.

In words, $q_1 <_{rc} q_2$ (which is read 'q_1 causes q_2 by read causality') if q_1 produces an element that is used but not consumed by q_2. Analogously, $q_1 <_{wc} q_2$ (which is read 'q_1 causes q_2 by write causality') if q_1 produces an element that is consumed by q_2 and $q_1 <_d q_2$ (which is read 'q_1 is deactivated by q_2') precisely when q_1 preserves an element that is consumed by q_2, meaning that q_1 is not applicable afterwards. Furthermore, $q_1 \diamond q_2$ if they overlap only on items that are preserved by both. Finally, $q_1 <_{wen[i]} q_2$ (which is read 'q_1 weakly enables q_2 at i') if q_1 deletes a piece of the NAC $\mathbf{N}[i]$ of q_2; while $q_1 <_{wdn[i]} q_2$ (which is read 'q_2 weakly disables q_1 at i') if q_2 produces a piece of the NAC $\mathbf{N}[i]$ of q_1. It is worth stressing that the relations introduced above are not always transitive.

Example 3.14 (relations of an STS). The rules of $STS(d)$ in Example 3.11 are related by the following dependencies. For write causality, we have 'cont1 $<_{wc}$ stop1' and 'cont2 $<_{wc}$ stop2'. The remaining dependencies are shown in the following table:

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

Definition 3.15 (STS-switch equivalence of sequences disregarding NACs). Let $\mathcal{S} = (T, P, \pi)$ be an *STS*, d be a derivation in $\mathcal{S}$ disregarding NACs and $s = \langle q_1, \ldots, q_n \rangle$ be its corresponding sequence of rule occurrence 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 *STS-switch equivalent* to the sequence s, written $s \overset{sw}{\sim}_{\mathcal{S}} s'$. Switch equivalence $\overset{sw}{\approx}_{\mathcal{S}}$ of rule sequences is the transitive closure of $\overset{sw}{\sim}_{\mathcal{S}}$.

In order to characterise the set of possible permutations of transformation steps of a given transformation sequence, we will now define suitable conditions for permutations of rule occurrences. We say rule sequences s of a derived STS $STS(d)$ are *legal sequences* if they are switch equivalent without NACs to the sequence of rules $seq(d)$ of d and the following condition holds for the NACs. For every NAC $\mathbf{N}[i]$ of a rule q_k, there is either a rule that *deletes* part of $\mathbf{N}[i]$ and is applied *before* q_k or there is a rule that *produces* part of $\mathbf{N}[i]$ and is applied *after* q_{k-1}. In both cases, $\mathbf{N}[i]$ cannot be present when applying q_k because the STS $STS(d)$ is a sort of 'unfolding' of the transformation sequence and every subobject is created at most once and deleted at most once (Corradini *et al.* 2008). Note that the first condition already ensures that each rule name in P occurs exactly once in a legal sequence s.

Definition 3.16 (legal sequence). Let $d = (d_1; \ldots; d_n)$ be a NAC-consistent transformation sequence in an $\mathcal{M}$-adhesive TS, and let

$$STS(d) = (T, P, \pi_N)$$

be its derived STS. 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 the following conditions hold:

(1) $s \overset{sw}{\approx}_{STS(d)} seq(d)$.
(2) For all NACs $\mathbf{N}_k[i]$ of q_k, either

$$\exists\ e \in \{1, \ldots, k-1\} : q_e <_{wen[i]} q_k$$

or

$$\exists\ l \in \{k, \ldots, n\} : q_k <_{wdn[i]} q_l.$$

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

Definition 3.17 (STS equivalence of rule sequences). Let d be a NAC-consistent transformation sequence of an $\mathcal{M}$-adhesive TS and let

$$Prc(d) = (STS(d), \mu)$$

be its derived process. Two sequences s, s' of rule names in $STS(d)$ are STS equivalent, written $s \approx_{STS(d)} s'$, if they are legal sequences with respect to d. The set of all STS-equivalent sequences of $Prc(d)$ is given by

$$Seq(d) = \{s \mid s \approx_{STS(d)} seq(d)\}.$$

Moreover, the specified class of transformation sequences of $Seq(d)$ is given by

$$Trafo(s) = [trafo_{STS(d)}(s)]_{\cong}$$

for single sequences and

$$Trafo(Seq(d)) = \bigcup_{s \in Seq(d)} Trafo(s)$$

for the complete set.

Theorem 3.18 (characterisation of permutation equivalence based on STSs). Given the process $Prc(d)$ of a NAC-consistent transformation sequence d.

(1) The class of permutation-equivalent transformation sequences of d coincides with the set of derived transformation sequences of the process $Prc(d)$ of d:

$$\pi\text{-}Equ(d) = Trafo(Seq(d)).$$

(2) The mapping *Trafo* defines a bijective correspondence between STS-equivalent sequences of rule names and permutation-equivalent transformation sequences:

$$Trafo : Seq(d) \xrightarrow{\sim} (\pi\text{-}Equ(d))/_{\cong}.$$

Proof. Let d be a NAC-consistent transformation sequence in an $\mathcal{M}$-adhesive TS and let $Prc(d) = (\mathcal{S}, \mu)$ be the process of d with $\mathcal{S} = (T, P, \pi)$. We have to show that each STS-equivalent rule sequence s' of $seq(d)$ in $\mathcal{S}$ defines a permutation-equivalent transformation sequence $trafo_{STS(d)}(s')$ of d, and, conversely, for each permutation-equivalent transformation sequence d' of d, there is an STS-equivalent rule sequence s' of $seq(d)$ in $\mathcal{S}$ such that $d' \cong trafo_{STS(d)}(s')$.

$$\forall\, s' \in P^* : s' \approx_{STS(d)} seq(d) \Rightarrow trafo_{STS(d)}(s') \overset{\pi}{\approx} d \tag{1}$$

$$\forall\, d' : d' \overset{\pi}{\approx} d \Rightarrow \exists\, s'.s' \approx_{STS(d)} seq(d) \wedge trafo_{STS(d)}(s') \cong d'. \tag{2}$$

The proof of Hermann (2009, Theorem 1) shows (1) and (2) for the case of adhesive transformation systems with NACs and monomorphic matches using the intersection and union operations on subobjects and distributivity. The operations are available for $\mathcal{M}$-adhesive transformation systems with effective unions, which we require by our Assumption 2.8: intersection is given by Fact 3.3 and distributivity is shown by Fact 3.5. Thus, (1) and (2) hold for $\mathcal{M}$-adhesive transformation systems with $\mathcal{M}$-matches.

Finally, by Definition 3.17, we have that

$$d' \in \mathit{Trafo}(\mathit{Prc}(d))$$

is equivalent to

$$d' \cong \mathit{trafo}_{STS(d)}(s')$$

and

$$s' \approx_{STS(d)} \mathit{seq}(d).$$

Using (1) and (2) together with Definition 2.19, we can derive

$$\pi\text{-}\mathit{Equ}(d) = \mathit{Trafo}(\mathit{Prc}(d)). \qquad \square$$

According to Theorem 3.18, the construction of the process $\mathit{Prc}(d)$ of a transformation sequence d specifies the equivalence class of all transformation sequences that are permutation equivalent to d. In the next section, we present an efficient analysis technique for processes based on Petri nets.

4. Analysis of permutation equivalence based on Petri nets

Using the process of a transformation sequence given by an STS as a basis, in this section we will present the construction of its *dependency net*, which is given by a P/T Petri net that just specifies the dependencies between the transformation steps. All details about the internal structure of the objects and the transformation rules are excluded, which enables us to increase the efficiency of the analysis of permutation equivalence (see Remark 2.21). The names of the generated places of the dependency net are composed of constant symbols and numbers, where constant symbols s are denoted by $\mathtt{s}$. In this section, we will use the monoidal notation of P/T Petri nets according to Meseguer and Montanari (1990) and ISO/IEC 15909-1:2004 (ISO/IEC 2004), which is equivalent to the classical notation of P/T Petri nets (Reisig 1985) – see Appendix B for a brief review of both notations.

Definition 4.1 (dependency net *DNet* of a transformation sequence). Let d be a NAC-consistent transformation sequence of an $\mathcal{M}$-adhesive TS and

$$STS(d) = (T, P, \pi)$$

be the generated STS of d. Let

$$s = \mathit{seq}(d) = \langle q_1, \ldots, q_n \rangle$$

be the sequence of rule names in $STS(d)$ according to the steps in d. The dependency net of d is given by the marked Petri net

$$\mathit{DNet}(d) = (\mathit{Net}, M)$$

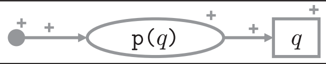
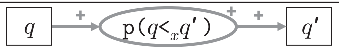
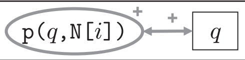
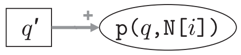
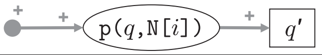
$STS(d) = (T,P,\pi)$	$DNet(d) = ((PL,TR,pre,post),M)$
1. For each $q \in P$	
2. For all $q,q' \in P$, $q <_x q'$, $x \in \{rc,wc,d\}$	
3. For all $q \in P$ with NACs $\mathbf{N}$ and for all $0 < i \leq \lvert\mathbf{N}\rvert$ with $q \not<_{wdn[i]} q$	
a) For $\mathbf{N}[i]$ of q	
b) For all $q' \in P$: $q' <_{wen[i]} q$	
c) For all $q' \in P$: $q <_{wdn[i]} q'$	

Fig. 8. Visualisation of the construction of the Petri net

where:

— $Net = (PL, TR, pre, post)$ with:

$$TR = P = \{i \mid 1 \leqslant i \leqslant |P|\}$$

$$\begin{aligned} PL = {} & \{\mathtt{p}(q) \mid q \in TR\} \\ & \cup \{\mathtt{p}(q' <_x q) \mid q, q' \in TR, x \in \{rc, wc, d\}, q' <_x q\} \\ & \cup \{\mathtt{p}(q, \mathtt{N}[i]) \mid q \in TR, \pi(q) = ((L_q, K_q, R_q), \mathbf{N}), 0 < i \leqslant |\mathbf{N}|, q \not<_{wdn[i]} q\} \end{aligned}$$

$$pre(q) = \mathtt{p}(q) \oplus \sum_{\substack{q' <_x q \\ x \in \{rc,wc,d\}}} \mathtt{p}(q' <_x q) \oplus \sum_{\substack{q' <_{wdn[i]} q \\ q' \neq q}} \mathtt{p}(q', \mathtt{N}[i]) \oplus \sum_{\mathtt{p}(q,\mathtt{N}[i]) \in PL} \mathtt{p}(q, \mathtt{N}[i])$$

$$post(q) = \sum_{\substack{q <_x q' \\ x \in \{rc,wc,d\}}} \mathtt{p}(q <_x q') \oplus \sum_{q <_{wen[i]} q'} \mathtt{p}(q', \mathtt{N}[i]) \oplus \sum_{\mathtt{p}(q,\mathtt{N}[i]) \in PL} \mathtt{p}(q, \mathtt{N}[i])$$

— $M = \sum_{q \in TR} \mathtt{p}(q) \oplus \sum_{\substack{q' <_{wdn[i]} q \\ \mathtt{p}(q',\mathtt{N}[i]) \in PL}} \mathtt{p}(q', \mathtt{N}[i]).$

Figure 8 shows how the dependency net is constructed algorithmically – the construction steps are performed in the order they appear in the table. Each step is visualised as a rule, where grey lines and plus signs mark the elements to be inserted. The matched context that is preserved by a rule is marked by black lines, for example, in Step 2 the new place '$p(q <_x q')$' is inserted between the already existing transitions q and q'. The tokens of the initial marking of the net are represented by bullets that are connected to their places by arcs. In the first step, each rule q of the STS is encoded as a transition and connected to a marked place, which prevents the transition from firing more than once. In Step 2, a new place is created between each pair of transitions in each of the relations $<_{rc}$, $<_{wc}$ and $<_d$

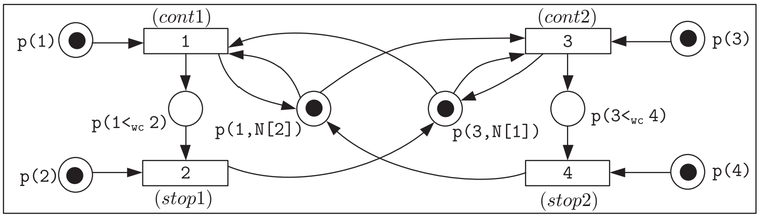


Fig. 9. Dependency net $DNet(d)$ as a Petri net

to enforce the corresponding dependency. The rest of the construction is concerned with places corresponding to NACs and can, in general, contain several tokens. Each token in such a place represents *the absence* of a piece of the NAC, so if the place is empty, the NAC is complete.

In this case, by Step (3a), the transition cannot fire. Consistent with this intuition, if $q' <_{wen[i]} q$, that is, transition q' consumes part of the NAC $\mathbf{N}[i]$ of q, then, by Step (3b), q' produces a token in the place corresponding to $\mathbf{N}[i]$. Symmetrically, if $q <_{wdn[i]} q'$, that is, q' produces part of NAC $\mathbf{N}[i]$ of q, then, by Step (3c), q' consumes a token from the place corresponding to $\mathbf{N}[i]$. Notice that each item of a NAC is either already in the start graph of the transformation sequence or produced by a single rule. If a rule generates part of one of its NACs, say $\mathbf{N}[i]$ ($q <_{wdn[i]} q$), then, by the acyclicity of $Prc(d)$, the NAC $\mathbf{N}[i]$ cannot be completed before the firing of q: hence we ignore it in the third step of the construction of the dependency net. Examples of such weakly self-disabling rules are ($1 = cont1$) and ($3 = cont2$) in Figure 7, where the specific NACs coincide with the right-hand sides of the rules ($NAC2 = R$).

Note that the constructed net is not, in general, a safe net because the places for the NACs can contain several tokens. Nevertheless, it is a bounded P/T net. The bound is the maximum of one and the maximal number of adjacent edges at a NAC place minus two.

Example 4.2 (dependency net). Consider the transformation sequence d in Figure 2 from Example 2.13 and its derived STS in Example 3.11. The marked Petri net in Figure 9 is the dependency net $DNet(d)$ according to Definition 4.1. 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 transformation step of d, and `p(3,N[1])` for the third transformation 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 start, transitions `1` and `2` ($cont1$ and $cont2$) are enabled. The firing sequences according to the transformation sequences d and d' in Figures 2 and 4 can be executed, and they are the only complete firing sequences of this net. Thus, the net specifies exactly the transformation sequences that are permutation equivalent to d.

We will now show that we can exploit the constructed Petri net $DNet(d)$ to characterise STS equivalence of sequences of rule occurrences by Theorem 4.4. Note that according to Definition 4.1, each sequence s of rule names in the STS of $Prc(d)$ 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 of the dependency net back to the STS. Note that the way we build the dependency net (Definition 4.1) ensures, by construction, that each transition can fire at most once.

Definition 4.3 (transition complete firing sequences). A firing sequence of a Petri net is said to be *transition complete* if each transition of the net occurs exactly once. The set of transition complete firing sequences of a dependency net $DNet(d)$ is denoted by $FSeq(DNet(d))$.

Theorem 4.4 (characterisation of STS equivalence based on Petri nets). Given the process $Prc(d)$ and the dependency net $DNet(d)$ of a NAC-consistent transformation sequence d of an $\mathcal{M}$-adhesive transformation system with $\mathcal{M}$-matches, the class of STS-equivalent sequences of $seq(d)$ coincides with the set of transition complete firing sequences in the dependency net $DNet(d)$, that is,

$$Seq(d) = FSeq(DNet(d)).$$

Remark 4.5 (bijective correspondence). Analogously to Theorem 3.18, there is also a bijective correspondence between STS sequences and transition complete firing sequences, which is in this case given directly by the identity function

$$id : Seq(d) \xrightarrow{\sim} FSeq(DNet(d)).$$

In order to prove Theorem 4.4, we will need Fact 4.6, which shows that STS-switch equivalence disregarding NACs of rule sequences respects the partial order of the relations '$<_{rc}, <_{wc}$' and '$<_d$', and *vice versa*. This is important for showing that the causal dependencies are correctly reflected within the dependency net, where firing sequences correspond to linearisations.

Fact 4.6 (linearisation). Let d be a NAC-consistent transformation sequence of an $\mathcal{M}$-adhesive TS, let $\mathcal{S} = STS(d)$ be the generated STS of d, and let $s = \langle s_1, \ldots, s_n \rangle$ be a permutation of $seq(d)$. Then

$$s \overset{sw}{\approx}_{\mathcal{S}} seq(d) \text{ if and only if } \forall\, i, j \in \{1, \ldots, n\},\ x \in \{rc, wc, d\} : s_i <_x s_j \Rightarrow i < j.$$

Proof of Theorem 4.4. Let

$$Prc(d) = (STS(d), \mu)$$

and

$$\mathcal{S} = STS(d).$$

We have to show that

$$s \approx_{STS(d)} seq(d)$$

if and only if s is a transition complete firing sequence of $DNet(d)$.

Let $seq(d) = \langle q_1, \ldots, q_n \rangle$ and $s = \langle s_1, \ldots, s_n \rangle$.

($\Rightarrow$):

By Definition 3.17, s is a legal sequence with respect to d in $STS(d)$. We show that s

is a transition complete firing sequence of $DNet(d)$. Since s is a permutation of $seq(d)$ in $STS(d)$ we know:

$$\text{Each transition occurs exactly once in } s. \qquad (*)$$

Consider the transition name $tr = s_m$ in s and the claimed firing step

$$M_m \xrightarrow{tr} M_{m+1}.$$

We will now check the activation of tr in M_m, that is,

$$M_m \geqslant pre(tr)$$

according to Definition 4.1. Let

$$pre(tr) = \sum_{pl \in PL} \lambda_{pl} \cdot pl.$$

We will now consider cases for pl:

— $pl = \mathtt{p}(q)$:
So $tr = q$ and $\lambda_{pl} = 1$. By definition, this place is initially marked with one token and there is no other transition connected to this place. From $(*)$, this token is available in M_m.

— $pl = \mathtt{p}(q{<_x}q'), x \in \{rc, wc, d\}$:
So $tr = q'$ and $\lambda_{pl} = 1$. By Definition 4.1, we then have $post(q) \geqslant pl$ and pl is not in the pre-domain of any transition other than $tr = q'$. By Fact 4.6, we have that q occurs before q' in s and by $(*)$ we know that q' was not fired already. Thus, $M_m \geqslant pl$.

— $pl = \mathtt{p}(q,\ \mathtt{N}[i])$:
For the initial marking M, we know by Definition 4.1 that $M \geqslant d \cdot pl$, with d being the amount of weak disabling causes, that is,

$$d = |DC|$$
$$DC = \{q_l \mid q, q' \in P, q <_{wdn[i]} q_l\}.$$

Moreover, by Definition 4.1, we know that $q \not<_{wdn[i]} q$. We now consider subcases:

– $q \neq tr$:
Let $q' = tr$. By Definition 4.1, we have that $\lambda_{pl} = 1$ and $q <_{wdn[i]} q'$. The only transition tr' in $TR \setminus DC$ with $pre(tr') \geqslant pl$ is q, and q consumes and produces one token. Each of the transitions in DC consumes exactly one token, and in sum, they consume exactly d tokens and from $(*)$, each transition occurs exactly once in s. Therefore, $M_m \geqslant pl$ because $tr = q'$ has not fired already according to $(*)$.

– $q = tr$:
So $\lambda_{pl} = 1$. Let $s_k = q$, that is, q occurs in s at position k. By Definition 3.16, there is one preceding rule occurrence $q' = s_e$ in s with

$$q' = s_e <_{wen[i]} s_k = q$$

or there is one subsequent rule occurrence $q' = s_l$ in s with

$$q = s_k <_{wdn[i]} s_l = q'$$

(because $q \not<_{wdn[i]} q$). Using (∗), this means that for the first case,

$$M_m \geqslant d \cdot pl + 1 - d \cdot pl = pl,$$

and for the second case,

$$M_m \geqslant d \cdot pl - (d-1)pl = pl.$$

($\Leftarrow$):

We will assume that s is a transition complete firing sequence of $DNet(d)$ and show that s is a legal sequence with respect to d in $STS(d)$. First, the fact that s is a transition complete firing sequence implies that each transition tr occurs exactly once.
We will now show that the two conditions in Definition 3.16 hold:

— Condition 1: $s \overset{sw}{\approx}_{\mathcal{S}} seq(d)$.
By Fact 4.6 this condition is equivalent to

$$\forall\, i, j \in \{1, \ldots, n\}, x \in \{rc, wc, d\} : s_i <_x s_j \Rightarrow i < j. \qquad (**)$$

According to Definition 4.1, there is exactly one initially unmarked place

$$pl = \mathtt{p}(q{<_x}q')$$

for each pair (q, q') with $q <_x q'$ and $x \in \{rc, wc, d\}$. This implies that for $s_i = q$ and $s_j = q'$, the transition s_i produces exactly one token and s_j consumes exactly one token from this place, and there is no other transition connected to this place. Hence, the condition is ensured because transition s_j is not activated before s_i has been fired.

— Condition 2: For all NACs $\mathbf{N}_k[i]$ of $s_m = q_k$, either

$$\exists\, e \in \{1, \ldots, m-1\} : s_e <_{wen[i]} s_m$$

or

$$\exists\, l \in \{m, \ldots, n\} : s_m <_{wdn[i]} s_l.$$

Consider a NAC $\mathbf{N}_k[i]$ of $q_k = s_m$ and consider cases:

– $q_k <_{wdn[i]} q_k$:
So we have $l = m$ for the above condition.

– $q_k \not<_{wdn[i]} q_k$:
So there is the place $\mathtt{p}(k,\mathtt{N}[i])$ such that the transition $s_m = q_k$ consumes exactly one token from that place. Consider the firing step

$$M_m \xrightarrow{s_m} M_{m+1}$$

according to s. Since $s_m = q_k$ has fired according to this step, there was a token on $\mathtt{p}(k,\mathtt{N}[i])$ in the marking M_m. The initial marking contains d tokens for this place, where d is the amount of weak disabling causes, that is,

$$\begin{aligned} d &= |DC| \\ DC &= \{q_{l'} \mid q_k <_{wdn[i]} q_{l'}\}. \end{aligned}$$

Let

$$EC = \{q_{e'} \mid q_{e'} <_{wen[i]} q_k\}$$

be the set of weak enabling causes of q_k for $\mathbf{N}_k[i]$. In order to show a contradiction, we will now assume that Condition 2 of Definition 3.16 does not hold so that all $q_{l'}$ 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 $\mathtt{p}(k,\mathtt{N}[i])$ and none of the transitions that precede q_k have produced a token on this place. Therefore, there is no token left on $\mathtt{p}(k,\mathtt{N}[i])$, which contradicts the firing of $s_m = q_k$, so Condition 2 holds. □

In order to compute the set of all permutation-equivalent transformation sequences for a given sequence, we can now combine the results presented so far to give our fourth main result, which shows that the analysis of permutation equivalence can be completely performed on the dependency net $DNet(d)$.

Theorem 4.7 (analysis of permutation equivalence based on Petri nets). Given the process $Prc(d)$ and the dependency net $DNet(d)$ of a NAC-consistent transformation sequence d:

(1) The class of permutation-equivalent transformation sequences of d coincides with the set of derived transformation sequences using $DNet(d)$, that is,

$$\pi\text{-}Equ(d) = Trafo(FSeq(DNet(d))).$$

(2) The mapping *Trafo* according to Definition 3.17 defines a bijective correspondence between transition complete firing sequences and permutation-equivalent transformation sequences, that is,

$$Trafo : FSeq(DNet(d)) \xrightarrow{\sim} (\pi\text{-}Equ(d))/_{\cong}.$$

Proof. By combining the characterisations of Theorems 3.18 and 4.4, we derive the equality

$$\pi\text{-}Equ(d) = Trafo(FSeq(DNet(d))),$$

and the bijection

$$Trafo : FSeq(DNet(d)) \xrightarrow{\sim} (\pi\text{-}Equ(d))/_{\cong}$$

is given by

$$Trafo : Seq(d) \xrightarrow{\sim} (\pi\text{-}Equ(d))/_{\cong}$$

of Theorem 3.18 with

$$Seq(d) = FSeq(DNet(d))$$

in Theorem 4.4. □

Remark 4.8 (analysis of permutation equivalence). We will now describe how our results can be used to carry out an efficient analysis of permutation equivalence, that is, in order to generate the complete set of permutation equivalent transformation sequences for a given sequence and to check the permutation equivalence of specific sequences. Given a NAC-consistent transformation sequence with general matches and NAC-schemata, we can first reduce the analysis problem to the derived instantiated transformation sequence with $\mathcal{M}$-matches and standard NACs according to Theorem 2.23 and Remark 2.24. According to Theorem 4.7, we can perform the analysis of permutation equivalence based on Petri nets by first constructing the dependency net $DNet(d)$. In order to generate all permutation-equivalent sequences, we construct the complete reachability graph of $DNet(d)$, where each path specifies one permutation-equivalent transformation sequence up to isomorphism. If only specific reorderings of the transformation steps need to be checked, we just check that the corresponding firing sequences are executable in $DNet(d)$.

The dependency net $DNet(d)$ is a compact representation of the equivalence class π-$Equ(d)$ specified by the process of a transformation sequence d. Moreover, the analysis of permutation equivalence based on the dependency net shows significant advantages with respect to efficiency, as shown in Remark 2.21.

5. Related work

Negative application conditions (NACs) for transformation systems based on the double-pushout approach (DPO) were introduced in Habel *et al.* (1996) for graph transformation systems and generalised in Ehrig *et al.* (2006) for adhesive transformation systems (in the weak-HLR variant). The definition of NAC-schemata and their satisfaction for non-injective matches was inspired by a construction proposed in Kastenberg *et al.* (2006), and it exploits the notion of extremal $\mathcal{E}$-$\mathcal{M}$-factorisation introduced in Braatz *et al.* (2010).

The definition of sequential independence for transformation steps with NACs goes back to Habel *et al.* (1996) for graph transformation, and was generalised to adhesive systems in Lambers *et al.* (2008) and Lambers (2009). Deterministic processes for DPO graph transformation systems were introduced in Corradini *et al.* (1996) and characterised as occurrence grammars in Baldan (2000): these concepts generalise the corresponding notions for Petri nets (Reisig 1985), and were generalised further in Baldan *et al.* (2006) to adhesive transformation systems. In fact, the construction of a process from a transformation sequence presented in Section 3 is a generalisation to the case with NACs of a corresponding construction proposed in Corradini *et al.* (2008), which also introduced Subobject Transformation Systems, and showed them to be related to Adhesive Transformation Systems in the same way as Elementary Net Systems (Rozenberg and Engelfriet 1996) are related to Place/Transition Petri nets.

Compared with Hermann (2009) and Hermann *et al.* (2010), in the current paper, we have generalised the approach from transformation systems based on the category of graphs to those based on an arbitrary $\mathcal{M}$-adhesive category. Furthermore, we have considered general, and possibly non-monic, matches of the left-hand sides of rules into the objects to be transformed. Petri nets with inhibitor arcs (or *inhibitor nets*) (Janicki and Koutny 1995; Busi and Pinna 1999; Kleijn and Koutny 2004; Baldan *et al.* 2004) form another computational model that is closely related to transformation systems with NACs. In such nets, a transition cannot fire if there are tokens on its *inhibitor places*, that is, on the places that are linked to it with inhibitor arcs[†]. Therefore these places play a role that is conceptually similar to NACs.

Work on the semantics of inhibitor nets distinguishes between the *a posteriori* semantics (as in Busi and Pinna (1999) and Baldan *et al.* (2004)), where the inhibitor places of a transition must be empty both before and after the transition is fired, and the *a priori* semantics (Janicki and Koutny 1995; Kleijn and Koutny 2004), where they have to be empty only before the transition fires: in the latter case, a transition can generate a token in an inhibitor place. Transformation rules often use NACs to ensure that a certain structure does not exist in the current state before generating it, as in the case for rule 'continueTask' of Example 2.13; this means that an '*a priori*' semantics is implicitly assumed in our framework. However, while the semantics of Janicki and Koutny (1995) and Kleijn and Koutny (2004) are based on *step sequences*, which allow the parallel firing of several enabled transitions, in our approach, an '*a priori*' *step* semantics would be unsound, so we only consider *linear* transformation sequences. In fact, the '*a priori*' step semantics would allow two instances of the rule 'continueTask' to fire simultaneously on the start graph (that is, graph G_0 of Figure 2), which would lead to an inconsistent state (according to the intended operational semantics of the system modelled in Figure 1), where two people work simultaneously on the same task.

It is worth stressing that the proposed notion of permutation equivalence would also be original in the framework of inhibitor nets. In fact, if we encode the system of Example 2.13 into an inhibitor net (by forgetting the graphical structure), the standard semantics for such nets would not consider the firing sequences corresponding to the two transformation sequences d of Figure 2 and d' of Figure 4 to be equivalent. Whether permutation equivalence would be meaningful for firing sequences of inhibitor nets and could be the basis of a new semantical framework for such nets is an interesting topic for future work.

Note that we could have used some sort of inhibitor arcs to model the inhibiting effect of NACs in the dependency net of a transformation sequence in Section 4. 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 an encoding of the process of a transformation sequence into a standard marked P/T net.

6. Conclusions and future work

In this paper, we have introduced the concept of permutation equivalence for transformation systems with negative application conditions (NACs) in $\mathcal{M}$-adhesive categories. Permutation equivalence is coarser than switch equivalence with NACs and

[†] For simplicity, we only consider the case of unweighted inhibitor arcs.

has interesting applications in the area of business processes (Brandt *et al.* 2009). Formally, we are able to define processes of $\mathcal{M}$-adhesive transformation systems based on subobject transformation systems inspired by processes for Petri nets (Rozenberg and Engelfriet 1996) and adhesive rewriting systems (Baldan *et al.* 2006).

Our main results show that processes represent equivalence classes of permutation-equivalent transformation sequences. Moreover, they can be analysed efficiently by complete firing sequences of a Petri net, which can be constructed effectively as a dependency net of a given transformation sequence. We have illustrated most of the constructions and results using a case study of a typed attributed graph transformation system using the new concept of NAC-schemata. Tool support for the analysis is available through the tool AGT-M (Hermann *et al.* 2010; Brandt *et al.* 2009), which is based on Wolfram Mathematica and provides the construction of the STS, the dependency net and the generation of the reachability graph for a given transformation sequence.

We are currently developing and analysing the interleaving semantics of processes of $\mathcal{M}$-adhesive transformation systems from a more algebraic point of view based on the construction and decomposition of concurrent transformation steps with NACs. First results indicate that the notion of permutation equivalence can be characterised by the underlying equivalence of these algebraic compositions and decompositions.

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. Furthermore, the notion of permutation equivalence can be extended to the more general case of nested application conditions (Habel and Pennemann 2009), which will probably lead to an extended concept for processes based on STSs including nested application conditions. Further efficiency improvements could be obtained by observing the symmetries occurring in the P/T Petri net, and applying symmetry reduction techniques to 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.

Appendix A. Category of typed attributed graphs

In this appendix we review the main constructions for the $\mathcal{M}$-adhesive category of typed attributed graphs ($\mathbf{AGraphs}_{ATG}, \mathcal{M}$) according to Ehrig *et al.* (2006).

An attributed graph consists of an extended directed graph for the structural part, called the E-graph, together with an algebra for the specification of the carrier sets of the value nodes. An E-graph extends a directed graph by additional attribute value nodes and edges for the attribution of structural nodes and edges.

Definition A.1 (E-graph and E-graph morphism). An *E-graph* G with

$$G = (V_G, V_D, E_G, E_{NA}, E_{EA}, (source_j, target_j)_{j \in \{G,NA,EA\}})$$

consists of the sets:

— V_G and V_D, which are called the graph and data nodes (or vertices), respectively;

— E_G, E_{NA} and E_{EA}, which are called the graph, node attribute and edge attribute edges, respectively; and
— the source and target functions
  – $source_G : E_G \to V_G, target_G : E_G \to V_G$ for graph edges;
  – $source_{NA} : E_{NA} \to V_G, target_{NA} : E_{NA} \to V_D$ for node attribute edges; and
  – $source_{EA} : E_{EA} \to E_G, target_{EA} : E_{EA} \to V_D$ for edge attribute edges:

$$\begin{tikzcd}
 & E_G \arrow[rr, bend left=5, "source_G"] \arrow[rr, bend right=5, "target_G"'] & & V_G & \\
E_{EA} \arrow[ur, "source_{EA}"] \arrow[drr, "target_{EA}"'] & & & & E_{NA} \arrow[ul, "source_{NA}"'] \arrow[dll, "target_{NA}"] \\
 & & V_D & &
\end{tikzcd}$$

Consider the E-graphs G^1 and G^2 with

$$G^k = (V_G^k, V_D^k, E_G^k, E_{NA}^k, E_{EA}^k, (source_j^k, target_j^k)_{j\in\{G,NA,EA\}})$$

for $k = 1, 2$. An E-graph morphism $f : G^1 \to G^2$ is a tuple

$$(f_{V_G}, f_{V_D}, f_{E_G}, f_{E_{NA}}, f_{E_{EA}})$$

with

$$f_{V_i} : V_i^1 \to V_i^2$$
$$f_{E_j} : E_j^1 \to E_j^2$$

for

$$i \in \{G, D\}$$
$$j \in \{G, NA, EA\}$$

such that f commutes with all source and target functions, for example

$$f_{V_G} \circ source_G^1 = source_G^2 \circ f_{E_G}.$$

The carrier sets of attribute values that form the single set V_D of an E-graph are defined by an additional data algebra D, which also specifies the operations for generating and manipulating data values. The carrier sets D_s of D contain the data elements for each sort $s \in S$ according to a data signature

$$DSIG = (S_D, OP_D).$$

These carrier sets are combined by disjoint union and form the set V_D of data elements.

Definition A.2 (attributed graph and attributed graph morphism). Let

$$DSIG = (S_D, OP_D)$$

be a data signature with attribute value sorts $S'_D \subseteq S_D$. An attributed graph

$$AG = (G, D)$$

consists of an E-graph G together with a $DSIG$-algebra D such that

$$\dot\cup_{s\in S'_D} D_S = V_D.$$

For two attributed graphs

$$AG^1 = (G^1, D^1)$$
$$AG^2 = (G^2, D^2),$$

an attributed graph morphism

$$f : AG^1 \to AG^2$$

is a pair $f = (f_G, f_D)$ with an E-graph morphism

$$f_G : G^1 \to G^2$$

and an algebra homomorphism

$$f_D : D^1 \to D^2$$

such that

$$\begin{tikzcd} D^1_s \arrow[r, "f_{D,s}"] \arrow[d, hook] \arrow[dr, phantom, "(1)"] & D^2_s \arrow[d, hook] \\ V^1_D \arrow[r, "f_{G,V_D}"] & V^2_D \end{tikzcd}$$

commutes for all $s \in S'_D$, where the vertical arrows are inclusions.

The category of typed attributed graphs $\mathbf{AGraphs}_{ATG}$ has as objects all attributed graphs with a *typing morphism* to the attributed graph ATG (type graph) and as arrows all attributed graph morphisms preserving the typing. Ehrig *et al.* (2006) showed that the category $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ is an adhesive HLR category, where the distinguished class of monomorphisms $\mathcal{M}$ contains all monomorphisms that are isomorphisms on the data part. For this reason, all results for adhesive HLR transformation systems presented in Ehrig *et al.* (2006) are valid. Since $\mathcal{M}$-adhesive categories (Ehrig *et al.* 2010) are a slight generalisation of weak adhesive and adhesive HLR categories, the category $(\mathbf{AGraphs}_{ATG}, \mathcal{M})$ is an $\mathcal{M}$-adhesive category.

Appendix B. Petri nets in monoidal notation

In this appendix we will briefly recall the classical notion of place/transition nets (P/T Petri nets) according to Reisig (1985) and its equivalent representation in monoidal notation according to Meseguer and Montanari (1990). We use the monoidal notation in Section 4 for the construction of the dependency net of a transformation sequence. Note that this notation forms a special case of the monoidal notation for the more general high-level Petri nets according to ISO/IEC 15909-1:2004 (ISO/IEC 2004).

Petri nets are a formal and graphical formalism for the specification of parallel and distributed systems and are used for the analysis of the concurrent behaviour of such systems. The main idea is that places specify locations, tokens on places specify resources available at these locations or, alternatively, control events, while transitions specify the possible actions of the system, which are dependent on the resources and control conditions.

Definition B.1 (P/T Petri net in classical notation). A P/T Petri net in classical notation is given by a tuple

$$N = (P, T, F, K, W),$$

consisting of a set of places P, a set of transitions T, a flow relation

$$F \subseteq (P \times T) \uplus (T \times P),$$

a capacity function $K : P \to \mathbb{N}_\omega$ specifying the (possibly unbounded) capacity for each place, and the weight function $W : F \to \mathbb{N}^+$ assigning the relevant weight to each edge of the flow relation.

A marking M for a P/T Petri net

$$N = (P, T, F, K, W)$$

is given by a function $M : P \to \mathbb{N}$ assigning each place an amount of token, where $M(p) \leqslant K(p)$ for each place p. For any transition $t \in T$ of a P/T-Petri net $N = (P, T, F, K, W)$, the pre-domain is denoted by

$$\bullet t = \{p \mid (p, t) \in F\}$$

and the post-domain by

$$t\bullet = \{p \mid (t, p) \in F\}.$$

A transition $t \in T$ is *M-activated*, if

$$\forall\, p \in \bullet t : M(p) \geqslant W(p, t)$$

and

$$\forall\, p \in t\bullet : M(p) + W(t, p) \leqslant K(p).$$

Finally, a firing step $M \xrightarrow{t} M'$ of N with initial marking M exists if transition t is M-activated. The resulting marking M' is given by

$$M'(p) = \begin{cases} M(p) - W(p,t) \text{ for } p \in \bullet t \setminus t\bullet, \\ M(p) + W(t,p) \text{ for } p \in t\bullet \setminus \bullet t, \\ M(p) - W(p,t) + W(t,p) \text{ for } p \in t\bullet \cap \bullet t, \\ M(p), \text{ otherwise.} \end{cases}$$

Meseguer and Montanari (1990) and ISO/IEC 15909-1:2004 (ISO/IEC 2004) say P/T Petri nets can be specified equivalently using the monoidal notation. This notation is based on a power set or monoid construction. Note that capacities are not explicitly specified, but can be encoded by corresponding complementary places. The main idea of the monoidal notation is to specify the pre- and post-domain of each transition by a multi-set of places using the concept of a monoid.

Definition B.2 (P/T Petri net in monoidal notation). A P/T Petri net in monoidal notation is given by

$$N = (P, T, pre, post)$$

consisting of a set P of places, a set T of transitions and the mappings $pre, post : T \to P^{\oplus}$ specifying the pre- and post-domain of each transition, where $(P^{\oplus}, \oplus, \lambda)$ is the free commutative monoid over P.

$$\begin{tikzcd} T \arrow[r, shift left, "pre"] \arrow[r, shift right, "post"'] & P^{\oplus} \end{tikzcd}$$

A marking M for a P/T Petri net

$$N = (P, T, pre, post)$$

is given by an element $M \in P^{\oplus}$ of the carrier set $P^{\oplus}$ of the monoid $(P^{\oplus}, \oplus, \lambda)$. A transition $t \in T$ is *M-activated* if $pre(t) \leqslant M$. Finally, a firing step $M \xrightarrow{t} M'$ of N with initial marking M exists if transition t is M-activated and the resulting marking M' is given by

$$M' = M \ominus pre(t) \oplus post(t).$$

See Example 4.2 for an example of a place/transition net and its firing behaviour.

Appendix C. Proofs of technical results

In this appendix we give the proofs for Facts 2.15, 2.22, 3.3, 3.5 and 4.6.

Fact 2.15 (compatibility of applicability and NAC-consistency with instantiation).

Proof. If we ignore the NACs, we have that the transformation step *via* p' can be composed with the diagrams (3) and (4) in

$$\begin{tikzcd} L \arrow[d, tail, "e"'] \arrow[dd, bend left, "f_2"] \arrow[dr, phantom, "(3)"] & K \arrow[l, tail] \arrow[r, tail] \arrow[d, tail, "k_e"'] \arrow[dd, bend left] \arrow[dr, phantom, "(4)"] & R \arrow[d, tail, "e^*"'] \arrow[dd, bend left, "f_2^*"] \\ L' \arrow[d, tail, "m_2"'] \arrow[dr, phantom, "(5)"] & K' \arrow[l, tail] \arrow[r, tail] \arrow[d, tail] \arrow[dr, phantom, "(6)"] & R' \arrow[d, tail, "m_2^*"'] \\ G_2 & D_2 \arrow[l, tail] \arrow[r, tail] & H_2 \end{tikzcd}$$

according to Definition 2.14, and this leads to a transformation step *via* p and match f_2. Conversely, for a transformation step *via* p and match m_2, we can conclude that K' is isomorphic to the pullback of

$$(L' \rightarrowtail G_2 \leftarrowtail D_2)$$

using the $\mathcal{M}$-pushout–pullback lemma (Ehrig *et al.* 2006, Theorem 4.26 (2)) and the uniqueness of pushout complements for rules in $\mathcal{M}$-adhesive transformation systems, and thus derive pushouts (3) and (5). The comatch m_2^* of the instantiated rule is induced by pushout (4). Finally, (6) is a pushout by pushout decomposition.

We now consider the NACs and a transformation diagram with step

$$G_2 \xRightarrow{p', m_2} H_2.$$

For a NAC-schema $n \in \mathbf{N}_S$, we have by Definition 2.11 for the satisfaction of NAC-schemata that a NAC occurrence

$$q' : N' \rightarrowtail G_2$$

of the instantiated rule p' defines a NAC occurrence of $n \in \mathbf{N}_S$, and, conversely, a violation of $n \in \mathbf{N}_S$ induces a NAC occurrence $q' : N' \rightarrowtail G_2$ of the instantiated rule p'. □

Fact 2.22 (sequential independence disregarding NACs for instantiated steps).

Proof. First, a mediating morphism $j' : L' \to D_1$ of the instantiated DPO diagrams directly induces a mediating morphism $j : L_2 \to D_1$ for the original DPO diagrams by

$$j = j' \circ e_2.$$

The case of $i' : R' \to D_2$ is dual.

Now, given a mediating morphism $j : L_2 \to D_1$, we will show that there is a mediating morphism $j' : L' \to D_1$ for the instantiated DPO diagram (the dual case with morphism $i : R_1 \to D_2$ is again analogous).

$$\begin{tikzcd}
K_1 \arrow[r, tail] \arrow[dd] & R_1 & & & L_2 \arrow[ddllll, dotted, bend right, "j"'] \arrow[dll, two heads, "j_e"'] \arrow[dl, two heads, "e_2"] \arrow[ddl, "f_2"] & K_2 \arrow[l, tail] \arrow[dd] \\
& & L'' \arrow[dll, tail, "j_1"] & L' \arrow[l, "iso"'] \arrow[d, tail, "m_2"'] & & \\
D_1 \arrow[rrr, tail, "g_1"] & & & G_1 & & D_2 \arrow[ll, tail] \arrow[l, phantom]
\end{tikzcd}$$

By Definition 2.14, we have the extremal $\mathcal{E}$-$\mathcal{M}$ factorisation

$$f_2 = m_2 \circ e_2.$$

We now construct the extremal $\mathcal{E}$-$\mathcal{M}$ factorisation

$$j = j_1 \circ j_e : L_2 \twoheadrightarrow L'' \rightarrowtail D_1.$$

By the uniqueness of extremal $\mathcal{E}$-$\mathcal{M}$ factorisations and commutativity

$$g_1 \circ j = f_2,$$

we have that $L'' \cong L'$ from *iso* and

$$m_2 = g_1 \circ j_1 \circ iso.$$

Therefore,

$$j_1 \circ iso : L' \to D_1$$

is compatible with m_2, that is,

$$m_2 = g_1 \circ j_1 \circ iso.$$

□

Fact 3.3 (intersection in $\mathbf{Sub}_{\mathcal{M}}(T)$).

Proof. Let $A, B \in |\mathbf{Sub}_{\mathcal{M}}(T)|$ and construct pullback (1) in $\mathbf{C}$ using $a \in \mathcal{M}$. This leads to $\mathcal{M}$-morphisms p_A and p_B, because $a, b \in \mathcal{M}$. Furthermore, p_A, p_B are morphisms in $\mathbf{Sub}_{\mathcal{M}}(T)$ by commutativity of the pullback. Now, a comparison object X for the product $A \cap B$ in $\mathbf{Sub}_{\mathcal{M}}(T)$ is also a comparison object for the pullback $A \cap B$ in $\mathbf{C}$. Thus, there is a unique morphism h satisfying the universal property. Furthermore, $h \in \mathcal{M}$ by decomposition of x_1, and h is a morphism in $\mathbf{Sub}_{\mathcal{M}}(T)$ by the commutativity of the diagram

$$\begin{tikzcd}
& X \arrow[dl, "x_1"'] \arrow[d, dotted, "h"] \arrow[dr, "x_2"] \arrow[ddr, bend left=60, "x"] & \\
A \arrow[dr, hook, "a"'] & A \cap B \arrow[l, "p_A"'] \arrow[r, "p_B"] \arrow[d, "i"] & B \arrow[dl, hook', "b"] \\
& T &
\end{tikzcd}$$

□

Fact 3.5 (distributivity).

Proof.

(i) The proof is analogous to the proof of Lack and Sobocinski (2005, Corollary 5.2) for adhesive categories, which we will just lift to $\mathcal{M}$-adhesive categories. Let $A, B, C \in |\mathbf{Sub}_{\mathcal{M}}(T)|$, then (1) in

$$\begin{tikzcd}
& & A \cap B \cap C \arrow[dll] \arrow[dr] & \\
A \cap B \arrow[dr] & & (1) & A \cap C \arrow[dll] \\
& (A \cap B) \cup (A \cap C) & &
\end{tikzcd}$$

$$\begin{tikzcd}
& & A \cap B \cap C \arrow[dll] \arrow[dr] \arrow[dd, dotted] & \\
A \cap B \arrow[dr] \arrow[dd] & & & A \cap C \arrow[dll] \arrow[dd] \\
& A \cap (B \cup C) \arrow[dd] & B \cap C \arrow[dll, dotted] \arrow[dr, dotted] & \\
B \arrow[dr] & & & C \arrow[dll] \\
& B \cup C & &
\end{tikzcd}$$

is a pushout in $\mathbf{C}$ by the general assumption that $\mathbf{C}$ has effective unions. The cube is commutative because all diagrams in $\mathbf{Sub}_{\mathcal{M}}(T)$ commute and

$$A \cap C \subseteq A \cap (B \cup C)$$

because

$$C \subseteq B \cup C.$$

The bottom face is a pushout in $\mathbf{C}$ along an $\mathcal{M}$-morphism because $\mathbf{C}$ has effective unions. The back faces are pullbacks in $\mathbf{C}$ according to Fact 3.3. The front left face of the cube is a pullback by pullback decomposition of the pullback $(2+3)$:

$$\begin{tikzcd}
A \cap B \arrow[r] \arrow[d] \arrow[dr, phantom, "(2)"] & A \cap (B \cup C) \arrow[r] \arrow[d] \arrow[dr, phantom, "(3)"] & A \arrow[d] \\
B \arrow[r] & B \cup C \arrow[r] & A \cup B \cup C
\end{tikzcd}$$

By analogous reasoning, the front right face of the cube is a pullback. By the VK-property of $\mathcal{M}$-adhesive categories, we get that the top face of the cube is a pushout, and by the uniqueness of pushouts, we deduce property (i).

(ii) Property (ii) now follows by duality in lattices. □

Fact 4.6 (linearisation).

Proof. Let

$$\forall\ i,j \in \{1,\dots,n\}, x \in \{rc, wc, d\} : s_i <_x s_j \Rightarrow i < j. \qquad (*)$$

$(\Rightarrow)$:

Let

$$s \overset{sw}{\approx}_{\mathcal{S}} seq(d)$$

and

$$seq(d) = \langle q_1,\dots,q_n\rangle\,.$$

We show that $(*)$ holds.

— We will first show the property for $s = seq(d)$, that is,

$$\forall\ i,j \in \{1,\dots,n\}, x \in \{rc, wc, d\} : q_i <_x q_j \Rightarrow i < j. \qquad (**)$$

Now

$$(**) \Leftrightarrow\ \forall\ i,j \in \{1,\dots,n\}, x \in \{rc, wc, d\} : i \geqslant j \Rightarrow q_i \not<_x q_j.$$

Let

$$\pi(q_i) = (\langle L_i, K_i, R_i\rangle\,, \mathbf{N}_i)$$
$$\pi(q_j) = (\langle L_j, K_j, R_j\rangle\,, \mathbf{N}_j).$$

For $i = j$, the condition is fulfilled directly because

$$\forall\ k \in \{1,\dots,n\} : L_k \cap R_k = K_k$$

according to Corradini *et al.* (2008, Proposition 30), where the proof can be lifted directly to the case of $\mathcal{M}$-adhesive categories using the results provided (constructions, intersection and union, as well as distributivity and the VK-property for the case where all morphisms are in $\mathcal{M}$).

For $i > j$, we consider cases:

– $x = rc$:
By definition, we have

$$q_i \not<_{rc} q_j \Leftrightarrow R_i \cap K_j \subseteq K_i.$$

We can build up the colimit of the instantiated transformation sequence d_I of d (see Definition 2.14) by stepwise pushouts. Let T_{i-1} be the colimit of the steps $d_1,\dots,d_{i-1}$. Then we have

$$K_j \subseteq T_{i-1}. \qquad (1)$$

Let T_i' be the colimit of transformation step d_i. So T_i' is given by the pushout (2) of $G_{i-1} \leftarrow D_i \rightarrow G_i$. We perform a pushout (3) of T_{i-1} and T_i' and obtain T_i. We now compose the pushouts (2) and (3) with the pushout (4) $D_i \leftarrow K_i \rightarrow R_i \rightarrow G_i$ of the transformation step d_i. This is also a pullback, so

$$R_i \cap T_{i-1} \cong K_i,$$

and using (1), this implies

$$R_i \cap K_j \subseteq K_i.$$

– $x = wc$:
By definition, we have

$$q_i \not<_{wc} q_j \Leftrightarrow R_i \cap L_j \subseteq K_i \cup K_j.$$

Using the above construction, we can also derive

$$L_j \subseteq T_{i-1},$$

so the equation holds.

– $x = d$:
By definition, we have

$$q_i \not<_{wc} q_j \Leftrightarrow K_i \cap L_j \subseteq K_j.$$

Using the above construction, we can also compose the pushout (5) $D_j \leftarrow K_j \rightarrow L_j \rightarrow G_{j-1}$ of the transformation 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), so the above equation holds.

— We will now show that the condition $(*)$ holds for every sequence s that is STS switch equivalent to $seq(d)$ disregarding 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, STS-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 Corradini *et al.* (2008, Theorem 32.2), so the condition is unaffected by any switching.

$(\Leftarrow)$:
We will show

$$\neg\left(s \overset{sw}{\approx}_{\mathcal{S}} seq(d)\right) \Rightarrow \neg(*).$$

by contraposition. Since s is a permutation of $seq(d)$, the condition

$$\neg\left(s \overset{sw}{\approx}_{\mathcal{S}} seq(d)\right)$$

means that s can be derived by switching neighbouring steps of $seq(d)$, where at least one switching is performed on a pair $(q_i; q_j)$ of steps that is dependent, that is,

$$\neg(q_i \diamond q_j),$$

which is equivalent to

$$(q_i <_x q_j)$$

for one or more $x \in \{rc, wc, d\}$ according to Corradini *et al.* (2008, Theorem 32.2) as above. Thus, this pair would violate the condition (∗) in the new order. Since s is assumed not to be STS-switch equivalent to $seq(d)$, there is at least one such pair where the final position of q_j is in front of q_i in s. □

Acknowledgements

We are grateful to Paolo Baldan and Barbara König for several fruitful discussions about the topics addressed in this paper, and to the anonymous referees for constructive criticisms that enabled us to improve it significantly.

References

Adámek, J., Herrlich, H and Strecker, G. (1990) *Abstract and Concrete Categories*, Wiley.

Baldan, P. (2000) *Modelling Concurrent Computations: from Contextual Petri Nets to Graph Grammars*, Ph.D. thesis, Computer Science Department, University of Pisa.

Baldan, P., Busi, N., Corradini, A and Pinna, G. M. (2004) Domain and event structure semantics for Petri nets with read and inhibitor arcs. *Theoretical Computer Science* **323** (1-3) 129–189.

Baldan, P., Corradini, A., Heindel, T., König, B and Sobociński, P. (2006) Processes for Adhesive Rewriting Systems. In: Aceto, L. and Ingólfsdóttir, A. (eds.) Proceedings FoSSaCS'06. *Springer-Verlag Lecture Notes in Computer Science* **3921** 202–216.

Braatz, B., Ehrig, H., Gabriel, K and Golas, U. (2010) Finitary $\mathcal{M}$-Adhesive Categories. In: Ehrig, H., Rensink, A., Rozenberg, G and Schürr, A. (eds.) Proceedings ICGT'10. *Springer-Verlag Lecture Notes in Computer Science* **6372** 234–249.

Brandt, C., Hermann, F and Engel, T. (2009) Modeling and Reconfiguration of critical Business Processes for the purpose of a Business Continuity Management respecting Security, Risk and Compliance requirements at Credit Suisse using Algebraic Graph Transformation. In: *Proceedings Dynamic and Declarative Business Processes (DDBP 2009)*, IEEE Xplore Digital Library 64–71.

Busi, N. and Pinna, G. M. (1999) Process semantics for Place/Transition nets with inhibitor and read arcs. *Fundamenta Informaticae* **40** (2-3) 165–197.

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

Corradini, A., Montanari, U and Rossi, F. (1996) Graph processes. *Fundamenta Informaticae* **26** (3/4) 241–265.

Ehrig, H., Ehrig, K., Prange, U and Taentzer, G. (2006) *Fundamentals of Algebraic Graph Transformation*, EATCS Monographs in Theoretical Computer Science, Springer-Verlag.

Ehrig, H., Golas, U and Hermann, F. (2010) Categorical Frameworks for Graph Transformation and HLR Systems based on the DPO Approach. *Bulletin of the EATCS* **102** 111–121.

Ehrig, H., Pfender, M and Schneider, H. (1973) Graph-grammars: an algebraic approach. In: Book, R. (ed.) *Switching and Automata Theory*, IEEE Computer Society Press 167–180.

Freyd, P. and Scedrov, A. (1990) *Categories, Allegories*, North-Holland.

Habel, A., Heckel, R and Taentzer, G. (1996) Graph Grammars with Negative Application Conditions. *Fundamenta Informaticae* **26** (3/4) 287–313.

Habel, A. and Pennemann, K.-H. (2009) Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science* **19** (2) 245–296.

Heindel, T. (2010) Hereditary Pushouts Reconsidered. In: Ehrig, H., Rensink, A., Rozenberg, G and Schürr, A. (eds.) Proceedings ICGT'10. *Springer-Verlag Lecture Notes in Computer Science* **6372** 250–265.

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

Hermann, F., Corradini, A., Ehrig, H and König, B. (2010) Efficient Analysis of Permutation Equivalence of Graph Derivations Based on Petri Nets. *Electronic Communications of the EASST* **29**.

ISO/IEC (2004) *ISO/IEC 15909-1:2004, Software and system engineering – High-level Petri nets – Part 1: Concepts, definitions and graphical notation*, ISO/IEC.

Janicki, R. and Koutny, M. (1995) Semantics of inhibitor nets. *Information and Computation* **123** (1) 1–16.

Kastenberg, H., Hermann, F and Modica, T. (2006) Towards Translating Graph Transformation Systems by Model Transformation. *Electronic Communications of the EASST* **4**.

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

Lack, S. and Sobociński, P. (2004) Adhesive Categories. In: Proceedings FOSSACS'04. *Springer-Verlag Lecture Notes in Computer Science* **2987** 273–288.

Lack, S. and Sobociński, P. (2005) Adhesive and quasiadhesive categories. *Theoretical Informatics and Applications* **39** (3) 511–545.

Lambers, L. (2009) *Certifying Rule-Based Models using Graph Transformation*, Ph.D. thesis, Technische Universität Berlin.

Lambers, L., Ehrig, H., Orejas, F and Prange, U. (2008) Parallelism and Concurrency in Adhesive High-Level Replacement Systems with Negative Application Conditions. In: Proceedings of the ACCAT workshop at ETAPS 2007. *Electronic Notes in Theoretical Computer Science* **203** (6) 43–66.

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

Reisig, W. (1985) *Petri Nets: An Introduction*, EATCS Monographs on Theoretical Computer Science **4**, Springer-Verlag.

Rozenberg, G. and Engelfriet, J. (1996) Elementary Net Systems. In: Reisig, W. and Rozenberg, G. (eds.) Lectures on Petri Nets I: Basic Models. *Springer-Verlag Lecture Notes in Computer Science* **1491** 12–121.