# Checking property preservation of refining transformations for model-driven development

Technische Universiteit Eindhoven
Department of Mathematics and Computer Science

# Checking Property Preservation of Refining Transformations For Model-Driven Development

Luc Engelen and Anton Wijs

12/08

ISSN 0926-4515


editors: prof.dr. P.M.E. De Bra
prof.dr.ir. J.J. van Wijk

Reports are available at:
http://library.tue.nl/catalog/TUEPublication.csp?Language=dut&Type=ComputerScienceReports&Sort=Author&level=1 and
http://library.tue.nl/catalog/TUEPublication.csp?Language=dut&Type=ComputerScienceReports&Sort=Year&Level=1

Computer Science Reports 12-08
Eindhoven, April 2012

# Checking Property Preservation of Refining Transformations for Model-Driven Development

Luc Engelen and Anton Wijs

Technische Universiteit Eindhoven, P.O. Box 513
5600 MB Eindhoven, The Netherlands
{L.J.P.Engelen,A.J.Wijs}@tue.nl

**Abstract.** In Model-Driven Software Development, a software product is created through iteratively refined modelling. It is crucial that this process preserves certain desirable properties of the initial model. However, checking this is increasingly difficult as the models are increasingly more refined. We propose an incremental model checking technique to determine the preservation of safety and liveness properties in models of concurrent systems with respect to changes applied on individual processes, formalised as transformations of Labelled Transition Systems. The preservation check involves checking bisimilarity between transformed and new behaviour, and never involves reexploring unchanged behaviour. We prove its correctness and demonstrate its applicability.

## 1 Introduction

Model-Driven Software Development (MDSD) [1] entails creating an initial design, i.e. model, and using this as the starting point for the development of the implementation. Ideally, MDSD enables deriving source code directly from the model; however, for that, the model has to be at a sufficiently low abstraction-level. In practice, systems are typically designed in a top-down manner, since at the start, many required details are either not yet known, or irrelevant at that moment. This means that initially, a model is created at a high level of abstraction, and over time, this model is incrementally refined. To make this process automatic, the refinement steps are often defined in terms of *model transformations* [1].

Model checking (MC) [6] can play a vital role here to ensure that the model after a transformation still satisfies a given property $\varphi$. However, employing traditional techniques would mean completely starting over the whole MC procedure after each transformation step. We propose an MC technique to determine if a transformation is guaranteed to preserve $\varphi$. If this is the case, then rechecking can always be avoided, regardless of the model structure. If this is not the case, then either rechecking should be done, or another transformation should be defined. This general nature of $\varphi$-preservation is in line with the idea of transformations as *reusable templates*. Fig. 1 sketches the refinement of model $M_1$, leading to model $M_n$ through a sequence of refinement steps. When a property holds for a model, which is indicated by a black checkmark, and this model is

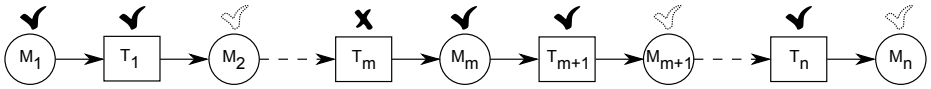



**Fig. 1:** Avoiding rechecking intermediate models by checking transformations

transformed by a transformation that preserves this property, which is also indicated by a black checkmark, then the property is guaranteed to hold for the resulting model. This is established without rechecking the property, which is indicated by a dashed checkmark. If a transformation is not guaranteed to preserve a property, e.g. $T_m$, the property needs to be rechecked for the resulting model.

In this report, models are represented by networks of Labelled Transition Systems (LTSs) [26], where each LTS describes the potential behaviour of a concurrent process, and synchronisation rules define how those LTSs can synchronise with each other. Refinement steps are formalised as sets of transformation rules applicable on process LTSs, defining how process behaviour should be refined. The results of this report are therefore relevant for any modelling language with a semantics expressible by networks of LTSs. For the proposed $\varphi$-preservation check, the only required information is the synchronisation rules, i.e. the description of the synchronisation mechanism, and the transformation rules. Furthermore, some reasonable assumptions are made concerning the applicability of the transformation rules on the process LTSs.

Recently [28], a fragment of the modal $\mu$-calculus [23] was identified, sufficiently expressive to capture the vast majority of practical safety, liveness, and fairness properties, which is compatible with *divergence-sensitive branching bisimilarity* [13], in the sense that two bisimilar systems preserve the same properties expressed in that fragment. Also, a technique called *maximal hiding* was developed, to automatically abstract away all behaviour not relevant for a property. We use these results to analyse the definitions of sets of LTS transformation rules, also in cases where the rules affect process behaviour synchronising with other processes. The check entails constructing networks of transformation rules based on their synchronisation dependencies, and checking whether the LTSs before and after transformation described by these networks are divergence-sensitive branching bisimilar after maximal hiding.

*Contributions.* A check for preservation of safety, liveness, and fairness properties of transformations of concurrent systems is formulated, and we prove it correct. Both its space and time complexity is proportional to the transformation rules, and hence shows speedups and reduction of memory use by many orders of magnitude compared to rechecking a property. To the best of our knowledge, we are the first to construct such a check.

*Paper structure.* Sect. 2 discusses related work and Sect. 3 introduces the preliminaries. In Sect. 4, we formalise LTS transformation. Next, in Sect. 5, we formulate and prove correct the main propositions regarding $\varphi$-preservation and

formulate an extension using divergencies in the model. Experimental results are given in Sect. 6, and Sect. 7 contains conclusions and pointers to future work.

## 2 Related Work

The first papers on IMC [36,37] propose techniques to reuse MC results of safety properties for a given LTS to determine whether it still satisfies the same property after some alterations. Large speedups are reported compared to complete rechecking, but the memory requirements are at least as high, since all states plus additional bookkeeping per state must reside in memory.

IMC algorithms can be seen as *dynamic graph algorithms* [11], and in this context, reachability is an *unbounded* problem [32], i.e. it cannot be determined solely based on the applied changes, which was also noted in [36]. Later work extends IMC to dynamic LTL model checking [8], and [21] reports complexity results for that problem. Papers on IMC for software models are [9,24,34].

Our work differs from the previously mentioned work as follows: firstly, we try to answer a slightly different, but no less relevant, IMC problem, which can be solved with a bounded technique: does a given number of transformation rules preserve a particular property (safety, liveness, or fairness) *in general*, given the synchronisation rules that the model adheres to, i.e. is there no possible network with such rules satisfying the property that can be altered in a way that the property no longer holds? Because of this, secondly, we strictly do not reexplore part of the LTS, but primarily focus on transformations, i.e. changes. Thirdly, we do not work with flat LTSs, but with networks of LTSs, and changes are applied to *these* LTSs, not the resulting network LTS. As such, we can take into account how the network LTS is composed. Due to this, and how we define the applicability of transformations, reachability is not an issue for us. Finally, we use bisimilarity checking in our IMC technique, as opposed to rechecking the property in changed parts of the LTS. In that sense, work on incremental bisimulation checking algorithms [35] is also related, but the context is very different, and our goal is not to maintain global bisimilarity results.

Other work, e.g. [2,4,5], uses *compositional model checking* [7] concepts, building a system using previously verified components. When a system does not consist of multiple components, the approach does not pay off, whereas ours does.

*Graph transformation* [19] is obviously related to LTS transformation, but most work involves graph transformation as a means to express potential behaviour of a system, whereas we use it to define refinement steps. In [20], semantics preservation of model transformations is studied for the setting where a model written in a language $X$ is transformed into a model written in a language $Y$. It nicely complements our work, since we consider transformations where the language remains the same, but the semantics is expected to change (insofar it is not relevant for the properties).

Monotonically adding functionality, as opposed to refining, is addressed in e.g. [3]. The focus is on updating property formulae as features are added; it

could be interesting to see if a similar approach is applicable in our setting to update properties. Work related to $B$, e.g. [27], is on strictly refining existing functionalities. We also support adding new functionality, as long as it is not relevant for the desired property.

Most related approaches use partial results of an earlier computation to perform a new one, thereby focussing on the specific old and new situation. In our approach, transformations are the primary subject, as it involves determining whether a transformation is $\varphi$-preserving in general. If this is the case, then any model can safely be transformed.

Finally, verification of model transformations has also been studied in the context of MDSD. Giese et al. relate input and output models when specifying a transformation, and use a theorem prover to show semantic equivalence between the input and output of the transformation [12]. A downside of this approach is that it is not completely automated and thus requires manual labor, whereas our approach is automated. Instance-based verification of model transformations is described in [22]. Their approach entails generating a certificate for each model that is transformed. These certificates are used to show that the model transformation preserves certain properties for the given input model, but cannot be used to show that properties are preserved for arbitrary input models.

## 3 Background

*Labelled transition system.* We express the semantics of a model in a finite-state LTS, which underlies action-based modelling languages. An LTS $\mathcal{G}$ is a tuple $\langle \mathcal{S}_\mathcal{G}, \mathcal{A}_\mathcal{G}, \mathcal{T}_\mathcal{G}, \mathcal{I}_\mathcal{G} \rangle$, where $\mathcal{S}_\mathcal{G}$ is the (finite) set of states, $\mathcal{A}_\mathcal{G}$ is the set of actions (including the invisible action $\tau$), $\mathcal{T}_\mathcal{G} \subseteq \mathcal{S}_\mathcal{G} \times \mathcal{A}_\mathcal{G} \times \mathcal{S}_\mathcal{G}$ is the transition relation, and $\mathcal{I}_\mathcal{G} \subseteq \mathcal{S}_\mathcal{G}$ the set of initial states.[1] Actions in $\mathcal{A}_\mathcal{G}$ are noted $a$, $b$, $c$, etc. A transition $\langle s_1, a, s_2 \rangle \in \mathcal{T}_\mathcal{G}$ (also noted $s_1 \xrightarrow{a}_\mathcal{G} s_2$) means that the system can move from state $s_1$ to state $s_2$ by performing action $a$. The reflexive transitive closure of $\xrightarrow{\tau}_\mathcal{G}$ is denoted by $\Longrightarrow_\mathcal{G}$.

*Network of* LTS*s.* A system consisting of a finite number of concurrent processes can be described by expressing how a number of LTSs interact with each other. For this, we use the *network of* LTS*s* model [26]. Given an integer $n > 0$, $1..n$ is the set of integers ranging from 1 to $n$. A vector $\overline{v}$ of size $n$ contains $n$ elements indexed by $1..n$. For $i \in 1..n$, $\overline{v}[i]$ denotes element $i$ in $\overline{v}$.

**Definition 1 (Network of LTSs).** *A* network of LTSs $\mathcal{M}$ *of size* $n$ *is a pair* $(\Pi, \mathcal{V})$ *where:*

- $\Pi$ *is a vector of* $n$ *(process)* LTS*s. For each* $i \in 1..n$*, we write* $\Pi[i] = (\mathcal{S}_i, \mathcal{A}_i, \mathcal{T}_i, \mathcal{I}_i)$*, and* $s_1 \xrightarrow{a}_i s_2$ *is shorthand for* $s_1 \xrightarrow{a}_{\Pi[i]} s_2$*;*

[1] Usually, LTSs representing potential behaviour of concurrent systems have only one initial state. Here, we support having multiple initial states to enable representing transformation rule patterns in terms of LTSs (see Section 4). In all other cases, one should keep in mind that LTSs are expected to have a single initial state.

– $\mathcal{V}$ *is a finite set of synchronisation rules. A synchronisation rule is a tuple* $(\bar{t}, a)$*, where* $a$ *is an action label and* $\bar{t}$ *is a vector of size* $n$ *called a* synchronisation vector*, whose elements are action labels from* $\bigcup_{i \in 1..n} \mathcal{A}_i$ *and a special symbol* $\bullet$ *which does not occur as a label in the* Lts*s.*

Combining the Ltss in $\Pi$ according to the rules in $\mathcal{V}$ in the network $\mathcal{M}$ produces a new Lts. A network $\mathcal{M}$ represents an Lts $(\mathcal{S}_\mathcal{M}, \mathcal{A}_\mathcal{M}, \mathcal{T}_\mathcal{M}, \mathcal{I}_\mathcal{M})$ where

– $\mathcal{I}_\mathcal{M} = \{\langle s_{I,1}, \ldots, s_{I,n}\rangle \mid \forall i \in 1..n. s_{I,i} \in \mathcal{I}_i\}$;
– $\mathcal{A}_\mathcal{M} = \{a \mid (\bar{t}, a) \in \mathcal{V}\}$;
– $\mathcal{S}_\mathcal{M} = \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$;
– $\mathcal{T}_\mathcal{M}$ is the smallest transition relation satisfying: $(\bar{t}, a) \in \mathcal{V} \wedge \forall i \in 1..n. (\bar{t}[i] = \bullet \wedge s'[i] = s[i]) \vee (\bar{t}[i] \neq \bullet \wedge s[i] \xrightarrow{\bar{t}[i]}_i s'[i]) \Rightarrow s \xrightarrow{a}_\mathcal{M} s'$.

The set of indices active for $\bar{t}$ is $Ac(\bar{t}) = \{i \mid i \in 1..n \wedge \bar{t}[i] \neq \bullet\}$, and the set of actions involved in $\bar{t}$ is $A(\bar{t}) = \bigcup_{i \in 1..n} \{\bar{t}[i]\} \setminus \{\bullet\}$. Let $\mathcal{A}_s = \{a \mid \exists (\bar{t}, b) \in \mathcal{V}_y. a \in A(\bar{t}) \wedge |Ac(\bar{t})| > 1\}$ be the set of actions involved in rules with multiple processes. With $\bar{s} \xrightarrow{a\ \bar{t}}_\mathcal{M} \bar{s}'$, we denote that $\bar{t}$ enables $\bar{s} \xrightarrow{a}_\mathcal{M} \bar{s}'$.

As explained in [10], the network of Ltss model is very basic, yet very expressive. Through synchronisation rules, it is possible to model not only independent process behaviour and traditional (two-party) *synchronising behaviour*, but also non-deterministic and multi-way synchronising behaviour. For example, consider a network with $n = 3$ and actions $a_i \in \mathcal{A}_i$, for $i \in 1..3$. Then, a rule $(\langle \bullet, a_2, \bullet\rangle, a_2)$ expresses that $a_2$ can be fired independently in $\Pi[2]$, and a rule $(\langle \bullet, a_2, a_3\rangle, a_4)$ expresses that actions $a_2, a_3$, if both can be fired simultaneously in $\Pi[2]$ and $\Pi[3]$, respectively, produce a single transition labelled $a_4$, which therefore models synchronisation of $\Pi[2]$ and $\Pi[3]$. But we can also define a rule $(\langle a_1, a_2, a_3\rangle, a_4)$ expressing multi-way synchronisation of all the process Ltss, and if we also have $a_2 \in \mathcal{A}_1$ and add a rule $(\langle a_2, \bullet, a_3\rangle, a_4)$, then we may have states in the network Lts where $\Pi[3]$ can non-deterministically synchronise with either $\Pi[1]$ or $\Pi[2]$, if both of the latter Ltss can fire $a_2$.

On the left of Fig. 2, a network comprising two process Ltss is shown. The corresponding network Lts is shown on the right. The synchronisation rules specify that actions $b$ and $d$ of the processes synchronise and lead to an action $e$ in the network Lts, that action $f$ can only be fired if it is enabled in both process Ltss, and that action $a$ of the leftmost process and action $c$ of the rightmost process lead to actions $a$ and $c$ in the network Lts.

*Hiding.* For networks of Ltss, we define the hiding operator $\tau_H$ with $H \subseteq \mathcal{A}_\mathcal{M}$ as follows:

**Definition 2 (Hiding).** *Let* $\mathcal{A}$ *be a set of actions and* $H \subseteq \mathcal{A}$*. The* hiding *of an action* $a \in \mathcal{A}$ *w.r.t.* $H$ *is defined as follows:*

$$\tau_H(a) = \begin{cases} a \text{ if } a \notin H \\ \tau \text{ if } a \in H \end{cases}$$

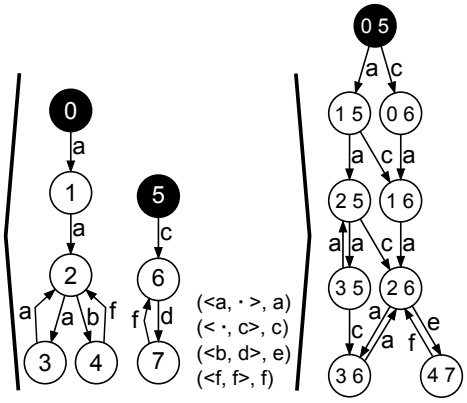



**Fig. 2:** A network and its corresponding LTS. Initial states are colored black

*The hiding of a network $\mathcal{M} = (\Pi, \mathcal{V})$ w.r.t. $H$ is defined as follows:*

$$\tau_H(\langle \Pi, \mathcal{V} \rangle) = \left\langle \Pi, \{(\bar{t}, \tau_H(a)) \mid (\bar{t}, a) \in \mathcal{V}\} \right\rangle .$$

*Divergence-sensitive branching bisimilarity.* As equivalence relation between LTSs, we consider divergence-sensitive branching bisimilarity [13,14], which preserves branching-time properties such as inevitable reachability and $\tau$-cycles.

**Definition 3 (Divergence-Sensitive Branching Bisimulation).** *A binary relation $B$ on $\mathcal{S}_\mathcal{G}$ is a divergence-sensitive branching bisimulation if $B$ is symmetric and $s\ B\ t$ implies that*

- *if $s \xrightarrow{a}_\mathcal{G} s'$ then*
  - *either $a = \tau$ with $s'\ B\ t$;*
  - *or $t \Longrightarrow_\mathcal{G} \hat{t} \xrightarrow{a}_\mathcal{G} t'$ with $s\ B\ \hat{t}$ and $s'\ B\ t'$.*
- *if for all $k \geq 0$ and $s = s_0$, $s_k \xrightarrow{\tau}_\mathcal{G} s_{k+1}$ then for all $\ell \geq 0$ and $t = t_0$, $t_\ell \xrightarrow{\tau}_\mathcal{G} t_{\ell+1}$ and $s_k\ B\ t_\ell$ for all $k, \ell$.*

*Two states $s$ and $t$ are* divergence-sensitive branching bisimilar*, noted $s \leftrightarroweq_b^\Delta t$, if there is a divergence-sensitive branching bisimulation $B$ with $s\ B\ t$.*

We say that two sets of states $S, S' \subseteq \mathcal{S}$ are divergence-sensitive branching bisimilar, i.e. $S \leftrightarroweq_b^\Delta S'$, iff $\forall s \in S. \exists s' \in S'. s \leftrightarroweq_b^\Delta s'$ and vice versa. Two LTSs $\mathcal{G}_1, \mathcal{G}_2$ are divergence-sensitive branching bisimilar, i.e. $\mathcal{G}_1 \leftrightarroweq_b^\Delta \mathcal{G}_2$, iff $\mathcal{I}_1 \leftrightarroweq_b^\Delta \mathcal{I}_2$.

A state $s$ is *diverging*, noted $s \uparrow$, iff an infinite $\tau$-path is reachable from $s$, which for finite LTSs means that a $\tau$-cycle is reachable via $\tau$-transitions.

Divergence-sensitive branching bisimilarity is a congruence for networks of LTSs if they are *admissible* w.r.t. $\tau$-transitions:

**Definition 4 (Network Admissibility).** *A network $\mathcal{M} = (\Pi, \mathcal{V})$ is called* admissible *iff the following holds:*

1. ***No synchronisation and renaming:*** $\forall(\bar{t}, a) \in \mathcal{V}.\bar{t}[i] = \tau \Rightarrow Ac(\bar{t}) = \{i\} \wedge a = \tau$;
2. ***No cut:*** $\exists s_1, s_2 \in \mathcal{S}_i.s_1 \xrightarrow{\tau}_i s_2 \Rightarrow \exists(\bar{t}, \tau) \in \mathcal{V}.\bar{t}[i] = \tau$.

In the following, we only consider admissible networks.

*The modal $\mu$-calculus $L_\mu^{dsbr}$.* In [28], a fragment of the modal $\mu$-calculus, called $L_\mu^{dsbr}$, was identified which is fully compatible with the so-called *maximal hiding* technique [28] and divergence-sensitive branching bisimilarity. Formulas in $L_\mu^{dsbr}$ consist of action formulas (noted $\alpha$) and state formulas (noted $\varphi$ and $\psi$). The syntax and semantics of these formulas are defined in Fig. 3.

Action formulas are built over the set of actions by using Boolean connectors as in Action-based CTL [29].

Interpretation $[\![\alpha]\!]_{\mathcal{A}_\mathcal{G}}$ of $\alpha$ on the set of actions $\mathcal{A}_\mathcal{G}$ denotes the set of actions satisfying $\alpha$. An action $a$ satisfies a formula $\alpha$ (noted $a \models_{\mathcal{A}_\mathcal{G}} \alpha$) iff $a \in [\![\alpha]\!]_{\mathcal{A}_\mathcal{G}}$. A transition $s_1 \xrightarrow{a}_\mathcal{G} s_2$ with $a \models_{\mathcal{A}_\mathcal{G}} \alpha$ is called an $\alpha$-transition.

State formulas are built from Boolean connectors, the minimal fixed point operator ($\mu$) defined over propositional variables $X$ belonging to a set $\mathcal{X}$, and some new operators:

1. The weak possibility modality $\langle(\varphi_1?.\alpha_1)^*\rangle\,\psi$, where $\alpha_1$ must capture $\tau$, characterises the states having an outgoing sequence of (0 or more) $\alpha_1$-transitions, whose intermediate states satisfy $\varphi_1$ and whose terminal state satisfies $\psi$, and the weak infinite looping operator $\langle\varphi_1?.\alpha_1\rangle$ @ characterises the states having an infinite outgoing sequence of $\alpha_1$-transitions whose intermediate states satisfy $\varphi_1$. The saturation operator is $[\varphi_1?.\alpha_1]\ \dashv = \neg\langle\varphi_1?.\alpha_1\rangle$ @. When $\varphi_1$ of a weak operator is $\mathsf{true}$, it can be omitted.
2. Strong modalities $\langle\alpha_2\rangle\,\varphi$ and $[\alpha_2]\,\varphi = \neg\langle\alpha_2\rangle\neg\varphi$, where $\alpha_2$ denotes only visible actions, are restricted syntactically such that they can appear only after a weak possibility or necessity modality. The intuition is that visible transitions matched by a strong modality will remain in the LTS after maximal hiding and $\leftrightarrows_b^\Delta$ minimisation, and the transition sequences preceding them can become invisible or even disappear in the minimised LTS without affecting the interpretation of the formula, because these sequences are still captured by the preceding weak modality.

A propositional context $\rho : \mathcal{X} \to 2^{\mathcal{S}_\mathcal{G}}$ is a partial function mapping propositional variables to subsets of states. The notation $\rho \oslash [U/X]$ stands for a propositional context identical to $\rho$ except for variable $X$, which is mapped to the state subset $U$. The interpretation $[\![\varphi]\!]_\mathcal{G}\,\rho$ of a state formula on an LTS $\mathcal{G}$ and a propositional context $\rho$ (which assigns a set of states to each propositional variable occurring free in $\varphi$) denotes the subset of states satisfying $\varphi$ in that context. The Boolean connectors are interpreted as usual in terms of set operations. The minimal fixed point operator $\mu X.\varphi_1$ (resp. the maximal fixed point operator $\nu X.\varphi_1$) denotes the least (resp. greatest) solution of the equation $X = \varphi_1$ interpreted over the complete lattice $\langle 2^{\mathcal{S}_\mathcal{G}}, \emptyset, \mathcal{S}_\mathcal{G}, \cap, \cup, \subseteq\rangle$.

**State formulas:**

$$\varphi ::= \mathsf{false} \mid \neg\varphi_1 \mid \varphi_1 \vee \varphi_2 \mid \langle(\varphi_1?.\alpha_1)^*\rangle\,\psi \mid \langle\varphi_1?.\alpha_1\rangle\,@ \mid X \mid \mu X.\varphi_1$$
$$\psi ::= \varphi \mid \langle\alpha_2\rangle\,\varphi \mid \neg\psi_1 \mid \psi_1 \vee \psi_2$$

where $\tau \in [\![\alpha_1]\!]_{\mathcal{A}_\mathcal{G}}$ and $\tau \notin [\![\alpha_2]\!]_{\mathcal{A}_\mathcal{G}}$

$$\begin{aligned}
[\![\mathsf{false}]\!]_\mathcal{G}\,\rho &= \emptyset\\
[\![\neg\varphi_1]\!]_\mathcal{G}\,\rho &= \mathcal{S}_\mathcal{G} \setminus [\![\varphi_1]\!]_\mathcal{G}\,\rho\\
[\![\varphi_1 \vee \varphi_2]\!]_\mathcal{G}\,\rho &= [\![\varphi_1]\!]_\mathcal{G}\,\rho \cup [\![\varphi_2]\!]_\mathcal{G}\,\rho\\
[\![\langle(\varphi_1?.\alpha_1)^*\rangle\,\psi]\!]_\mathcal{G}\,\rho &= \{s \in \mathcal{S}_\mathcal{G} \mid \exists m \geq 0.s = s_0 \wedge (\forall 0 \leq i < m.s_i \xrightarrow{a_{i+1}} s_{i+1} \in \mathcal{T}_\mathcal{G}\\
&\qquad \wedge\ a_{i+1} \in [\![\alpha_1]\!]_{\mathcal{A}_\mathcal{G}} \wedge\ s_i \in [\![\varphi_1]\!]_\mathcal{G}\,\rho) \wedge\ s_m \in [\![\psi]\!]_\mathcal{G}\,\rho\}\\
[\![\langle\varphi_1?.\alpha_1\rangle\,@]\!]_\mathcal{G}\,\rho &= \{s \in \mathcal{S}_\mathcal{G} \mid s = s_0 \wedge \forall i \geq 0.(s_i \xrightarrow{a_{i+1}} s_{i+1} \in \mathcal{T}_\mathcal{G} \wedge a_{i+1} \in [\![\alpha]\!]_{\mathcal{A}_\mathcal{G}}\\
&\qquad \wedge\ s_i \in [\![\varphi_1]\!]_\mathcal{G}\,\rho)\}\\
[\![X]\!]_\mathcal{G}\,\rho &= \rho(X)\\
[\![\mu X.\varphi_1]\!]_\mathcal{G}\,\rho &= \bigcap\{U \subseteq \mathcal{S}_\mathcal{G} \mid [\![\varphi_1]\!]_\mathcal{G}\,(\rho \oslash [U/X]) \subseteq U\}\\
[\![\langle\alpha_2\rangle\,\varphi]\!]_\mathcal{G}\,\rho &= \{s \in \mathcal{S}_\mathcal{G} \mid \exists s \xrightarrow{a} s' \in \mathcal{T}_\mathcal{G}.a \in [\![\alpha_2]\!]_{\mathcal{A}_\mathcal{G}} \wedge s' \in [\![\varphi]\!]_\mathcal{G}\,\rho\}
\end{aligned}$$

**Action formulas:**

$$\alpha ::= a \mid \mathsf{false} \mid \neg\alpha_1 \mid \alpha_1 \vee \alpha_2$$

$$\begin{aligned}
[\![a]\!]_{\mathcal{A}_\mathcal{G}} &= \{a\}\\
[\![\mathsf{false}]\!]_{\mathcal{A}_\mathcal{G}} &= \emptyset\\
[\![\neg\alpha_1]\!]_{\mathcal{A}_\mathcal{G}} &= \mathcal{A}_\mathcal{G} \setminus [\![\alpha_1]\!]_{\mathcal{A}_\mathcal{G}}\\
[\![\alpha_1 \vee \alpha_2]\!]_{\mathcal{A}_\mathcal{G}} &= [\![\alpha_1]\!]_{\mathcal{A}_\mathcal{G}} \cup [\![\alpha_2]\!]_{\mathcal{A}_\mathcal{G}}
\end{aligned}$$

**Fig. 3:** Syntax and semantics of $L_\mu^{dsbr}$

The fact that a state $s$ satisfies a closed formula $\varphi$, i.e. a formula without free propositional variables, is denoted by $s \models_\mathcal{G} \varphi$. Finally, $\models_\mathcal{G} \varphi$ denotes $\forall s_I \in \mathcal{I}_\mathcal{G}.s_I \models_\mathcal{G} \varphi$.

In $L_\mu^{dsbr}$, it is possible to express safety, liveness, and fairness properties. For example, from the divergence-sensitive branching bisimilarity point of view, deadlock states are states leading eventually via $\tau$-transitions to states without successors:

$$\mathit{deadlock} = [\mathsf{true}^*]\,[\neg\tau]\,\mathsf{false} \wedge [\tau]\,\dashv$$

*Maximal hiding.* When checking a state formula $\varphi$ on an LTS, some actions can be hidden (renamed to $\tau$) without disturbing the interpretation of $\varphi$.

**Definition 5 (Hiding Set).** *Let $\alpha$ be an action formula interpreted over a set of actions $\mathcal{A}_\mathcal{G}$. The* hiding set *of $\alpha$ w.r.t. $\mathcal{A}_\mathcal{G}$ is defined as follows:*

$$h_{\mathcal{A}_\mathcal{G}}(\alpha) = \begin{cases} [\![\alpha]\!]_{\mathcal{A}_\mathcal{G}} & \text{if } \tau \models \alpha \\ \mathcal{A} \setminus [\![\alpha]\!]_{\mathcal{A}_\mathcal{G}} & \text{if } \tau \not\models \alpha \end{cases}$$

*The hiding set of a state formula $\varphi$ w.r.t. $\mathcal{A}_\mathcal{G}$, noted $h_{\mathcal{A}_\mathcal{G}}(\varphi)$, is defined as the intersection of $h_{\mathcal{A}_\mathcal{G}}(\alpha)$ for all action subformulas $\alpha$ of $\varphi$.*

We denote maximal hiding in an LTS $\mathcal{G}$ as $\tilde{\tau}_\varphi(\mathcal{G}) = \tau_{h_{\mathcal{A}_\mathcal{G}}(\varphi)}(\mathcal{G})$. In [28], it is shown that maximal hiding preserves $L_\mu^{dsbr}$ properties: $\models_\mathcal{G} \varphi \iff \models_{\tilde{\tau}_\varphi(\mathcal{G})} \varphi$. As an example, consider the $L_\mu^{dsbr}$ formula below, expressing the inevitable reachability of a *recv* action after every *send* action:

$$\varphi = [\mathsf{true}^*]\,[\mathit{send}]\,([(\neg \mathit{recv})^*]\,\neg \mathit{deadlock} \wedge [\neg \mathit{recv}]\,\dashv)$$

The subformula stating the inevitable reachability of a *recv* action is the conjunction of a weak necessity modality forbidding the occurrence of deadlocks before a *recv* action has been reached, and a weak saturation operator forbidding the presence of cycles not passing through a *recv* action. When checking $\varphi$ on an LTS, one can hide $h_{\mathcal{A}_\mathcal{G}}(\varphi) = h_{\mathcal{A}_\mathcal{G}}(\mathit{send}) \cap h_{\mathcal{A}_\mathcal{G}}(\neg \mathit{recv}) = (\mathcal{A}_\mathcal{G} \setminus [\![\mathit{send}]\!]_{\mathcal{A}_\mathcal{G}}) \cap [\![\neg \mathit{recv}]\!]_{\mathcal{A}_\mathcal{G}} = (\mathcal{A}_\mathcal{G} \setminus \{\mathit{send}\}) \cap (\mathcal{A}_\mathcal{G} \setminus \{\mathit{recv}\}) = \mathcal{A}_\mathcal{G} \setminus \{\mathit{send}, \mathit{recv}\}$, i.e., all actions other than *send* and *recv*, without changing the interpretation of the formula.

Now we can state the main result about $L_\mu^{dsbr}$, namely that for closed $\varphi$, this fragment is compatible with the $\leftrightarrows_b^\Delta$ relation.

**Proposition 1 (Compatibility with $\leftrightarrows_b^\Delta$).** *Let $\mathcal{G} = \langle \mathcal{S}_\mathcal{G}, \mathcal{A}_\mathcal{G}, \mathcal{T}_\mathcal{G}, \mathcal{I}_\mathcal{G} \rangle$ be an* LTS *and let $s_1, s_2 \in \mathcal{S}_\mathcal{G}$ such that $s_1 \leftrightarrows_b^\Delta s_2$. Then:*

$$s_1 \models_\mathcal{G} \varphi \Leftrightarrow s_2 \models_\mathcal{G} \varphi$$

*for any closed state formula $\varphi$ of $L_\mu^{dsbr}$.*

*Proof.* Given in [28]. □

This allows reducing an LTS (after maximal hiding) modulo divergence-sensitive branching bisimilarity before verifying a closed $L_\mu^{dsbr}$ formula. This is done in [28]. It also allows reasoning about LTSs w.r.t. properties: given a $L_\mu^{dsbr}$ formula $\varphi$ and LTSs $\mathcal{G}_1, \mathcal{G}_2$ with $\models_{\mathcal{G}_1} \varphi$, then if $\tilde{\tau}_\varphi(\mathcal{G}_1) \leftrightarrows_b^\Delta \tilde{\tau}_\varphi(\mathcal{G}_2)$, also $\models_{\mathcal{G}_2} \varphi$.

## 4 LTS Transformations

In this report, models are represented by LTSs and refinement steps are formalised as LTS transformations. These transformations are defined by means of *transformation rules*:

**Definition 6 (LTS transformation rule).** *An* LTS transformation rule $r : \mathcal{L}^r \mapsto \mathcal{R}^r$ *consists of a* left pattern LTS $\mathcal{L}^r = (\mathcal{S}_{\mathcal{L}^r}, \mathcal{A}_{\mathcal{L}^r}, \mathcal{T}_{\mathcal{L}^r}, \mathcal{I}_{\mathcal{L}^r})$ *and a* right pattern LTS $\mathcal{R}^r = (\mathcal{S}_{\mathcal{R}^r}, \mathcal{A}_{\mathcal{R}^r}, \mathcal{T}_{\mathcal{R}^r}, \mathcal{I}_{\mathcal{R}^r})$, *with* $\mathcal{I}_{\mathcal{L}^r} = \mathcal{I}_{\mathcal{R}^r} = (\mathcal{S}_{\mathcal{L}^r} \cap \mathcal{S}_{\mathcal{R}^r})$.

States $\mathcal{S}_{\mathcal{L}^r} \cap \mathcal{S}_{\mathcal{R}^r}$ (the *glue-states*) are all initial and define how $\mathcal{R}^r$ should replace $\mathcal{L}^r$, as all changes are applied relative to them. In graph transformation terminology, LTS $\mathcal{L}^r \cap \mathcal{R}^r$ would be called the *interface*.[2] We call a rule *applicable* on an LTS $\mathcal{G}$ iff there exists at least one *match* $m_r : \mathcal{L}^r \mapsto \mathcal{G}$ for which the following holds:

**Definition 7 (Transformation rule match).** *Let $\mathcal{G} = (\mathcal{S}_\mathcal{G}, \mathcal{A}_\mathcal{G}, \mathcal{T}_\mathcal{G}, \mathcal{I}_\mathcal{G})$ be an* LTS*. A transformation rule $r : \mathcal{L}^r \mapsto \mathcal{R}^r$ has a* match *$m_r : \mathcal{L}^r \mapsto \mathcal{G}$ on $\mathcal{G}$ iff $m_r$ is injective and:*

1. $\forall s_1 \xrightarrow{a}_{\mathcal{L}^r} s_2 . m_r(s_1) \xrightarrow{a}_\mathcal{G} m_r(s_2)$;[3]
2. $\forall s_1 \in \mathcal{S}_{\mathcal{L}^r} \setminus \mathcal{S}_{\mathcal{R}^r}, s_2 \in \mathcal{S}_\mathcal{G}$ :
   - $m_r(s_1) \xrightarrow{a}_\mathcal{G} s_2 \Rightarrow \exists s \in \mathcal{S}_{\mathcal{L}^r} . s_1 \xrightarrow{a} s \wedge m_r(s) = s_2$;
   - $s_2 \xrightarrow{a}_\mathcal{G} m_r(s_1) \Rightarrow \exists s \in \mathcal{S}_{\mathcal{L}^r} . s \xrightarrow{a} s_1 \wedge m_r(s) = s_2$.

We will shortly comment on the latter part of this definition, but first, we define the effect of a match. If a rule $r$ is applicable to $\mathcal{G}$, then the latter can be *transformed* as follows:

**Definition 8 (LTS transformation).** *The* transformation *of an* LTS $\mathcal{G} = (\mathcal{S}_\mathcal{G}, \mathcal{A}_\mathcal{G}, \mathcal{T}_\mathcal{G}, \mathcal{I}_\mathcal{G})$ *according to a rule $r : \mathcal{L}^r \mapsto \mathcal{R}^r$ and a given match $m_r : \mathcal{L}^r \mapsto \mathcal{G}$ is defined as follows: $T_r^m(\mathcal{G}) = (\mathcal{S}_r^m, \mathcal{A}_r^m, \mathcal{T}_r^m, \mathcal{I}_\mathcal{G})$ where*

- $\mathcal{S}_r^m = (\mathcal{S} \setminus \{m_r(s) \mid s \in (\mathcal{S}_{\mathcal{L}^r} \setminus \mathcal{S}_{\mathcal{R}^r})\}) \cup (\mathcal{S}_{\mathcal{R}^r} \setminus \mathcal{S}_{\mathcal{L}^r})$;
- $\mathcal{T}_r^m = (\mathcal{T}_\mathcal{G} \cup \{s_1 \xrightarrow{a}_{\mathcal{R}^r} s_2 \mid s_1, s_2 \in (\mathcal{S}_{\mathcal{R}^r} \setminus \mathcal{S}_{\mathcal{L}^r})\} \cup$
  $\{m_r(s) \xrightarrow{a}_\mathcal{G} s_2 \mid s \in (\mathcal{S}_{\mathcal{L}^r} \cap \mathcal{S}_{\mathcal{R}^r}) \wedge s_2 \in (\mathcal{S}_{\mathcal{R}^r} \setminus \mathcal{S}_{\mathcal{L}^r}) \wedge s \xrightarrow{a}_{\mathcal{R}^r} s_2\} \cup$
  $\{s_1 \xrightarrow{a}_\mathcal{G} m_r(s) \mid s \in (\mathcal{S}_{\mathcal{L}^r} \cap \mathcal{S}_{\mathcal{R}^r}) \wedge s_1 \in (\mathcal{S}_{\mathcal{R}^r} \setminus \mathcal{S}_{\mathcal{L}^r}) \wedge s_1 \xrightarrow{a}_{\mathcal{R}^r} s\})$
  $\setminus \{s_1 \xrightarrow{a}_\mathcal{G} s_2 \mid \exists s, s' \in \mathcal{S}_{\mathcal{L}^r} . m_r(s) = s_1 \wedge m_r(s') = s_2 \wedge \{s, s'\} \cap (\mathcal{S}_{\mathcal{L}^r} \setminus \mathcal{S}_{\mathcal{R}^r}) \neq \emptyset\}$;
- $\mathcal{A}_r^m = \{a \mid \exists s_1 \xrightarrow{a}_{T_r^m(\mathcal{G})} s_2\} \cup \{\tau\}$.

LTS $T_r^m(\mathcal{G})$ is also called a *direct derivation* from $\mathcal{G}$ via $r$. The new set of states $\mathcal{S}_r^m$ consists of $\mathcal{S}_\mathcal{G}$ without the states matched to by $m_r$ that are represented in $\mathcal{S}_{\mathcal{L}^r} \setminus \mathcal{S}_{\mathcal{R}^r}$ (the states that are to be removed), and with the states in $\mathcal{S}_{\mathcal{R}^r} \setminus \mathcal{S}_{\mathcal{L}^r}$ (the newly added states). We assume that the latter states are 'fresh' in $\mathcal{S}_r^m$. Transitions $\mathcal{T}_r^m$ consist of $\mathcal{T}_\mathcal{G}$ together with the transitions between newly added states, and between new states and glue-states, but without the transitions between a removed state and either another removed state or a glue-state (the LTS on the right in Fig. 4 is an example of a transformation).

Returning to Def. 7, condition 2 expresses the so-called *dangling condition*. In graph transformation, an edge is called dangling if exactly one of its connecting vertices is removed by rule application (e.g., see [20,31,18]). It can be shown that a direct derivation exists iff $m_r$ satisfies the dangling condition [18]. In our setting, a rule not satisfying the dangling condition may result in reachable states becoming unreachable through transformation, and this effect cannot be

[2] We define $\mathcal{G}_1 \cap \mathcal{G}_2 = (\mathcal{S}_1 \cap \mathcal{S}_2, \mathcal{A}_1 \cap \mathcal{A}_2, \mathcal{T}_1 \cap \mathcal{T}_2, \mathcal{I}_1 \cap \mathcal{I}_2)$.

[3] Here, a state $s$ is interpreted as an LTS $\langle \{s\}, \emptyset, \emptyset, \{s\} \rangle$.

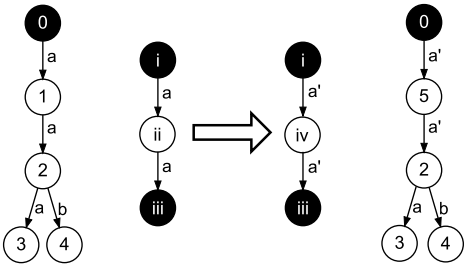



**Fig. 4:** Transformation rule matching

determined by just looking at the rule. In Fig. 4, the two smaller LTSs in the middle connected by an arrow denote a transformation rule. Here, it defines that any state matched on (ii) should be removed and replaced by a new state ((iv) in the rule). In the LTS on the left, the left pattern can be matched on $\{0, 1, 2\}$, but not on $\{1, 2, 3\}$, otherwise applying the rule would make 4 unreachable. The reason for making all glue-states initial, such as state (iii) in the transformation rule in Fig. 4, is to ensure that all glue-states are always involved in bisimilarity checks of rule patterns. This, however, only really starts being necessary when considering transformations of multiple parties in synchronising behaviour (see Section 5.3).

We refer with $m_r(\mathcal{S}_{\mathcal{L}^r}) \subseteq \mathcal{S}_{\mathcal{G}}$ to the set of states to which $\mathcal{S}_{\mathcal{L}^r}$ is exactly matched (they have the same number of elements), and with $m_r^{-1}$ to the inverse, i.e. $m_r^{-1}(m_r(\mathcal{S}_{\mathcal{L}^r})) = \mathcal{S}_{\mathcal{L}^r}$. Finally, $\hat{m}_r(\mathcal{S}_{\mathcal{L}^r})$ refers to the states which relate to non-glue-states in $\mathcal{L}^r$: $\hat{m}_r(\mathcal{S}_{\mathcal{L}^r}) = \{s \in m_r(\mathcal{S}_{\mathcal{L}^r}) \mid m_r^{-1}(s) \notin \mathcal{S}_{\mathcal{R}^r}\}$.

Now, we can express the following lemma about transitions between states:

**Lemma 1.** $\forall s \in m_r(\mathcal{S}_{\mathcal{L}^r}), s' \in \mathcal{S}_{\mathcal{G}} \setminus m_r(\mathcal{S}_{\mathcal{L}^r}).s \xrightarrow{a}_{\mathcal{G}} s' \vee s' \xrightarrow{a}_{\mathcal{G}} s \Rightarrow s \in m_r(\mathcal{S}_{\mathcal{L}^r}) \setminus \hat{m}_r(\mathcal{S}_{\mathcal{L}^r})$

*Proof.* Follows directly from Def. 7 and the fact that $m_r$ is an injection. According to Def. 7, for all states $s \in \hat{m}_r(\mathcal{S}_{\mathcal{L}^r}), s' \in \mathcal{S}_{\mathcal{G}}$, if either $s \xrightarrow{a}_{\mathcal{G}} s'$ or $s' \xrightarrow{a}_{\mathcal{G}} s$, then $s' \in m_r(\mathcal{S}_{\mathcal{L}^r})$. Since here, $s' \in \mathcal{S}_{\mathcal{G}} \setminus m_r(\mathcal{S}_{\mathcal{L}^r})$, we must have that $s \notin \hat{m}_r(\mathcal{S}_{\mathcal{L}^r})$, therefore $s \in m_r(\mathcal{S}_{\mathcal{L}^r}) \setminus \hat{m}_r(\mathcal{S}_{\mathcal{L}^r})$. □

Lemma 1 directly results from the way we treat dangling edges: if a transition connects two states, one of which is matched on by $r$, and one which is not matched on by $r$, then the state matched on must be a glue-state; if this was not the case, then the state would not exist anymore after transformation, but then the transition would be a dangling edge, contradicting the existence of the match.

*Rule systems.* With transformation rules, a *rule system* $\Sigma = (R, \hat{\mathcal{V}})$ can be built, with $R$ a set of transformation rules and $\hat{\mathcal{V}}$ a set of synchronisation rules. In this report, a rule system is a formal description how given potential behaviour (represented by an LTS) evolves into other (often more detailed) potential behaviour.

In that context, we are not interested in all possible interleavings of applying the transformation rules in a system, and the number of applications should be finite. For this reason, we assume that rule systems are *terminating* and *confluent*. Let $\mathcal{G} \Rrightarrow_R \mathcal{G}'$ denote the fact that an LTS $\mathcal{G}'$ can be obtained by applying a rule $r \in R$ on one match in LTS $\mathcal{G}$, and let $\Rrightarrow_R^*$ be the reflexive, transitive closure of $\Rrightarrow_R$. Then, $\Sigma$ is *terminating* iff $\Rrightarrow_R$ is terminating, and $\Sigma$ is confluent iff the following holds.

**Definition 9 (Confluent rule system).** *Let $\Sigma = (R, \hat{\mathcal{V}})$ be a rule system and $\mathcal{G}$ be an* LTS*. $\Sigma$ is called* confluent *iff for all* LTS*s $\mathcal{G}_1$, $\mathcal{G}_2$ with $\mathcal{G} \Rrightarrow_R^* \mathcal{G}_1$, $\mathcal{G} \Rrightarrow_R^* \mathcal{G}_2$, there exists an* LTS *$\mathcal{G}_3$ such that $\mathcal{G}_1 \Rrightarrow_R^* \mathcal{G}_3$ and $\mathcal{G}_2 \Rrightarrow_R^* \mathcal{G}_3$*

From graph theory, it is known that for general rule systems, confluence is undecidable, but it is decidable under certain conditions [25,31]. Here, we ensure that $\Sigma$ is terminating and confluent by requiring that (1) for all $r \in R$, action labels in the right pattern are 'fresh', i.e. that $\mathcal{A}_{\mathcal{R}^r} \cap (\mathcal{A}_{\mathcal{G}} \cup \bigcup_{r' \in R \setminus \{r\}} \mathcal{A}_{\mathcal{R}^{r'}}) = \emptyset$ and (2) no part of $\mathcal{G}$ matched on by more than one rule is altered, i.e. for all $r_1, r_2 \in R$, $m_{r_1}$, $m_{r_2}$, let $\mathcal{G}' = m_{r_1}(\mathcal{L}^{r_1}) \cap m_{r_2}(\mathcal{L}^{r_2})$, then if $\mathcal{G}' \neq \emptyset$, we have $\mathcal{G}' \cap \hat{m}_{r_1}(\mathcal{L}^{r_1}) = \emptyset$ and $\mathcal{G}' \cap \hat{m}_{r_2}(\mathcal{L}^{r_2}) = \emptyset$. These conditions ensure that rule application on one match (1) does not result in new matches, and (2) removes exactly one match. These conditions can be checked straightforwardly.

For terminating, confluent $\Sigma$, applying all $r \in R$ as often as possible results in a particular LTS, independent of the order of rule application. We refer to that LTS as $T_R^+(\mathcal{G})$. Finally, we define the *transformation of a network of* LTS*s*.

**Definition 10 (Network transformation).** *Given a network $\mathcal{M} = (\Pi, \mathcal{V})$ and a rule system $\Sigma = (R, \hat{\mathcal{V}})$, the transformed network is defined as follows:*

$$T_\Sigma(\mathcal{M}) = (\{T_R^+(\mathcal{G}) \mid \mathcal{G} \in \Pi\}, \mathcal{V} \cup \hat{\mathcal{V}})$$

## 5 General $\varphi$-Preservation Under Transformations

In this section, we show how property preservation of transformations can be checked by generating networks from rule systems, and comparing the LTSs of these networks. Fig. 5 gives an overview of the approach.

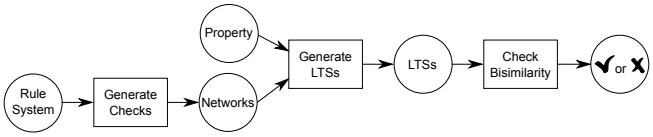



**Fig. 5:** Checking property preservation by comparing LTSs

First, networks are generated based on the left and right-hand sides of transformation rules. Then, the network LTSs corresponding to these networks are

generated, while applying maximal hiding w.r.t. the property at hand. Finally, property preservation is checked by comparing the network LTSs generated from left-hand sides of transformation rules with the network LTSs of the corresponding right-hand sides.

### 5.1 The Setting

Consider a design for a concurrent system, formalised by a network $\mathcal{M}_y = (\Pi_y, \mathcal{V}_y)$ describing an LTS $(\mathcal{S}_y, \mathcal{T}_y, \mathcal{A}_y, \mathcal{I}_y)$, with $y \in \mathbb{N}$, a property $\varphi \in L_\mu^{dsbr}$, and a terminating, confluent rule system $\Sigma_y$ formally describing a design step to a new network $\mathcal{M}_{y+1} = T_{\Sigma_y}(\mathcal{M}_y)$. We call $\Sigma_y$ *$\varphi$-preserving* iff $\models_{\mathcal{M}_y} \varphi \iff \models_{\mathcal{M}_{y+1}} \varphi$, regardless of the structures of the LTSs $\mathcal{M}_y$, $\mathcal{M}_{y+1}$, as long as $\Sigma_y$ is confluent w.r.t. $\mathcal{M}_y$. If we know that $\models_{\mathcal{M}_y} \varphi$ and $\Sigma_y$ is $\varphi$-preserving, it means that $\varphi$ does not need to be rechecked in $\mathcal{M}_{y+1}$. Since $L_\mu^{dsbr}$ is compatible with maximal hiding and divergence-sensitive branching bisimilarity, $\Sigma_y$ is $\varphi$-preserving if $\tilde{\tau}_\varphi(\mathcal{M}_y) \leftrightarrows_b^\Delta \tilde{\tau}_\varphi(\mathcal{M}_{y+1})$. In this section, we discuss under which conditions a rule system implies the bisimilarity of $\tilde{\tau}_\varphi(\mathcal{M}_y)$ and $\tilde{\tau}_\varphi(\mathcal{M}_{y+1})$. The most important condition roughly boils down to checking whether, after some appropriate rewriting, the left and right patterns of the transformation rules are divergence-sensitive branching bisimilar after maximal hiding. If this is the case, then applying the rules does not result in a structurally different network LTS, given that the rules satisfy the dangling condition; if they do not, state reachability may be altered through transformation. In Section 5.2, we will first look at applying a single transformation rule, after which we will consider the more general case with multiple rules in a rule system in Section 5.3. Without loss of generality, we assume that each $r \in R_y$ has exactly one match to some $\Pi_y[i]$, and that each $\Pi_y[i]$ is matched on by exactly one $r$. This is expressed by indexing the $r \in R_y$; rule $r_i$ is matched on $\Pi_y[i]$. If $R_y$ contains only one rule, we omit its index. Since $\Sigma_y$ is confluent, the results of this section can be lifted to the more general case where rules may have an arbitrary number of matches. With this assumption, it can also safely be assumed that all the $\mathcal{A}_i$ are disjoint. If this is not the case, some renaming of actions can resolve this.

### 5.2 Applying a Single Transformation Rule

Given a rule $r : \mathcal{L}^r \mapsto \mathcal{R}^r$, let $\mathcal{L}_\kappa^r$ be $\langle \mathcal{S}_{\mathcal{L}^r}, \mathcal{A}_{\mathcal{L}^r} \cup \{\kappa_s \mid s \in \mathcal{S}_{\mathcal{L}^r} \cap \mathcal{S}_{\mathcal{R}^r}\}, \mathcal{T}_{\mathcal{L}^r} \cup \{\langle s, \kappa_s, s\rangle \mid s \in \mathcal{S}_{\mathcal{L}^r} \cap \mathcal{S}_{\mathcal{R}^r}\}, \mathcal{I}_{\mathcal{L}^r}\rangle$. The LTS $\mathcal{L}_\kappa^r$ equals $\mathcal{L}^r$ extended with selfloops on the glue-states, where each loop is labelled with an action uniquely related to that state; we assume that the $\kappa$-actions are not originally in $\mathcal{A}_{\mathcal{L}^r}$.[4] In a similar way, $\mathcal{R}_\kappa^r$ can be derived from a given $\mathcal{R}^r$, and $r_\kappa : \mathcal{L}_\kappa^r \mapsto \mathcal{R}_\kappa^r$ refers to the extended rule $r$.

The $\kappa$-selfloops are added to glue-states in the patterns of $r$ to make explicit that the matches of these states, since they form the interface between the rule matches and other states in $\mathcal{M}_y$, $\mathcal{M}_{y+1}$, may have outgoing transitions not

[4] The Greek word κόλλα means 'glue'.

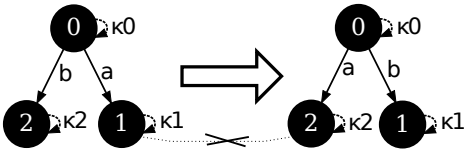



**Fig. 6:** $\kappa$-loops ensure $1 \not\leftrightarrows_b^\Delta 2$

present in the patterns. Without these loops, a divergence-sensitive branching bisimilarity check of patterns could consider two deadlock states to be bisimilar, while they are actually different glue-states that are possibly matched on states with different outgoing transitions not present in the patterns (e.g. see Fig. 6, with $a, b \in h_{\mathcal{A}_y}(\varphi)$)). In other words, with these extra transitions, we ensure that $\mathcal{L}^r_\kappa \leftrightarrows_b^\Delta \mathcal{R}^r_\kappa$ iff there exists a divergence-sensitive branching bisimulation where all the glue-states are related to themselves; a glue-state $s$ in $\mathcal{L}^r_\kappa$ (or $\mathcal{R}^r_\kappa$) with selfloop $s \xrightarrow{\kappa_s} s$ must at least be related to itself in $\mathcal{R}^r_\kappa$ (or $\mathcal{L}^r_\kappa$), since it is the only state where a $\kappa_s$-transition is enabled. So, if there exists a divergence-sensitive branching bisimulation $B$ with $s_{I,\mathcal{L}^r_\kappa}\ B\ s_{I,\mathcal{R}^r_\kappa}$, then we know that $B$ is a divergence-sensitive branching bisimulation with $\mathcal{I}_{\mathcal{L}^r}\ B\ \mathcal{I}_{\mathcal{R}^r}$ and $\{(s,s) \mid s \in \mathcal{S}_{\mathcal{L}^r} \cap \mathcal{S}_{\mathcal{R}^r}\} \subseteq B$. This implies the following lemma:

**Lemma 2.** *Let $B$ be a divergence-sensitive branching bisimulation between $\mathcal{L}^r_\kappa$ and $\mathcal{R}^r_\kappa$. Then, the following holds:*

$$\forall s \in \mathcal{S}_{\mathcal{L}^r} \cap \mathcal{S}_{\mathcal{R}^r}, s' \in \mathcal{S}_{\mathcal{R}^r}.s\ B\ s' \Rightarrow (s' \Rightarrow_{\mathcal{R}^r} s)$$

*Proof.* Since $s \in \mathcal{S}_{\mathcal{L}^r} \cap \mathcal{S}_{\mathcal{R}^r}$, we have $s \xrightarrow{\kappa_s}_{\mathcal{L}^r_\kappa} s$ (and $s \xrightarrow{\kappa_s}_{\mathcal{R}^r_\kappa} s$). Also, since $s\ B\ s'$, $\kappa_s \neq \tau$ and $B$ is a divergence-sensitive branching bisimulation, by Def. 3, there exist $\hat{s}, s'' \in \mathcal{S}_{\mathcal{R}^r_\kappa}$ with $s' \Rightarrow_{\mathcal{R}^r_\kappa} \hat{s} \xrightarrow{\kappa_s}_{\mathcal{R}^r_\kappa} s''$. Because $s$ is the only state from which a $\kappa_s$-transition can be fired, we must have $\hat{s} = s$, therefore $s' \Rightarrow_{\mathcal{R}^r_\kappa} s$. □

In the following, we refer with $m'_r : \mathcal{R}^r \mapsto T^m_r(\mathcal{G})$ to a match between the right-hand side of $r$ and $T^m_r(\mathcal{G})$, corresponding with a match $m_r : \mathcal{L}^r \mapsto \mathcal{G}$ such that for all glue-states $s$, we have $m_r(s) = m'_r(s)$. The functions ${m_r^{-1}}'$ and $\hat{m}'_r$ are defined similar to $m_r^{-1}$ and $\hat{m}_r$, respectively.

The following proposition directly gives rise to a $\varphi$-preservation check for rule systems consisting of a single rule, i.e. systems $\Sigma_y$ with $R_y = \{r\}$.

**Proposition 2.** *Let $\mathcal{G}$ be an* LTS*, $r : \mathcal{L}^r \mapsto \mathcal{R}^r$ be a transformation rule, and $m_r : \mathcal{L}^r \mapsto \mathcal{G}$ be a match. Now, the following holds:*

$$\mathcal{L}^r_\kappa \leftrightarrows_b^\Delta \mathcal{R}^r_\kappa \Rightarrow \mathcal{G} \leftrightarrows_b^\Delta T^m_r(\mathcal{G})$$

*Proof.* By definition, we have $\mathcal{G} \leftrightarrows_b^\Delta T^m_r(\mathcal{G})$ iff there exists a divergence-sensitive branching bisimulation $C$ such that $\mathcal{I}_\mathcal{G}\ C\ \mathcal{I}_{T^m_r(\mathcal{G})}$. We will show that such a bisimulation can be constructed.

Obviously, $\mathcal{G} \leftrightarrows_b^{\Delta} \mathcal{G}$, e.g. consider the identity relation $B = \{(s,s) \mid s \in \mathcal{S}_{\mathcal{G}}\}$, which is in fact a strong bisimulation. Furthermore, since $\mathcal{L}_\kappa^r \leftrightarrows_b^{\Delta} \mathcal{R}_\kappa^r$, there exists a divergence-sensitive branching bisimulation relation $\hat{B}$ with $\mathcal{I}_{\mathcal{L}^r} \hat{B} \mathcal{I}_{\mathcal{R}^r}$ and $\{(s,s) \mid s \in \mathcal{S}_{\mathcal{L}^r} \cap \mathcal{S}_{\mathcal{R}^r}\} \subseteq \hat{B}$. Now we construct a divergence-sensitive branching bisimulation $C$ between $\mathcal{G}$ and $T_r^m(\mathcal{G})$ as follows: $C = B' \cup \hat{B}'$, with $B' = \{(s,s) \mid (s,s) \in B \wedge s \in \mathcal{S}_{\mathcal{G}} \setminus \hat{m}_r(\mathcal{S}_{\mathcal{L}^r})\}$ and $\hat{B}' = \{(m_r(s), m'_r(p)) \mid (s,p) \in \hat{B}\}$. We show that $C$ is a divergence-sensitive branching bisimulation by proving that for all $(s,p) \in C$, Def. 3 holds. We distinguish two cases:
(I) $(s,p) \in B'$, hence $s = p$. Again, we distinguish two cases:

1. $s \xrightarrow{a}_{\mathcal{G}} s'$. We distinguish two cases:
   (a) $s' \in \mathcal{S}_{\mathcal{G}} \setminus \hat{m}_r(\mathcal{S}_{\mathcal{L}^r})$, and we have $s \; B' \; s$ and $s' \; B' \; s'$;
   (b) $s' \in \hat{m}_r(\mathcal{S}_{\mathcal{L}^r})$. By Lemma 1, $s \in m_r(\mathcal{S}_{\mathcal{L}^r})$. Then, $m_r^{-1}(s) \xrightarrow{a}_{\mathcal{L}^r} m_r^{-1}(s')$. Since $s \; \hat{B}' \; s$, also $m_r^{-1}(s) \; \hat{B} \; m_r^{-1'}(s)$ and either $a = \tau$ and $m_r^{-1}(s') \; \hat{B} \; m_r^{-1'}(s)$, hence also $s' \; \hat{B}' \; s$, or there exist $\hat{p}, p' \in \mathcal{S}_{\mathcal{R}^r}$ with $m_r^{-1'}(s) \Rightarrow_{\mathcal{R}^r} \hat{p} \xrightarrow{a}_{\mathcal{R}^r} p'$ and $m_r^{-1}(s) \; \hat{B} \; \hat{p}$ and $m_r^{-1}(s') \; \hat{B} \; p'$, hence also $s \Rightarrow_{T_r^m(\mathcal{G})} m'_r(\hat{p}) \xrightarrow{a}_{T_r^m(\mathcal{G})} m'_r(p')$ and $s \; \hat{B}' \; m'_r(\hat{p})$ and $s' \; \hat{B}' \; m'_r(p')$.
2. $s \uparrow$ (there exists an infinite path $\sigma$ of $\tau$-transitions in $\mathcal{G}$). We show that such a path also exists in $T_r^m(\mathcal{G})$. Let $s_1, s_2, \ldots$ be the sequence of states in $m_r(\mathcal{S}_{\mathcal{L}^r}) \setminus \hat{m}_r(\mathcal{S}_{\mathcal{L}^r})$ as they appear in $\sigma$. For each $s_i$, we distinguish two cases:
   (a) There exists an $s_{i+1}$ in the sequence. Then, $m_r^{-1}(s_i) \Rightarrow_{\mathcal{L}^r} m_r^{-1}(s_{i+1})$ and since $m_r^{-1}(s_i) \; \hat{B} \; m_r^{-1'}(s_i)$, also $m_r^{-1}(s_{i+1}) \; \hat{B} \; m_r^{-1'}(s_i)$, and by Lemma 2, $m_r^{-1'}(s_i) \Rightarrow_{\mathcal{R}^r} m_r^{-1'}(s_{i+1})$, hence $s_i \Rightarrow_{T_r^m(\mathcal{G})} s_{i+1}$.
   (b) There does not exist an $s_{i+1}$ in the sequence. Then, $s_i \uparrow$ in $\mathcal{L}^r$. Since $s_i \; \hat{B}' \; s_i$, also $s_i \uparrow$ in $\mathcal{R}^r$, hence in $T_r^m(\mathcal{G})$.

(II) $(s,p) \in \hat{B}'$. We distinguish two cases:

1. $s \xrightarrow{a}_{\mathcal{G}} s'$. We distinguish two cases:
   (a) $s' \in \mathcal{S}_{\mathcal{G}} \setminus m_r(\mathcal{S}_{\mathcal{L}^r})$. By Lemma 1, $s \in m_r(\mathcal{S}_{\mathcal{L}^r}) \setminus \hat{m}_r(\mathcal{S}_{\mathcal{L}^r})$. By Lemma 2, $m_r^{-1'}(p) \Rightarrow_{\mathcal{R}^r} m_r^{-1'}(s)$, hence $p \Rightarrow_{T_r^m(\mathcal{G})} s$, and we have $s \; B' \; s$ and $s' \; B' \; s'$.
   (b) $s' \in m_r(\mathcal{S}_{\mathcal{L}^r})$. Then, $m_r^{-1}(s) \xrightarrow{a}_{\mathcal{L}^r} m_r^{-1}(s')$. Since $s \; \hat{B}' \; s$, also $m_r^{-1}(s) \; \hat{B} \; m_r^{-1'}(s)$ and either $a = \tau$ and $m_r^{-1}(s') \; \hat{B} \; m_r^{-1'}(s)$, hence also $s' \; \hat{B}' \; s$, or there exist $\hat{p}, p' \in \mathcal{S}_{\mathcal{R}^r}$ with $m_r^{-1'}(s) \Rightarrow_{\mathcal{R}^r} \hat{p} \xrightarrow{a}_{\mathcal{R}^r} p'$ and $m_r^{-1}(s) \; \hat{B} \; \hat{p}$ and $m_r^{-1}(s') \; \hat{B} \; p'$, hence also $s \Rightarrow_{T_r^m(\mathcal{G})} m'_r(\hat{p}) \xrightarrow{a}_{T_r^m(\mathcal{G})} m'_r(p')$ and $s \; \hat{B}' \; m'_r(\hat{p})$ and $s' \; \hat{B}' \; m'_r(p')$.
2. $s \uparrow$. The proof for case (I).2 is also applicable here. □

By Props. 1 and 2, if $\mathcal{L}_\kappa^r \leftrightarrows_b^{\Delta} \mathcal{R}_\kappa^r$, then $\models_{\mathcal{G}} \varphi \iff \models_{T_r^m(\mathcal{G})} \varphi$. Moreover, due to the compatibility of $L_\mu^{dsbr}$ with maximal hiding, this also holds if we hide all actions in $h_{\mathcal{A}_{\mathcal{G}}}(\varphi)$ and $h_{\mathcal{A}_r^m}(\varphi)$ (note that the $\kappa$-actions are *not* in $\mathcal{A}_{\mathcal{G}}$ and $\mathcal{A}_r^m$), i.e. if $\tilde{\tau}_\varphi(\mathcal{L}_\kappa^r) \leftrightarrows_b^{\Delta} \tilde{\tau}_\varphi(\mathcal{R}_\kappa^r)$, then $\models_{\mathcal{G}} \varphi \iff \models_{T_r^m(\mathcal{G})} \varphi$.

### 5.3 Multiple Transformation Rules and Synchronising Behaviour

If a rule system consists of *multiple* transformation rules, then multiple transformations can be applied in a single transformation step. To check $\varphi$-preservation of such rule systems, we need to take possible synchronisation between different rule patterns into account. For example, consider the network $\mathcal{M}_0 = (\langle A, B\rangle, \{(\langle a, b\rangle, c)\})$, with $A = s_1 \xrightarrow{a} s_2$ and $B = s_3 \xrightarrow{b} s_4$,[5] and the rule system $\Sigma_0 = (\{r_1, r_2\}, \{(\langle a', b'\rangle, c)\}\})$ with $r_1 = (p_1 \xrightarrow{a} p_2) \mapsto (p_1 \xrightarrow{a'} p_2)$ and $r_2 = (p_3 \xrightarrow{b} p_4) \mapsto (p_3 \xrightarrow{b'} p_4)$. Note that $r_1$ can be matched on $A$, and $r_2$ can be matched on $B$. The two rules affect two different actions involved in the same synchronisation rule. Individually, the rules seem to fundamentally change the system behaviour, but since the rule system also adds the new synchronisation rule $(\langle a', b'\rangle, c)$, the final network Lts is equal to the one before transformation. To incorporate such possible dependencies between rule patterns, we developed a $\varphi$-preservation check involving *networks* of rule patterns.

In this section, we will first discuss some required notations to reason about combinations of patterns and combinations of states of the different process Ltss in a network. After that, we will formulate the generalised $\varphi$-preservation check and prove it correct.

First of all, remember that we assume that $r_i$ is matched on $\Pi[i]$. The set of id's of process Ltss that potentially can synchronise with behaviour in $\mathcal{L}^{r_i}$ is $dep(\mathcal{L}^{r_i}) = \bigcup\{Ac(\bar{t}) \mid (\bar{t}, a) \in \mathcal{V}_y \wedge \bar{t}[i] \in \mathcal{A}_{\mathcal{L}^{r_i}}\}$. This means that $j$ is in $dep(\mathcal{L}^{r_i})$ iff there exists a synchronisation rule $(\bar{t}, a)$ in $\mathcal{V}_y$ such that both $i \neq \bullet$ and $j \neq \bullet$, i.e. both $i$ and $j$ are active for that rule, and the behaviour in $\Pi[i]$ is matched on by $r_i$. The set of actions of process $j$ on which $\mathcal{L}^{r_i}$ depends is $A^{\mathcal{L}^{r_i}}_{dep}(j) = \{\bar{t}[j] \mid (\bar{t}, a) \in \mathcal{V}_y \wedge \bar{t}[i] \in \mathcal{A}_{\mathcal{L}^{r_i}} \wedge \bar{t}[j] \neq \bullet\}$. Say that the set of all rules applicable on $\mathcal{L}^{r_i}$ is called $F$, then the set of all actions $\bigcup_{t\in F}\{\bar{t}[j]\} \setminus \{\bullet\}$ constitutes $A^{\mathcal{L}^{r_i}}_{dep}(j)$. Sets $dep(\mathcal{R}^{r_i})$ and $A^{\mathcal{R}^{r_i}}_{dep}(j)$ are defined similarly.

When considering transformation rules which affect synchronisation actions, i.e. involving multiple process Ltss, one quickly realises that in general, one cannot determine whether a given rule system $\Sigma_y$ involving such rules is $\varphi$-preserving by just analysing the $\mathcal{L}^{r_i}$ and $\mathcal{R}^{r_i}$ of all $r_i \in R_y$. However, we prove that this can be done if $\Sigma_y$ has a number of properties w.r.t. synchronising behaviour in $\mathcal{M}_y$, which we together call *synchronisation uniformity*.

**Definition 11 (Synchronisation uniformity).** *We say that $\Sigma_y$ is* synchronisation uniform *w.r.t. $\mathcal{M}_y$ iff the following holds:*

1. $\forall a \in \mathcal{A}_s.(\exists r_i \in R_y.a \in \mathcal{A}_{\mathcal{L}^{r_i}}) \Rightarrow \forall s_1 \xrightarrow{a}_i s_2.s_1, s_2 \in m_{r_i}(\mathcal{S}_{\mathcal{L}^{r_i}})$;
2. $\forall r_i \in R_y, j \in dep(\mathcal{L}^{r_i}).A^{\mathcal{L}^{r_i}}_{dep}(j) \subseteq \mathcal{A}_{\mathcal{L}^{r_j}}$;
3. $\forall(\bar{t}, a) \in \hat{\mathcal{V}}_y, i \in 1..n.\bar{t}[i] = \bullet \vee \bar{t}[i] \in \mathcal{A}_{\mathcal{R}^{r_i}}$

[5] For convenience, we consider a transition $s \xrightarrow{a} s'$ to be shorthand for the Lts $(\{s, s'\}, \{a\}, \{\langle s, a, s'\rangle\}, \{s\})$.

Condition 1 states that if a transformation rule is applicable to a synchronising transition, then it is applicable to *all* synchronising transitions with that label in $\mathcal{M}_y$. If this is not guaranteed, it becomes very hard to reason about $\mathcal{M}_{y+1}$; it is difficult to determine a priori exactly which transitions in different process LTSs will be able to synchronise in the network, so predicting the effect of rewriting e.g. $a$-transitions in some places, but keeping other $a$-transitions the same is as difficult.[6] Condition 2 says that all actions that can synchronise with $\mathcal{L}^{r_i}$ are also being transformed by $\Sigma_y$. If this does not hold, it becomes hard to analyse the synchronising behaviour as appearing in transformation patterns, since some of the involved behaviour is outside their scope. Finally, condition 3 says that each new synchronisation rule $(\bar{t}, a) \in \hat{\mathcal{V}}_y$ only involves actions from the $\mathcal{R}^{r_i}$ of the corresponding $r_i$. This condition is not referred to anymore later on, but it is crucial to rule out the possibility of transforming merely by introducing synchronisation rules; e.g. if we define a new synchronisation rule involving existing actions $a$, $b$, and these actions were previously not allowed to synchronise, then we clearly change the model without actually transforming anything.

In the remainder of this report, we consider $\Sigma_y$ to be synchronisation uniform w.r.t. $\mathcal{M}_y$. This may seem a big assumption, but in practice, one tends to transform synchronising behaviour in a uniform way; usually, synchronising actions, say $a$ and $b$, represent communication over some channel. If one wants to transform this behaviour, i.e. change the details about the communication, it is natural (1) to do this consistently in all places where $a$ and $b$ occur, and (2) not only transform e.g. $a$, but also $b$ to keep both sides in the communication compatible with each other.

*State vectors.* A state in $\mathcal{S}_y$ (and $\mathcal{S}_{y+1}$) is a vector $\overline{s} = \langle \overline{s}[1], \ldots, \overline{s}[n] \rangle$. An arbitrary $\overline{s} \in \mathcal{S}_y$ can have up to $n$ elements that are matched on by some transformation rule. We denote the set of indices of elements in $\overline{s}$ matched on by the corresponding rule with $M(\overline{s}) = \{i \mid \overline{s}[i] \in m_{r_i}(\mathcal{S}_{\mathcal{L}^{r_i}})\}$. In a similar way, $\hat{M}(\overline{s}) = \{i \mid \overline{s}[i] \in \hat{m}_{r_i}(\mathcal{S}_{\mathcal{L}^{r_i}})\}$. Again, note that we assume that if $\overline{s}[i]$ is matched on, then it is matched on by rule $r_i$. We will now formulate a number of lemma's.

**Lemma 3.** $\overline{s} \xrightarrow{a}_{\mathcal{M}_y} \overline{s}' \Rightarrow \hat{M}(\overline{s}') \subseteq M(\overline{s}) \wedge \hat{M}(\overline{s}) \subseteq M(\overline{s}')$

*Proof.* $(\hat{M}(\overline{s}') \subseteq M(\overline{s}))$ Let $i \in \hat{M}(\overline{s}')$, i.e. $\overline{s}'[i] \in \hat{m}_{r_i}(\mathcal{S}_{\mathcal{L}^{r_i}})$, so there exists $\hat{m}_{r_i}^{-1}(\overline{s}'[i]) \in \mathcal{S}_{\mathcal{L}^{r_i}} \setminus \mathcal{S}_{\mathcal{R}^{r_i}}$. Let $(\bar{t}, a)$ enable $\overline{s} \xrightarrow{a}_{\mathcal{M}_y} \overline{s}'$. Either $i \in Ac(\bar{t})$, then $\overline{s}[i] \xrightarrow{\bar{t}[i]}_i \overline{s}'[i]$ and by Def. 7, there exists an $\hat{s} \in \mathcal{S}_{\mathcal{L}^{r_i}}$ with $m_{r_i}(\hat{s}) = \overline{s}[i]$, so $i \in M(\overline{s})$, or $i \notin Ac(\bar{t})$ and $\overline{s}'[i] = \overline{s}[i]$. Then $i \in \hat{M}(\overline{s})$, so $i \in M(\overline{s})$. With the same reasoning, $(\hat{M}(\overline{s}) \subseteq M(\overline{s}'))$ holds. □

The underlying principle making Lemma 3 valid is the way transformation rules are matched (Def. 7); specifically, dangling edges cannot occur. If state $\overline{s}[i]$

[6] Note that condition 1 and the fact that $\Sigma_y$ is confluent implies that if a synchronising $a$-transition is transformed by a rule, then only this rule transforms all transitions labelled $a$ in $\mathcal{M}_y$.

(resp. $\overline{s}'[i]$) is matched on, but not a glue-state, i.e. $i \in \hat{M}(\overline{s})$ (resp. $i \in \hat{M}(\overline{s}')$), then the corresponding state $\overline{s}'[i]$ (resp. $\overline{s}[i]$) must also be matched on. If $i$ is active for the transition from $\overline{s}$ to $\overline{s}'$, then this follows from Def. 7. If $i$ is not active, then this follows from the fact that $\overline{s}[i] = \overline{s}'[i]$.

It follows from the definitions of $Ac(\overline{t})$ and $dep(\mathcal{L}^{r_i})$ that if in a state $\overline{s}$, $\overline{s}[i]$ is matched on by rule $r_i$ and $i$ is active for a synchronisation rule $(\overline{t}, a)$, then $Ac(\overline{t}) \subseteq dep(\mathcal{L}^{r_i})$:

**Lemma 4.** $\overline{s} \xrightarrow{a}{}^{\overline{t}}_{\mathcal{M}_y} \overline{s}' \wedge \exists i \in M(\overline{s}) \cap M(\overline{s}') \cap Ac(\overline{t}) \Rightarrow Ac(\overline{t}) \subseteq dep(\mathcal{L}^{r_i})$

*Proof.* Let $i \in M(\overline{s}) \cap M(\overline{s}') \cap Ac(\overline{t})$. Since $i \in M(\overline{s})$ and $i \in M(\overline{s}')$, we have $\overline{s}[i], \overline{s}'[i] \in m_{r_i}(\mathcal{S}_{\mathcal{L}^{r_i}})$. Since $\overline{s}[i] \xrightarrow{\overline{t}[i]}_i \overline{s}'[i]$, due to the isomorphism between $\mathcal{L}^{r_i}$ and $m_{r_i}(\mathcal{S}_{\mathcal{L}^{r_i}})$, also $m_{r_i}^{-1}(\overline{s}[i]) \xrightarrow{\overline{t}[i]}_{\mathcal{L}^{r_i}} m_{r_i}^{-1}(\overline{s}'[i])$, hence $\overline{t}[i] \in \mathcal{A}_{\mathcal{L}^{r_i}}$, so by the def. of $dep(\mathcal{L}^{r_i})$, $Ac(\overline{t}) \subseteq dep(\mathcal{L}^{r_i})$. Therefore, by Def. 11, case 2, for all $j \in Ac(\overline{t})$, $A_{dep}^{\mathcal{L}^{r_i}}(j) \subseteq \mathcal{A}_{\mathcal{L}^{r_j}}$, so $\overline{t}[j] \in \mathcal{A}_{\mathcal{L}^{r_j}}$ and by def. of $dep(\mathcal{L}^{r_j})$, $Ac(\overline{t}) \subseteq dep(\mathcal{L}^{r_j})$. □

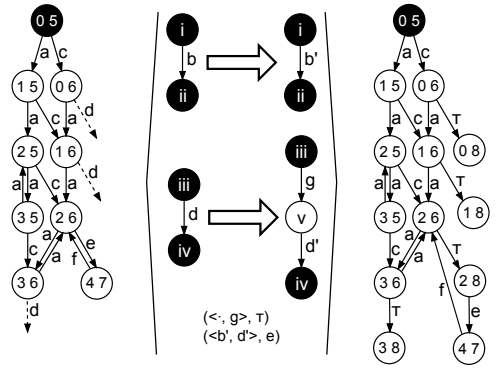
From Def. 11, case 2, it follows that if outgoing transitions from $\overline{s}[i]$ can potentially synchronise with outgoing transitions from some $\overline{s}[j]$, and if $\overline{s}[i]$ is subject to transformation, i.e. is matched on by $r_i$, then $\overline{s}[j]$ is subject to transformation as well, i.e. is matched on by $r_j$. This fact is proven in the following lemma:

**Lemma 5.** $\overline{s} \xrightarrow{a}{}^{\overline{t}}_{\mathcal{M}_y} \overline{s}' \wedge M(\overline{s}) \cap M(\overline{s}') \cap Ac(\overline{t}) \neq \emptyset \Rightarrow Ac(\overline{t}) \subseteq M(\overline{s})$

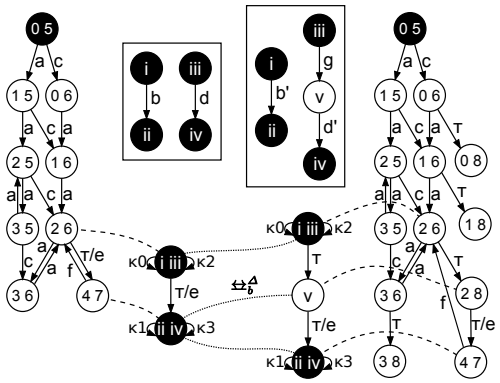
*Proof.* Let $i \in M(\overline{s}) \cap M(\overline{s}') \cap Ac(\overline{t})$. We distinguish two cases:

1. $|Ac(\overline{t})| = 1$, then trivially, since $i \in M(\overline{s})$, $Ac(\overline{t}) \subseteq M(\overline{s})$.
2. $|Ac(\overline{t})| > 1$. Hence, $A(\overline{t}) \subseteq \mathcal{A}_s$. By Lemma 4, $Ac(\overline{t}) \subseteq dep(\mathcal{L}^{r_i})$, so by Def. 11, case 2, for all $j \in Ac(\overline{t})$, $A_{dep}^{\mathcal{L}^{r_i}}(j) \subseteq \mathcal{A}_{\mathcal{L}^{r_j}}$. Clearly, $\overline{t}[j] \in A_{dep}^{\mathcal{L}^{r_i}}(j)$, and since $\overline{t}[j] \in \mathcal{A}_s$, by Def. 11, case 1, for all $j \in Ac(\overline{t})$, we have for all $s_1 \xrightarrow{\overline{t}[j]}_j s_2$ that $s_1, s_2 \in m_{r_j}(\mathcal{S}_{\mathcal{L}^{r_j}})$, so $\overline{s}[j], \overline{s}'[j] \in m_{r_j}(\mathcal{S}_{\mathcal{L}^{r_j}})$, i.e. $j \in M(\overline{s})$. □

*Networks of transformation rules.* In order to reason about synchronisations within transformation patterns, we need to consider appropriate pattern combinations that capture both successful and unsuccessful synchronisation. Given a network of LTSs which involves synchronising behaviour, we cannot easily determine a priori, i.e. without constructing the explicit network LTS, whether or not successful synchronisation can occur in some states in the network LTS, and whether or not there will be reachable states in the network LTS from which an unsuccessful attempt to synchronise can be performed. In other words, we do not know a priori which scenarios related to synchronisation are actually relevant for the network. Therefore, to reason about $\varphi$-preservation in general, we will have to take all possible scenarios into account. This is illustrated in Fig. 7(a). On the left, the LTS of the network in Fig. 2 is shown, now with unsuccessful attempts of $\Pi[2]$ to synchronise displayed as dashed transitions, i.e. the transition




**(a)**



**(b)**

**Fig. 7:** Transformation of successful and unsuccessful synchronising behaviour

which is enabled in $\Pi[2]$ is shown, but since the corresponding required transition in $\Pi[1]$ is not enabled, synchronisation fails, resulting in no transition in the network Lts. If we apply the rule system in the middle of Fig. 7(a), which transforms $d$-transitions in $\Pi[2]$ (see Fig. 2) and introduces two new synchronisation rules, then the network Lts after transformation suddenly contains a number of deadlock states, resulting from the fact that in multiple states, a $d$-transition is enabled from $\Pi[2]$, but no $b$-transition is enabled in $\Pi[1]$.

We will combine rule patterns into networks to reason about different scenarios. Let $\xi^{r_i}$ be a vector of transformation rules relevant for the behaviour in $\mathcal{L}^{r_i}$. Formally, for all $j \in 1..n$, we have the following, with $*$ a dummy state:

$$\xi^{r_i}[j] = \begin{cases} * \mapsto * \text{ if } j \notin dep(\mathcal{L}^{r_i}) \\ r_{j,\kappa} \quad \text{if } j \in dep(\mathcal{L}^{r_i}) \end{cases}$$

For a given vector $\xi^{r_i}$, $\xi^{r_i}_{\mathcal{L}}$ is the vector of left patterns of the (extended) rules in $\xi^{r_i}$, and $\xi^{r_i}_{\mathcal{R}}$ is the vector of right patterns. The networks $\Xi^{r_i}_{\mathcal{L}} = (\xi^{r_i}_{\mathcal{L}}, \mathcal{V}_y)$ and

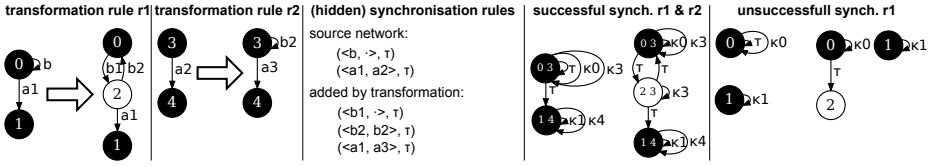



**Fig. 8:** Transforming synchronising behaviour requires analysing all scenarios

$\Xi^{r_i}_{\mathcal{R}} = (\xi^{r_i}_{\mathcal{R}}, \mathcal{V}_y \cup \hat{\mathcal{V}}_y)$ allow comparing synchronising behaviour in rule patterns, before and after the transformation according to $\Sigma_y$, in particular involving $r_i$.

Finally, we want to construct vectors based on $\xi^{r_i}$ where some parties are absent, to reason about unsuccessful synchronisation. Fig. 8 shows an example inspired by a case study we performed involving the transformation of multi-party synchronisation into multiple two-party synchronisations: on the left, there are two rules $r_1$ and $r_2$ that apply on synchronisation actions $a_1$, $a_2$, and new synchronisation rules are introduced by transformation. On the right of the synchronisation rules, successful synchronisation of the patterns of $r_1$ and $r_2$ before and after transformation is shown. These networks are divergence-sensitive branching bisimilar. However, next to that, the scenario where $a_1$ is enabled but not $a_2$ is shown ($*$'s are omitted), i.e. only the behaviour described by $r_1$ is relevant, which may hold for some $\overline{s} \in \mathcal{S}_y$. After transformation, a transition from 0 to 2 is possible, leading to a deadlock. Before the transformation, this was not possible. Hence, the transformations are not $\varphi$-preserving. To consider different scenarios, we define a projection operator $/I$ ($I \subseteq 1..n$): for each $j \in 1..n$, $\xi^{r_i}/I[j] = \xi^{r_i}[j]$ if $j \in I$, otherwise $\xi^{r_i}/I[j] = * \mapsto *$. This operator can similarly be applied on the $\xi^{r_i}_{\mathcal{L}}$ and $\xi^{r_i}_{\mathcal{R}}$, and we say that $\Xi^{r_i}_{\mathcal{L}}/I = (\xi^{r_i}_{\mathcal{L}}/I, \mathcal{V}_y)$ and $\Xi^{r_i}_{\mathcal{R}}/I = (\xi^{r_i}_{\mathcal{R}}/I, \mathcal{V}_y \cup \hat{\mathcal{V}}_y)$.

Similar to matches $m_{r_i}$ for a single rule $r_i$, we can define how state vectors in $\Xi^{r_i}_{\mathcal{L}}/I = (\xi^{r_i}_{\mathcal{L}}/I, \mathcal{V}_y)$ can be matched on states of $\mathcal{M}_y$, by means of referring to the matches of individual vector elements. Due to the presence of $*$-states, the $\xi^{r_i}_{\mathcal{L}}/I$ ($\xi^{r_i}_{\mathcal{R}}/I$) can possibly be matched to multiple states in $\mathcal{S}_y$ ($\mathcal{S}_{y+1}$). We formalise this mapping as follows:

**Definition 12 (Mapping of state vectors).** *A state $\overline{s}^* \in \mathcal{S}_{\Xi^{r_i}_{\mathcal{L}}/I}$ is mapped to a state $\overline{s} \in \mathcal{S}_y$, denoted as $\overline{s}^* \vdash^{r_i,I}_{\mathcal{L}} \overline{s}$, iff for all $j \in I$:*

$$\overline{s}^*[j] \neq * \Rightarrow \overline{s}[j] = m_{\xi^{r_i}/I[j]}(\overline{s}^*[j])$$

*That is, all states $\overline{s}^*[j]$, with $j \in I$, are matched by rule $\xi^r_i[j]$, i.e. $r_j$, to $\overline{s}[j]$.*

This mapping actually constitutes a simulation relation between state vectors: $\overline{s}^* \vdash^{r_i,I}_{\mathcal{L}} \overline{s}$ implies that the states $\{\overline{s}[i] \mid i \in I\}$ in state $\overline{s}$ together can simulate the combined behaviour of the states $\{\overline{s}^*[i] \mid i \in I\}$ in state $\overline{s}^*$. This is expressed in the following lemma:

**Lemma 6.** *For all $r_i \in R_y$, $I \subseteq 1..n$, $\overline{s}^* \in \mathcal{S}_{\Xi^{r_i}_{\mathcal{L}}/I}$, $\overline{s} \in \mathcal{S}_y$ with $\overline{s}^* \vdash^{r_i,I}_{\mathcal{L}} \overline{s}$, $(\overline{t}, a) \in \mathcal{V}_y$ with $Ac(\overline{t}) \subseteq I : \overline{s}^* \xrightarrow{a}{}_{\Xi^{r_i}_{\mathcal{L}}/I} \overline{s}^{*\prime} \Rightarrow \overline{s} \xrightarrow{a}{}_{\mathcal{M}_y} \overline{s}' \wedge \overline{s}^{*\prime} \vdash^{r_i,I}_{\mathcal{L}} \overline{s}'$*

*Proof.* Consider a $(\bar{t}, a) \in \mathcal{V}$ with $Ac(\bar{t}) \subseteq I \wedge \overline{s}^* \xrightarrow{a}{}_{\Xi^{r_i}_{\mathcal{L}}/I}^{\bar{t}} \overline{s}^{*\prime}$. Since $* \notin Ac(\bar{t})$, by definition of $\vdash^{r_i,I}_{\mathcal{L}}$, we must have for each $\overline{s}^*[j]$ with $j \in Ac(\bar{t})$ that $\overline{s}[j] = m_{\xi^{r_i}/I[j]}(\overline{s}^*[j])$. But then, due to the isomorphisms between $\xi^{r_i}_{\mathcal{L}}/I[j]$ and $m_{\xi^{r_i}/I[j]}(\mathcal{S}_{\xi^{r_i}_{\mathcal{L}}/I[j]})$ for each $j \in Ac(\bar{t})$, $(\bar{t}, a)$ is also applicable in $\overline{s}$, i.e. $\overline{s} \xrightarrow{a}{}_{\mathcal{M}_y}^{\bar{t}} \overline{s}'$. Finally, since $\overline{s}^{*\prime} \in \mathcal{S}_{\Xi^{r_i}_{\mathcal{R}}/I}$, for all $j \in Ac(\bar{t})$, $\overline{s}^{*\prime}[j] \neq *$, and due to the isomorphisms, $\overline{s}'[j] = m_{\xi^{r_i}/I[j]}(\overline{s}^{*\prime}[j])$, therefore $\overline{s}^{*\prime} \vdash^{r_i,I}_{\mathcal{L}} \overline{s}'$. □

We now come to the main proposition formalising our $\varphi$-preservation check. Here, we provide a sketch of its correctness. The full proof can be found in Appendix A.

**Proposition 3.** *Let $\varphi \in L^{dsbr}_{\mu}$ be a temporal property with $\models_{\mathcal{M}_y} \varphi$. Then $\Sigma_y$ is* $\varphi$-preserving *if for all $r_i \in R_y$, $I \subseteq dep(\mathcal{L}^{r_i})$:*

$$\tilde{\tau}_{\varphi}(\Xi^{r_i}_{\mathcal{L}}/I) \leftrightarrows^{\Delta}_{b} \tilde{\tau}_{\varphi}(\Xi^{r_i}_{\mathcal{R}}/I) \tag{1}$$

*Proof sketch.* By def., $\Sigma_y$ is $\varphi$-preserving iff there exists a divergence-sensitive branching bisimulation $C$ showing that $\tilde{\tau}_{\varphi}(\mathcal{M}_y) \leftrightarrows^{\Delta}_{b} \tilde{\tau}_{\varphi}(\mathcal{M}_{y+1})$. Such a bisimulation can be constructed based on (1). We use the fact that rule patterns and their matches are isomorph, i.e. both $\mathcal{L}^{r_i}$ and $m_{r_i}(\mathcal{L}^{r_i})$, and $\mathcal{R}^{r_i}$ and $m_{r_i}(\mathcal{R}^{r_i})$ are isomorph. This also holds for networks of rule patterns; in Fig. 7(b), some rule networks are given for the example of Fig. 2 (with $\tau/e$, we denote that $e$ is hidden after maximal hiding). The combinations of the patterns of the two rules represent successful synchronisation before and after transformation. The state vectors in the left rule network can be matched on state vectors (2 6) and (4 7) in the left system Lts, and the state vectors in the right rule network can be matched on state vectors (2 6), (2 8), and (4 7) in the right system Lts. In fact, these isomorphisms imply *simulation* relations between rule networks and system networks. This can be formalised as follows: for a state vector $\overline{s}^*$ in a rule network $\Xi^{r_i}_{\mathcal{L}}/I$ (resp. $\Xi^{r_i}_{\mathcal{R}}/I$) and a state vector $\overline{s}$ in $\mathcal{M}_y$ (resp. $\mathcal{M}_{y+1}$), we say that $\overline{s}$ simulates $\overline{s}^*$, denoted $\overline{s}^* \vdash \overline{s}$, iff $\forall i \in 1..n. \overline{s}^*[i] \neq * \Rightarrow \overline{s}[i] = m_{r_i}(\overline{s}^*[i])$, i.e. besides the place-holder states, all process states in $\overline{s}^*$ are matched on the corresponding states in $\overline{s}$. Now, first of all, if $\overline{s}^* \vdash \overline{s}$, and $\overline{s}^* \xrightarrow{a} \overline{s}^{*\prime}$, then also $\overline{s} \xrightarrow{a} \overline{s}'$ and $\overline{s}^{*\prime} \vdash \overline{s}'$. Second of all, it also works in the other direction, in cases that $\overline{s} \xrightarrow{a}{}^{\bar{t}} \overline{s}'$, $Ac(\bar{t}) \subseteq I$, and both $Ac(\bar{t}) \subseteq M(\overline{s})$ and $Ac(\bar{t}) \subseteq M(\overline{s}')$. In those cases, the involved behaviour of every active $\Pi[i]$ ($i \in Ac(\bar{t})$) is matched on by $r_i$. These simulation relations are preserved after maximal hiding.

By (1), all left and right rule pattern networks induced by the synchronisation rules are divergence-sensitive branching bisimilar. The union $D$ of all relations constructed from these bisimulations in combination with the simulation relations between rule and system networks (such as in Fig. 7(b), where (2 6) on the left is related to (2 6) on the right, and (4 7) on the left is related to (2 8) and (4 7) on the right) relates all matched on behaviour in the left and right system Ltss. Now consider a relation $D'$ such that for state vectors

$\overline{s} \in \mathcal{S}_y$, $\overline{p} \in \mathcal{S}_{y+1}$, we have $\overline{s}\ D'\ \overline{p}$ iff $\forall i \notin M(\overline{s}).\overline{s}[i] = \overline{p}[i]$. In words, $D'$ relates all state vectors with exactly the same elements apart from those matched on by a transformation rule. Now, we can construct $C$ as $(D \cap D') \cup D''$, with $D'' = \{(\overline{s}, \overline{s}) \mid M(\overline{s}) = \emptyset\}$, i.e. those states are related which can both perform exactly the same non-transformed behaviour, and perform divergence-sensitive branching bisimilar behaviour insofar it has been subjected to transformation. One can prove that $C$ is a divergence-sensitive branching bisimulation by considering the cases of Def. 3 (see Appendix A).[7]

*Complexity.* The divergence-sensitive branching bisimilarity checks of networks of extended $r_i \in R_y$ are "locally persistent" [32], since $\mathcal{M}_y$ is not taken into account. For flat LTSs, reachability cannot be determined by only analysing the changes, but in our setting, process LTS states cannot become unreachable unless specified in a transformation rule, and when network LTS states become unreachable, it follows directly from the transformations applied to the process LTSs. The complexity of each divergence-sensitive branching bisimilarity check is linear to the size of the network of dependent $r_i$ (cf. [16]). Dependency (*dep*) induces a partitioning of the $r_i$ into classes, and per class, the number of checks is exponential to the number of $r_i$ in the class; however, in practice, $r_i$ patterns tend to be very small, and most classes contain only one or two $r_i$. Finally, it is important to note that the size of $\mathcal{M}_{y+1}$ often grows exponentially with the number of matches, but the checks do not.

### 5.4 $\varphi$-Preservation Checking Using Global Divergency Information

In practice, the criteria formulated in Prop. 3 are often too strong. For instance, a check may fail if $\mathcal{L}^{r_i}$ and $\mathcal{R}^{r_i}$ are not equivalent regarding $\tau$-divergence, even if $r_i$ will only be applied on a particular $\mathcal{M}_y$ in places where states are already diverging. Hence, the proposition can be extended by taking (system global) divergency information of $\mathcal{H}_y$ into account.

With $\mathcal{N} \subseteq \mathcal{S}_y$, we refer to all states *not* diverging in $\mathcal{H}_y$. Determining $\mathcal{N}$ involves $\tau$-compression, i.e. reducing $\tau$-cycles to states, and adding a selfloop with a new action $\tau'$ to these states. Now, all states which can reach via a $\tau$-path a $\tau'$-transition are diverging.

We extend the $R_y$ to $R_y^\Delta = \{r_{i\Delta} : \mathcal{L}_\Delta^{r_i} \mapsto \mathcal{R}_\Delta^{r_i} \mid r_i \in R_y\}$, and perform the divergence-sensitive branching bisimilarity checks of Prop. 3 for $R_y^\Delta$ instead of $R_y$. Here, $\mathcal{L}_\Delta^{r_i} = \langle \mathcal{S}_{\mathcal{L}^{r_i}}, \mathcal{A}_{\mathcal{L}^{r_i}} \cup \{\tau\}, \mathcal{T}_{\mathcal{L}^{r_i}} \cup \mathcal{T}_\Delta, \mathcal{I}_{\mathcal{L}^{r_i}} \rangle$, with $\mathcal{T}_\Delta = \{\langle s, \tau, s\rangle \mid s \in \mathcal{S}_{\mathcal{L}^{r_i}} \wedge \neg\exists \overline{p} \in \mathcal{N}.\overline{p}[i] = m_{r_i}(s)\}$, i.e. states not part of non-diverging state vectors in $\mathcal{M}_y$ get a $\tau$-selfloop in $\mathcal{L}^{r_i}$, to make their diverge in $\mathcal{H}_y$ obvious. Pattern $\mathcal{R}_\Delta^{r_i}$ is defined in a similar way.

[7] Note that the checks will correctly detect that the rule system of Fig. 7 does not yield a divergence-sensitive branching bisimulation between the system LTSs, since the unsuccessful synchronisation networks resulting from $(iii) \xrightarrow{d} (iv)$ and $(iii) \xrightarrow{g} (v) \xrightarrow{d'} (iv)$ are not divergence-sensitive branching bisimilar.

It turns out that $\mathcal{N}$ can be updated after a transformation without analysing $\mathcal{H}_{y+1}$. For an $\overline{s}^* \in \mathcal{S}_{\Xi^{r_i}_{\mathcal{L}}/I}$, let $rem(\overline{s}^*) = \{j \mid \overline{s}^*[j] \in \mathcal{L}^{r_j}_{\Delta} \setminus \mathcal{R}^{r_j}\}$ be the indices in $\overline{s}^*$ of states to be removed, and for a $\overline{p}^* \in \mathcal{S}_{\Xi^{r_i}_{\mathcal{R}}/I}$, let $add(\overline{p}^*) = \{j \mid \overline{p}^*[j] \in \mathcal{R}^{r_j} \setminus \mathcal{L}^{r_j}_{\Delta}\}$ be the indices in $\overline{p}^*$ of states to be added. Then updating $\mathcal{N}$ entails doing the following for each $r_{i\,\Delta} \in R^{\Delta}_y$, $I \subseteq dep(\mathcal{L}^{r_i}_{\Delta})$:

1. **Adding:** Note that for all $\overline{p}^* \in \mathcal{S}_{\Xi^{r_i}_{\mathcal{R}}/I}$, there exists $\overline{s}^* \in \mathcal{S}_{\Xi^{r_i}_{\mathcal{L}}/I}$ with $\overline{s}^* \leftrightarrows^{\Delta}_b \overline{p}^*$. If $\overline{p}^* \downarrow$, i.e. does not diverge, and $add(\overline{p}^*) \neq \emptyset$, add all $\langle p_1, \ldots, p_n \rangle$ to $\mathcal{N}$ where for each $j \in 1..n$:

$$p_j = \begin{cases} \overline{p}^*[j] & \text{if } j \in add(\overline{p}^*) \\ \overline{s}[j] & \text{if } \overline{p}^*[j] = * \wedge \overline{s} \in \mathcal{N} \wedge \overline{s}^* \vdash^{r_j, I}_{\mathcal{L}} \overline{s} \\ m_{r_j}(\overline{p}^*[j]) & \text{otherwise} \end{cases}$$

2. **Removing:** $\forall \overline{s}^* \in \mathcal{S}_{\Xi^{r_i}_{\mathcal{L}}/I}. rem(\overline{s}^*) \neq \emptyset \Rightarrow \mathcal{N} := \mathcal{N} \setminus \{\overline{s} \in \mathcal{S}_y \mid \overline{s}^* \vdash^{r_i, I}_{\mathcal{L}} \overline{s}\}$

In words, all state vectors with at least one element removed by transformation need to be removed from $\mathcal{N}$ since they no longer exist in $\mathcal{H}_{y+1}$, and all states with new elements need to be added by mapping existing elements and $*$-elements to the appropriate states in $\mathcal{H}_{y+1}$. Note that remaining states cannot change regarding diverging behaviour. We show this here for the case that for $\overline{s} \in \mathcal{S}_{\mathcal{H}_y}$, $\overline{s} \uparrow$ holds due to $\overline{s}[i] \uparrow$. A more general proof can be constructed. Since $\overline{s}[i]$ remains after transformation, it must relate to a glue-state $\overline{s}^*[i] \in \mathcal{S}_{\Xi^{r_i}_{\mathcal{L}}/\{i\}} \cap \mathcal{S}_{\Xi^{r_i}_{\mathcal{R}}/\{i\}}$. Since $\Xi^{r_i}_{\mathcal{L}}/\{i\} \leftrightarrows^{\Delta}_b \Xi^{r_i}_{\mathcal{R}}/\{i\}$, we must have that $\overline{s}^*[i] \in \Xi^{r_i}_{\mathcal{L}}/\{i\}$ is divergence-sensitive branching bisimilar to $\overline{s}^*[i] \in \Xi^{r_i}_{\mathcal{R}}/\{i\}$, since it is the only state able to perform a $\kappa_{\overline{s}^*[i]}$-transition. But then, it cannot have changed regarding divergency after transformation, and we have a contradiction.

Note that $\mathcal{N}$ only grows linear with the size of $\mathcal{M}_{y+1}$ in the worst case that new states are introduced, but no divergence. However, many practical transformations introduce divergence (see Section 6), and unlike the set of explored states when doing model checking, $\mathcal{N}$ only needs infrequent updates, and can be maintained on secondary storage.

## 6 Experimental Results

Table 1 shows experimental results for five case studies with various rule systems[8], some preserving a relevant property (noted by ✓) and some not (noted by ✗). The number of explored states and the runtime for full exploration are given for the *initial* model and the *trans*formed model. The applied rule systems have been analysed separately ($\varphi$-*pres.*), and for these checks, the maximum number of states of the two LTSs involved in a check is given in the form "(size left pattern)+(size right pattern)". Furthermore, the number of required checks, and the total runtime are reported. The experiments have been performed on a

[8] The rule system definitions can be found in Appendix B.

**Table 1:** Transformation results for several specifications of concurrent systems. *max #st.* = size of largest check performed, in terms of sizes of compared LTSs. "$n_1 + n_2$" means LTS with $n_1$ states is compared to LTS with $n_2$ states.

| | | **ACS** | **1394-fin** | **wafer** | **brdcst (1)** | **brdcst (2)** | **c.syst. (1)** | **c.syst. (2)** |
|---|---|---|---|---|---|---|---|---|
| **initial** | *#states* | 4,764 | 188,569 | 78,919 | 161,051 | 161,051 | 759,375 | 759,375 |
| | *time* (sec.) | 1.85 | 379.08 | 4.88 | 3.48 | 3.48 | 29.97 | 29.97 |
| **trans.** | *#states* | 21,936 | 5,849,124 | 375,937 | 4,084,101 | 28,629,151 | 656,356,768 | 656,356,768 |
| | *time* (sec.) | 10.23 | 18,045.13 | 49.33 | 83.53 | 952.85 | 48,795.28 | 45,553.27 |
| **$\varphi$-pres.** | *max. #st.* | $2+3$ | $2+6$ | $2+5$ | $27+30$ | $27+31$ | $15+58$ | $15+58$ |
| | *#checks* | 3 | 3 | 3 | 7 | 7 | 63 | 63 |
| | *result* | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ |
| | *time* (sec.) | 0.01 | 0.01 | 0.01 | 0.616 | 0.792 | 1.90 | 1.90 |

machine with two dual-core AMD OPTERON (tm) processors 885 2.6 GHz, 126 GB RAM, running RED HAT 4.3.2-7. For divergence-sensitive branching bisimilarity checking, we used the *ltsconvert* tool of the MCRL2 toolset [15]. The first three models are part of the distribution of MCRL2. We generated their LTSs with the MCRL2 tools, and manually transformed them to incorporate refined information concerning internal steps. In the other two cases, synchronising behaviour was transformed, and the network LTSs have been constructed from sets of process LTSs using EXP.OPEN [26]; *brdcst* is a system of fifteen processes communicating via broadcast, i.e. three processes at a time synchronise simultaneously. A practical transformation is to break this down into a series of two-party synchronisations, e.g. due to restrictions imposed for the eventual implementation. We defined two rule systems for this, and they could be applied fifty times using a prototype tool developed by us. The first of these rule systems does not preserve properties, whereas it can be shown that the second one does, using divergency information of the input models. The *c.syst.* case is a communication system, where in five different places, communication between two processes is refined to use the ABP protocol, representing an adaptation to the use of lossy channels. The different rule networks for the various checks were produced by our prototype, and we again used divergency information of the input models. We analysed two versions of the rule system, one containing a subtle error (the receiver of messages does not expect messages with the wrong bit). In this case, we expect that it is possible to define a $\varphi$-preserving rule system formalising the desired transformation, i.e. replacing one way of communicating by another should not affect the truth-value of properties not stating anything about the used communication mechanism. Therefore, the negative outcome of a check is a strong indication that transformation of the network LTS results in a violation of the property.

The gain in speed is obvious, as is the gain in memory use, since it is linear to the number of analysed states. In the future, we wish to look at larger, practical case studies to further validate the applicability of our techniques.

## 7 Conclusions and Future Work

We formulated a check to determine whether a terminating confluent system of LTS transformation rules preserves a specific safety, liveness, or fairness property for networks of LTSs in general, as long as the rule system is synchronisation uniform w.r.t. the network. One should keep in mind that the question of whether or not a rule system is $\varphi$-preserving in general is different from the question whether a rule system is $\varphi$-preserving when applied on a particular model. Our proposed technique is not complete when applied to answer the second question; applying a rule system which is not deemed to be $\varphi$-preserving may still result in a new network LTS in which $\varphi$ holds, because this depends on the structure of the original network. In a way, this is a limitation of our technique, but on the other hand, the notion of $\varphi$-preservation in general is a useful criterion for the concept of transformation rules as reusable templates, since a $\varphi$-preserving rule system can safely be applied on any network of LTSs for which it is synchronisation uniform.

Besides the synchronisation rules, the system model is not involved in the check, which entails divergence-sensitive branching bisimilarity checking of networks of transformation rules. Its complexity completely depends on the rule system. The check can be further extended by involving system divergency, at the expense of introducing more bookkeeping, but with the benefit that it can more often identify preservation. It is important to note that the approach supports transformations being performed one after another, leading to systems $\mathcal{M}_{y+1}$, $\mathcal{M}_{y+2}$, etc., without requiring that any system LTS must be constructed; new divergency information can be derived, and only the process LTSs, which grow linearly, need to be maintained. Finally, checking multiple properties simply involves performing all the checks for multiple hiding sets.

*Future work.* All required functionality has been implemented, but it still needs to be integrated into one tool. We plan to consider a more general network of LTSs model involving asynchronous communication, which would allow transforming one type of communication to the other. Another extension would be to support the addition and deletion of processes in a network, and we will consider situations where $\varphi$ is transformed. Clearly, as long as its hiding set remains the same, it is preserved, but perhaps more can be achieved. We plan to further extend the technique, possibly making it complete w.r.t. the question whether a property is preserved when a rule system is applied to a particular model. To achieve this, we will investigate how our technique can be combined with those in related work. Finally, we will consider more expressive notions of transformation, for instance involving *negative application conditions* (NACs) [17]. This will allow more refined definitions of transformation rules. There exists work on bisimilarity in the presence of NACs, e.g. [33]. In our setting, however, NACs can be expected to be in conflict with the notion of uniform transformation of synchronising behaviour, i.e. synchronisation uniformity, since they are used to define exceptions to a transformation rule.

## References


1. S. Beydeda, M. Book, and V. Gruhn, editors. *Model-Driven Software Development*. Springer, 2005.
2. P. Böhm. A Framework for Incremental Modelling and Verification of On-Chip Protocols. In *Proc. FMCAD'10*, pages 159–166. IEEE, 2010.
3. C. Braunstein and E. Encrenaz. CTL-Property Transformations Along an Incremental Design Process. In *Proc. AVOCS'04*, volume 128 of *ENTCS*, pages 263–178. Elsevier, 2004.
4. S. Burmester, H. Giese, M. Hirsch, and D. Schilling. Incremental Design and Formal Verification with UML/RT in the FUJABA Real-Time Tool Suite. In *Proc. SVERTS'04*, pages 1–20, 2004.
5. H. Chockler, A. Ivrii, A. Matsliah, S. Moran, and Z. Nevo. Incremental Formal Verification of Hardware. In *Proc. FMCAD'11*, pages 135–143. IEEE, 2011.
6. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 1999.
7. E.M. Clarke, D.E. Long, and K.L. McMillan. Compositional Model Checking. In *Proc. LICS'89*, pages 353–362. IEEE, 1989.
8. G. Cohen and O. Kupferman. Incremental LTL Model Checking. in Proc. Workshop on Sem. and Verif. of Hardware and Software Syst., 2003.
9. C.L. Conway, K.S. Namjoshi, D. Dams, and S.A. Edwards. Incremental Algorithms for Inter-procedural Analysis of Safety Properties. In *Proc. CAV'05*, volume 3576 of *LNCS*, pages 449–461. Springer, 2005.
10. P. Crouzen and F. Lang. Smart Reduction. In *Proc. FASE'11*, volume 6603 of *LNCS*, pages 111–126. Springer, 2011.
11. D. Eppstein, Z. Galil, and G. Italiano. Dynamic Graph Algorithms. In *CRC Handbook of Algorithms and Theory of Computation*, chapter 22. CRC Press, 1997.
12. H. Giese, S. Glesner, J. Leitner, W. Schäfer, and R. Wagner. Towards Verified Model Transformations. In *Proc. MoDeVa'06*, pages 78–93, 2006.
13. R.J. van Glabbeek, B. Luttik, and N. Trčka. Branching Bisimilarity with Explicit Divergence. *Fundamenta Informaticae*, 93(4):371–392, 2009.
14. R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.
15. J.F. Groote, J. Keiren, A. Mathijssen, B. Ploeger, F. Stappers, C. Tankink, Y. Usenko, M. van Weerdenburg, W. Wesselink, T. Willemse, and J. van der Wulp. The mCRL2 Toolset. In *Proc. WASDeTT'08*, 2008.
16. J.F. Groote and F.W. Vaandrager. An Efficient Algorithm for Branching Bisimulation and Stuttering Equivalence. In *Proc. ICALP'90*, volume 443 of *LNCS*, pages 626–638. Springer, 1990.
17. A. Habel, R. Heckel, and G. Taentzer. Graph Grammars with Negative Application Conditions. *Fundamenta Informaticae*, 26(3-4):287–313, 1996.
18. A. Habel, J. Müller, and D. Plump. Double-Pushout Graph Transformation Revisited. *Mathematical Structures in Computer Science*, 11(5):637–688, 2001.
19. R. Heckel. Graph Transformation in a Nutshell. In *Proc. FoVMT'04*, volume 148 of *ENTCS*, pages 187–198. Elsevier, 2006.
20. M. Hülsbusch, B. König, A. Rensink, M. Semenyak, C. Soltenborn, and H. Wehrheim. Showing Full Semantics Preservation in Model Transformation - A Comparison of Techniques. In *Proc. IFM'10*, volume 6396 of *LNCS*, pages 183–198. Springer, 2010.
21. D. Kähler and T. Wilke. Program Complexity of Dynamic LTL Model Checking. In *Proc. CSL'03*, volume 2803 of *LNCS*, pages 271–284. Springer, 2003.

22. G. Karsai and A. Narayanan. On the Correctness of Model Transformations in the Development of Embedded Systems. In *Proc. 13th Monterey Workshop*, volume 4888 of *LNCS*, pages 1–18. Springer, 2007.
23. D. Kozen. Results on the Propositional $\mu$-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
24. S. Krishnamurti and K. Fisler. Foundations of Incremental Aspect Model-Checking. *ACM Trans. on Softw. Eng. and Meth.*, 16(2), 2007.
25. L. Lambers, H. Ehrig, and F. Orejas. Efficient Detection of Conflicts in Graph-based Model Transformation. In *Proc. GraMoT'05*, volume 152 of *ENTCS*, pages 97–109. Elsevier, 2006.
26. F. Lang. Exp.Open 2.0: A Flexible Tool Integrating Partial Order, Compositional, and On-the-Fly Verification Methods. In *Proc. IFM'05*, volume 3771 of *LNCS*, pages 70–88. Springer, 2005.
27. K. Lano. *The B Language and Method, A Guide to Practical Formal Development*. Springer, 1996.
28. R. Mateescu and A.J. Wijs. Property-Dependent Reductions for the Modal Mu-Calculus. In *Proc. SPIN'11*, volume 6823 of *LNCS*, pages 2–19. Springer, 2011.
29. R. De Nicola and F.W. Vaandrager. Action versus State Based Logics for Transition Systems. In *Semantics of Systems of Concurrent Processes, LITP Spring School on Theoretical Computer Science*, volume 469 of *LNCS*, pages 407–419. Springer, 1990.
30. B. Ploeger. Analysis of ACS using mCRL2. CS-Report 09-11, Eindhoven University of Technology, 2009.
31. D. Plump. Confluence of Graph Transformation Revisited. In *Processes, Terms and Cycles: Steps on the Road to Infinity: Essays Dedicated to Jan Willem Klop on the Occasion of His 60th Birthday*, volume 3838 of *LNCS*, pages 280–308. Springer, 2005.
32. G. Ramalingam and T. Reps. On The Computational Complexity of Dynamic Graph Problems. *Theoretical Computer Science*, 158:233–277, 1996.
33. G. Rangel, B. König, and H. Ehrig. Deriving Bisimulation Congruences in the Presence of Negative Application Conditions. In *Proc. FOSSACS'08*, volume 4962 of *LNCS*, pages 413–427. Springer, 2008.
34. W. Ruanthong and P. Muenchaisri. Model Checking for Aspect-Oriented Software Evolution. *WSEAS Trans. on Computers*, 4(2):216–221, 2005.
35. D. Saha. An Incremental Bisimulation Algorithm. In *Proc. FSTTCS'07*, volume 4855 of *LNCS*, pages 204–215. Springer, 2007.
36. O.V. Sokolsky and S.A. Smolka. Incremental Model Checking in the Modal Mu-Calculus. In *Proc. CAV'94*, volume 818 of *LNCS*, pages 351–363. Springer, 1994.
37. G.M. Swamy. *Incremental Methods for Formal Verification and Logic Synthesis*. PhD thesis, University of California, 1996.

## A Correctness Proof of the $\varphi$-Preservation Check

With $\mathcal{H}_y$ ($\mathcal{H}_{y+1}$), we refer to $\tilde{\tau}_\varphi(\mathcal{M}_y)$ ($\tilde{\tau}_\varphi(\mathcal{M}_{y+1})$).

**Proposition 3.** *Let $\varphi \in L_\mu^{dsbr}$ be a temporal property with $\models_{\mathcal{M}_y} \varphi$. Then $\Sigma_y$ is* $\varphi$-preserving *if for all $r_i \in R_y$, $I \subseteq dep(\mathcal{L}^{r_i})$:*

$$\tilde{\tau}_\varphi(\Xi_{\mathcal{L}}^{r_i}/I) \leftrightarrows_b^\Delta \tilde{\tau}_\varphi(\Xi_{\mathcal{R}}^{r_i}/I) \tag{1}$$

*Proof.* By def., $\Sigma_y$ is $\varphi$-preserving iff there exists a divergence-sensitive branching bisimulation $C$ showing that $\mathcal{H}_y \leftrightarrows_b^\Delta \mathcal{H}_{y+1}$, i.e. $\forall s_{I,y} \in \mathcal{S}_y, s_{I,y+1} \in \mathcal{S}_{y+1}.s_{I,y}\ C\ s_{I,y+1}$. We will show that such a bisimulation can be constructed, given that (1) holds. By (1), there are for each $r_i \in R_y$, $2^{|dep(\mathcal{L}^{r_i})|} - 1$ bisimulations relating the $\tilde{\tau}_\varphi(\Xi_{\mathcal{L}}^{r_i}/I)$ and $\tilde{\tau}_\varphi(\Xi_{\mathcal{R}}^{r_i}/I)$.[9] With this, we construct $C$ between $\mathcal{H}_y$ and $\mathcal{H}_{y+1}$ as follows: $C = \{(\overline{s},\overline{p}) \mid \forall i \in 1..n.(i \notin \hat{M}(\overline{s}) \Rightarrow \overline{s}[i] = \overline{p}[i]) \wedge (i \in M(\overline{s}) \Rightarrow \forall I \subseteq dep(\mathcal{L}^{r_i}).\exists \overline{s}^* \in \mathcal{S}_{\tilde{\tau}_\varphi(\Xi_{\mathcal{L}}^{r_i}/I)}, \overline{p}^* \in \mathcal{S}_{\tilde{\tau}_\varphi(\Xi_{\mathcal{R}}^{r_i}/I)}.\overline{s}^* \vdash_{\mathcal{L}}^{r_i,I} \overline{s} \wedge \overline{p}^* \vdash_{\mathcal{R}}^{r_i,I} \overline{p} \wedge \overline{s}^* \leftrightarrows_b^\Delta \overline{p}^*)\}$. We show that $C$ is a divergence-sensitive branching bisimulation by proving that for all $(\overline{s},\overline{p}) \in C$, Def. 3 holds. We distinguish four cases:

1. $\overline{s} \xrightarrow{a}_{\mathcal{H}_y} \overline{s}'$. This is enabled by some $(\bar{t}, a) \in \mathcal{V}_y$. We distinguish two cases:
   (a) $\hat{M}(\overline{s}) \cap Ac(\bar{t}) \neq \emptyset$. We have $\hat{M}(\overline{s}) \subseteq M(\overline{s}')$ (Lemma 3) and $\hat{M}(\overline{s}) \subseteq M(\overline{s})$, so $M(\overline{s}) \cap M(\overline{s}') \cap Ac(\bar{t}) \neq \emptyset$. Hence, by Lemma 4, $Ac(\bar{t}) \subseteq dep(\mathcal{L}^{r_i})$, and by Lemma 5, $Ac(\bar{t}) \subseteq M(\overline{s})$. Let $i \in M(\overline{s}) \cap M(\overline{s}') \cap Ac(\bar{t})$. By def. of $C$, since $i \in M(\overline{s})$ and $Ac(\bar{t}) \subseteq dep(\mathcal{L}^{r_i})$, we have an $\overline{s}^* \in \mathcal{S}_{\tilde{\tau}_\varphi(\Xi_{\mathcal{L}}^{r_i}/Ac(\bar{t}))}$ with $\overline{s}^* \vdash_{\mathcal{L}}^{r_i,Ac(\bar{t})} \overline{s}$, and by Lemma 6, $\overline{s}^* \xrightarrow{a}_{\tilde{\tau}_\varphi(\Xi_{\mathcal{L}}^{r_i}/Ac(\bar{t}))} \overline{s}^{*\prime}$ with $\overline{s}^{*\prime} \vdash_{\mathcal{L}}^{r_i,Ac(\bar{t})} \overline{s}'$. By def. of $C$, since $i \in M(\overline{s})$, we have a $\overline{p}^* \in \mathcal{S}_{\tilde{\tau}_\varphi(\Xi_{\mathcal{R}}^{r_i}/Ac(\bar{t}))}$ with $\overline{p}^* \vdash_{\mathcal{R}}^{r_i,Ac(\bar{t})} \overline{p}$ and $\overline{s}^* \leftrightarrows_b^\Delta \overline{p}^*$. So in $\tilde{\tau}_\varphi(\Xi_{\mathcal{R}}^{r_i}/Ac(\bar{t}))$
      – either $a = \tau$ and $\overline{s}^{*\prime} \leftrightarrows_b^\Delta \overline{p}^*$. Since for all $j \notin Ac(\bar{t})$, $\overline{s}'[j] = \overline{s}[j]$, we have $\overline{s}'\ C\ \overline{p}$;
      – or we have $\overline{p}^{*\prime}, \overline{p}^{*\prime\prime}$ such that $\overline{p}^* \Rightarrow_{\tilde{\tau}_\varphi(\Xi_{\mathcal{R}}^{r_i}/Ac(\bar{t}))} \overline{p}^{*\prime} \xrightarrow{a}_{\tilde{\tau}_\varphi(\Xi_{\mathcal{R}}^{r_i}/Ac(\bar{t}))} \overline{p}^{*\prime\prime}$, with $\overline{p}^* \leftrightarrows_b^\Delta \overline{p}^{*\prime}$ and $\overline{s}^{*\prime} \leftrightarrows_b^\Delta \overline{p}^{*\prime\prime}$. Repeated application of Lemma 6 shows that $\overline{p} \Rightarrow_{\mathcal{H}_{y+1}} \overline{p}' \xrightarrow{a}_{\mathcal{H}_{y+1}} \overline{p}''$ with $\overline{p}^{*\prime} \vdash_{\mathcal{R}}^{r_i,Ac(\bar{t})} \overline{p}'$ and $\overline{p}^{*\prime\prime} \vdash_{\mathcal{R}}^{r_i,Ac(\bar{t})} \overline{p}''$. Since for all $j \notin Ac(\bar{t})$, $\overline{s}'[j] = \overline{s}[j]$ and $\overline{p}''[j] = \overline{p}'[j] = \overline{p}[j]$, we have $\overline{s}\ C\ \overline{p}'$ and $\overline{s}'\ C\ \overline{p}''$.
   (b) $\hat{M}(\overline{s}) \cap Ac(\bar{t}) = \emptyset$. We distinguish two cases:
      i. $\hat{M}(\overline{s}') \cap Ac(\bar{t}) = \emptyset$. We will first show that $\overline{p} \Rightarrow_{\mathcal{H}_{y+1}} \overline{p}' \xrightarrow{a}_{\mathcal{H}_{y+1}} \overline{p}''$ by showing that for each $j \in Ac(\bar{t})$, $\overline{p}[j] \Rightarrow_j \overline{p}'[j]$ and $\overline{p}'[j] = \overline{s}[j]$. By def. of $C$, $\forall j \in Ac(\bar{t}) \setminus M(\overline{s}).\overline{p}[j] = \overline{s}[j]$. For each $j \in Ac(\bar{t}) \cap M(\overline{s})$, since $\hat{M}(\overline{s}) \cap Ac(\bar{t}) = \emptyset$, we have $j \in M(\overline{s}) \setminus \hat{M}(\overline{s})$ and since $\{j\} \subseteq dep(\mathcal{L}^{r_j})$, by the isomorphisms, also an (initial) $\overline{s}^* \in \mathcal{S}_{\tilde{\tau}_\varphi(\Xi_{\mathcal{L}}^{r_j}/\{j\})} \cap \mathcal{S}_{\tilde{\tau}_\varphi(\Xi_{\mathcal{R}}^{r_j}/\{j\})}$

[9] There are at most $2^{|dep(\mathcal{L}^{r_i})|}$ different combinations of processes able to participate in synchronisations included in $\mathcal{L}^{r_i}$; the case where no process is able to do so is trivial. Note that for each $r_i$ and all $r_j$ with $j \in dep(\mathcal{L}^{r_i})$, the bisimulations are identical, so the number of distinct bisimulations is much lower.

with $\overline{s}^* \overset{\kappa_{\overline{s}^*[j]}}{\rightarrow} \overline{s}^*$. By def. of $C$, there is a $\overline{p}^* \in \mathcal{S}_{\tilde{\tau}_\varphi(\Xi_{\mathcal{R}}^{r_j}/\{j\})}$ such that $\overline{s}^* \vdash_{\mathcal{L}}^{r_j,\{j\}} \overline{s}$, $\overline{p}^* \vdash_{\mathcal{R}}^{r_j,\{j\}} \overline{p}$, and $\overline{s}^* \leftrightarrows_b^{\Delta} \overline{p}^*$. By Def. 3, since $\kappa_{\overline{s}^*[j]} \neq \tau$, there must be $\overline{p}^{*\prime}$, $\overline{p}^{*\prime\prime}$ such that $\overline{p}^* \Rightarrow_{\tilde{\tau}_\varphi(\Xi_{\mathcal{R}}^{r_j}/\{j\})} \overline{p}^{*\prime} \overset{\kappa_{\overline{s}^*[j]}}{\rightarrow}_{\tilde{\tau}_\varphi(\Xi_{\mathcal{R}}^{r_j}/\{j\})} \overline{p}^{*\prime\prime}$. But since only $\overline{s}^*[j]$ has an outgoing $\kappa_{\overline{s}^*[j]}$-transition in $\mathcal{R}^{r_j}$, we must have $\overline{p}^{*\prime}[j] = \overline{s}^*[j]$. Since $\overline{p}^* \vdash_{\mathcal{R}}^{r_j,\{j\}} \overline{p}$, rep. appl. of Lemma 6 shows that $\overline{p} \Rightarrow_{\mathcal{H}_{y+1}} \overline{p}'$ with $\overline{s}^* \vdash_{\mathcal{R}}^{r_j,\{j\}} \overline{p}'$, hence $\overline{p}'[j] = \overline{s}[j]$. So for each $j \in Ac(\overline{t})$, $\overline{p}[j] \Rightarrow_j \overline{p}'[j]$ and $\overline{p}'[j] = \overline{s}[j]$. Since $\hat{M}(\overline{s}) \cap Ac(\overline{t}) = \emptyset$ and $\forall j \notin Ac(\overline{t}).\overline{p}'[j] = \overline{p}[j]$, we have $\overline{s}\ C\ \overline{p}'$. Since $\hat{M}(\overline{s}') \cap Ac(\overline{t}) = \emptyset$, we have $\forall j \in Ac(\overline{t}).\overline{p}''[j] = \overline{s}'[j]$. Since $\forall j \notin Ac(\overline{t}).\overline{s}'[j] = \overline{s}[j] \wedge \overline{p}''[j] = \overline{p}'[j]$, we have $\overline{s}'\ C\ \overline{p}''$.

   ii. $\hat{M}(\overline{s}') \cap Ac(\overline{t}) \neq \emptyset$. We have $\hat{M}(\overline{s}') \subseteq M(\overline{s})$ (Lemma 3) and $\hat{M}(\overline{s}') \subseteq M(\overline{s}')$, so $M(\overline{s}) \cap M(\overline{s}') \cap Ac(\overline{t}) \neq \emptyset$. By Lemma 5, ... (continued as in 1.(a)).

2. $\overline{p} \overset{a}{\rightarrow}_{\mathcal{H}_{y+1}} \overline{p}'$. Similar to the previous case.

3. $\overline{s}\uparrow$. We call the infinite $\tau$-path $\sigma$. Let $\overline{s}_i \overset{\tau}{\rightarrow}^{\overline{t}_i}_{\mathcal{H}_y} \overline{s}_{i+1}$ be the first transition in $\sigma$ for which $M(\overline{s}_i) \cap M(\overline{s}_{i+1}) \cap Ac(\overline{t}_i) \neq \emptyset$. If it does not exist, then clearly, $\overline{p}\uparrow$. If it does, then this action can also be performed in the appropriate $\tilde{\tau}_\varphi(\Xi_{\mathcal{L}}^{r_j}/Ac(\overline{t}_i))$, by case 1 simulated in $\tilde{\tau}_\varphi(\Xi_{\mathcal{R}}^{r_j}/Ac(\overline{t}_i))$, and by the isomorphisms, also simulated in $\mathcal{H}_{y+1}$. If there is a finite number of such transitions, there is still an infinite number of reachable $\tau$-transitions in $\mathcal{H}_{y+1}$, hence $\overline{p}\uparrow$. If there is an infinite number of such transitions, then since there is a finite number of networks $\Xi_{\mathcal{L}}^{r_j}/I$ ($R_y$ is finite and $1..n$ is finite), one of these must contain an infinite $\tau$-path that can be matched on a part of $\sigma$. But then, there is a diverging state $\overline{s}_i^*$ in this network, which is $\leftrightarrows_b^{\Delta}$ to a $\overline{p}_i^* \in \mathcal{S}_{\tilde{\tau}_\varphi(\Xi_{\mathcal{R}}^{r_j}/Ac(\overline{t}_i))}$, which can be matched on a $\overline{p}_i \in \mathcal{S}_{y+1}$. Due to the isomorphisms, $\overline{p}_i\uparrow$. Since $\overline{p} \Rightarrow_{\mathcal{H}_{y+1}} \overline{p}_i$ (all $\tau$-transitions up to $\overline{s}_i$ have been simulated), $\overline{p}\uparrow$.

4. $\overline{p}\uparrow$. Similar to the previous case. □

## B Specifics of the Performed Experiments

In this section, we discuss the experiments of Sect. 6 in more detail. We describe the input models used for these experiments and show how they are transformed.

### B.1 Broadcast

Broadcast is a system of fifteen processes communicating via three-party broadcast, i.e. three processes at a time synchronise simultaneously. Fig. 9 shows two pairs of three such processes. For each group of three processes, there is a synchronisation rule that states that actions *a1*, *a2* and *a3* synchronise.

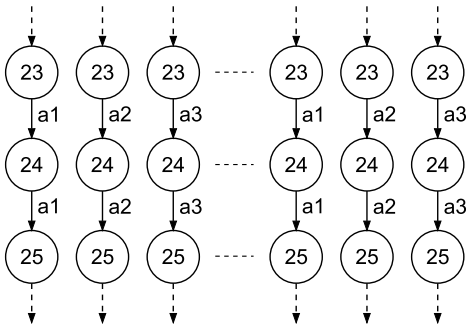



**Fig. 9:** Groups of three processes that communicate via broadcast

A practical transformation is to break this down into a series of two-party synchronisation, e.g. due to restrictions imposed for the eventual implementation. Three transformation rules that refine a model in this way are shown in Fig. 10. After transformation, new synchronisation rules are introduced that define that $a1'$ and $a2'$, and $a2''$ and $a3'$ synchronise. This naive refinement does not preserve properties.

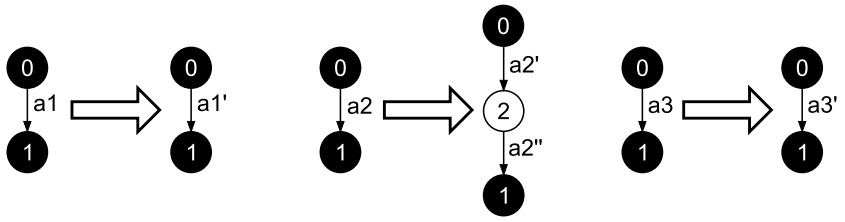



**Fig. 10:** Three transformation rules that replace a three-party broadcast by pairwise communication

Improved versions of these transformation rules are shown in Fig. 11. After transformation, new synchronisation rules are introduced that define that

- *m1a1* and *m2a1*,
- *c1a1* and *c2a1*,
- *a1a1* and *a2a1*, and
- *a2′* and *a3′*

synchronise.

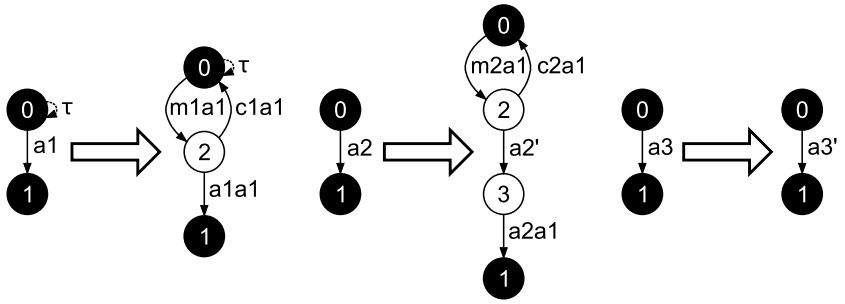



**Fig. 11:** Three improved transformation rules that replace a three-party broadcast by pairwise communication

Note in the rules in Fig. 11 that *a2* and *a3* are replaced by *a2′* and *a3′*, respectively, after transformation. This is done to make the rule system terminating and confluent; otherwise, each rule would be applicable again on its own output.

To check whether the transformation rules of Fig. 11 preserve properties, a number of checks [10] have to be performed. Creating and performing these requires the tool LTSCOMPARE from the MCRL2 toolkit, and EXP.OPEN and BCG_IO from the CADP toolkit. You can perform the checks as follows:

- Install the MCRL2 toolkit.
- Install the CADP toolkit.
- Download the EXP.OPEN definitions of the checks.
- Convert the modified left-hand sides and right-hand sides of the transformation rules in Aldebaran format to Binary Coded Graphs by running `createbcg`.
- Create networks by running `createnetworks`.
- Convert the networks to the Aldebaran format by running `createaut`.
- Check whether the networks for the left-hand sides are divergence-sensitive branching bisimilar to the networks for the right-hand sides by running `check`.

[10] An archive containing all files necessary to perform these checks can be found at `http://www.win.tue.nl/~awijs/incmc`.

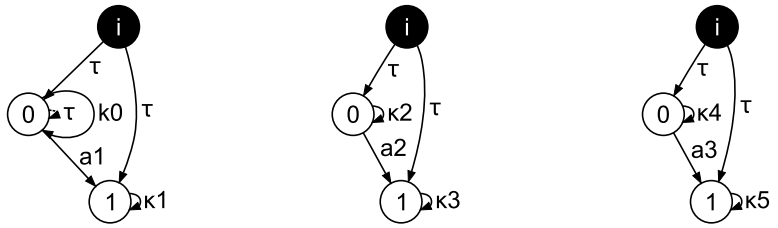



**Fig. 12:** Process LTSs of the left-hand sides of the transformation rules

Figs. 12 and 13 show the process LTSs that are used for the checks. These LTSs are created from the left-hand and right-hand sides of the three transformation rules. The tools EXP.OPEN and LTSCOMPARE of the MCRL2 toolkit cannot handle process LTSs with multiple initial states. For this reason, one initial state is added to each of the LTSs, as well as $\tau$-transitions to the original initial states. The figures also show the $\kappa$-loops that are added to the original initial states. Each of the checks determines whether a network consisting of a combination of process LTSs created from the left-hand sides of transformation rules is divergence-sensitive branching bisimilar with the network consisting of the corresponding process LTSs created from the right-hand sides, after hiding the appropriate actions in both networks.

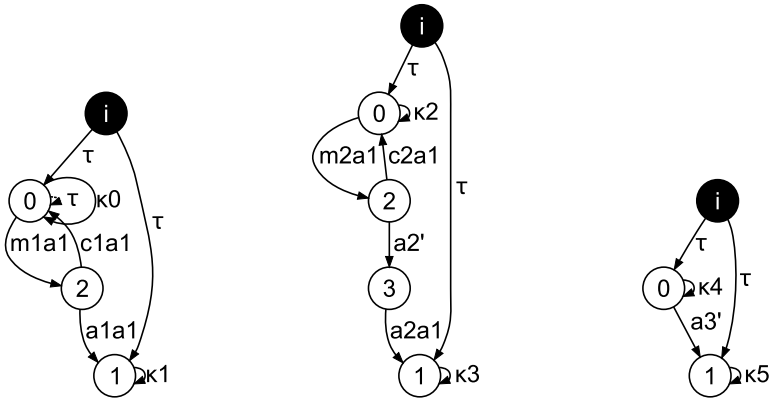



**Fig. 13:** Process LTSs of the right-hand sides of the transformation rules

### B.2 Alternating Bit Protocol

Fig. 14 shows two components, $P$ and $Q$, that communicate via four buffers, *B1*, *B2*, *B3*, and *B4*. For the experiment described in this report, we analyzed a model containing five instances of such a communicating system operating concurrently.

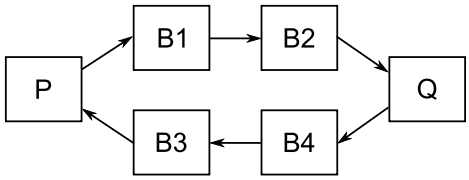



**Fig. 14:** Two components ($P$ and $Q$) that communicate via four buffers ($B1$, $B2$, $B3$ and $B4$)

Figs. 15 to 20 show the process LTSs representing the six components. Process $P$ either performs an action $pa$ or an action $qa$ and then communicates with component $Q$. After receiving either an $a$ or a $b$ from $P$, process $Q$ performs an action $qa$ or an action $qb$. Afterwards, $Q$ acknowledges the message reception.

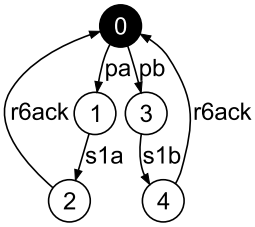



**Fig. 15:** Process P

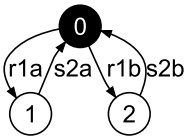



**Fig. 16:** Process B1

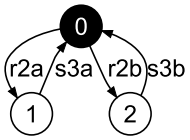



**Fig. 17:** Process B2

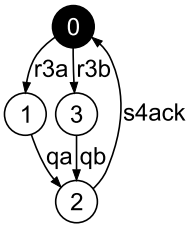



**Fig. 18:** Process Q

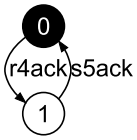



**Fig. 19:** Process B3

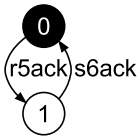



**Fig. 20:** Process B4

Fig. 21 shows two transformation rules that replace two of the buffers by two processes that implement the alternating bit protocol. The alternating bit is encoded by the added suffixes "t" (TRUE) and "f" (FALSE) in the transition labels. Applying this transformation to the five concurrent communicating systems (and adding appropriate synchronisation rules in the obtained network of LTSs) leads to an explosion of the state-space and thus to a large exploration time. Checking whether these transformation rules preserve properties, however,

takes very little time. Note that there are also transformation rules for the other processes only doing some simple renaming, e.g. $pa$ to $pa'$, to make the rule system synchronisation uniform w.r.t. the synchronisation rules of the system. The actual check then involves checking whether $2^6 - 1 = 63$ combinations of active processes before and after transformation produce divergence-sensitive branching bisimilar LTSs (after maximal hiding). However, the generation of each rule network LTS only took at most 0.01s, and the check involving all processes, i.e. involving the largest LTSs also only took 0.01s, leading an overall runtime of approximately $(0.01\text{s} * 126 \text{ networks}) + (0.01\text{s} * 63 \text{ checks}) = 1.89\text{s}$, which is very close to the measured 1.90s.

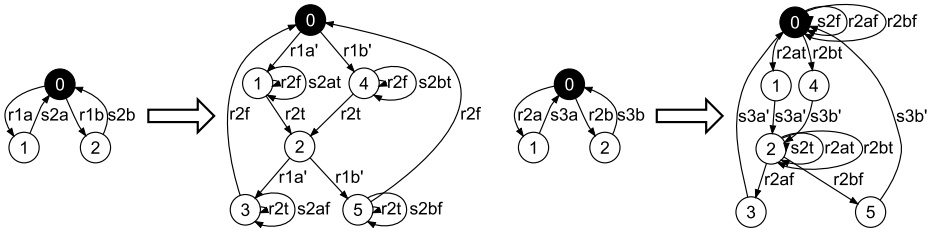



**Fig. 21:** Two transformation rules that replace buffers

### B.3 ACS (comparable with 1394-fin and Wafer Stepper)

The ACS Manager along with Containers and Components is a part of the software of the ALMA project carried out by the European Southern Observatory (ESO) [30]. The intention of this project is to put more than 60 radio telescopes on a plane high up in the mountains of Chili for radio astronomy. A specification is part of the official distribution of the MCRL2 toolset. Fig. 22 describes a transformation of the receive action ($rcv$) into a more detailed procedure involving decompression of the received message. This rule was applied on the two components and one container present in the specification. The other party in the two-way synchronisation (the *send* action), was basically left unchanged (rewritten to a $send'$ action to ensure that the rule system is terminating, confluent, and synchronisation uniform).

Checking for preservation of a property when two-party synchronising behaviour is transformed requires performing three checks, representing the scenarios of successful synchronisation and unsuccessful synchronisation where either one of the parties unsuccessfully attempts to synchronise. All checks involve determining whether the corresponding networks of left- and right-hand sides of the relevant rules are divergence-sensitive branching bisimilar after maximal hiding, which can be done very fast. In the networks of right-hand side patterns, a synchronisation rule is added stating that *decompress* can be fired by itself.

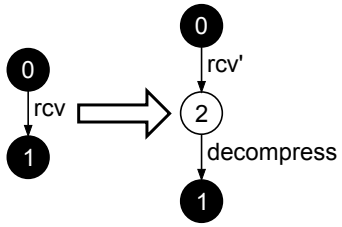



**Fig. 22:** A transformation rule refining the processing of received messages

The 1394-fin (Firewire) case and the Wafer Stepper case are two other mCRL2 specifications which have been transformed using rules very similar to this rule (but involving different numbers of transitions).

**Science Reports** **Department of Mathematics and Computer Science**
**Technische Universiteit Eindhoven**

If you want to receive reports, send an email to: wsinsan@tue.nl (we cannot guarantee the availability of the requested reports).

## *In this series appeared (from 2009):*

| | | |
|---|---|---|
| 09/01 | Wil M.P. van der Aalst, Kees M. van Hee, Peter Massuthe, Natalia Sidorova and Jan Martijn van der Werf | Compositional Service Trees |
| 09/02 | P.J.l. Cuijpers, F.A.J. Koenders, M.G.P. Pustjens, B.A.G. Senders, P.J.A. van Tilburg, P. Verduin | Queue merge: a Binary Operator for Modeling Queueing Behavior |
| 09/03 | Maarten G. Meulen, Frank P.M. Stappers and Tim A.C. Willemse | Breadth-Bounded Model Checking |
| 09/04 | Muhammad Atif and MohammadReza Mousavi | Formal Specification and Analysis of Accelerated Heartbeat Protocols |
| 09/05 | Michael Franssen | Placeholder Calculus for First-Order logic |
| 09/06 | Daniel Trivellato, Fred Spiessens, Nicola Zannone and Sandro Etalle | POLIPO: Policies & OntoLogies for the Interoperability, Portability, and autOnomy |
| 09/07 | Marco Zapletal, Wil M.P. van der Aalst, Nick Russell, Philipp Liegl and Hannes Werthner | Pattern-based Analysis of Windows Workflow |
| 09/08 | Mike Holenderski, Reinder J. Bril and Johan J. Lukkien | Swift mode changes in memory constrained real-time systems |
| 09/09 | Dragan Bošnački, Aad Mathijssen and Yaroslav S. Usenko | Behavioural analysis of an I²C Linux Driver |
| 09/10 | Ugur Keskin | In-Vehicle Communication Networks: A Literature Survey |
| 09/11 | Bas Ploeger | Analysis of ACS using mCRL2 |
| 09/12 | Wolfgang Boehmer, Christoph Brandt and Jan Friso Groote | Evaluation of a Business Continuity Plan using Process Algebra and Modal Logic |
| 09/13 | Luca Aceto, Anna Ingolfsdottir, MohammadReza Mousavi and Michel A. Reniers | A Rule Format for Unit Elements |
| 09/14 | Maja Pešić, Dragan Bošnački and Wil M.P. van der Aalst | Enacting Declarative Languages using LTL: Avoiding Errors and Improving Performance |
| 09/15 | MohammadReza Mousavi and Emil Sekerinski, Editors | Proceedings of Formal Methods 2009 Doctoral Symposium |
| 09/16 | Muhammad Atif | Formal Analysis of Consensus Protocols in Asynchronous Distributed Systems |
| 09/17 | Jeroen Keiren and Tim A.C. Willemse | Bisimulation Minimisations for Boolean Equation Systems |
| 09/18 | Kees van Hee, Jan Hidders, Geert-Jan Houben, Jan Paredaens, Philippe Thiran | On-the-fly Auditing of Business Processes |
| 10/01 | Ammar Osaiweran, Marcel Boosten, MohammadReza Mousavi | Analytical Software Design: Introduction and Industrial Experience Report |
| 10/02 | F.E.J. Kruseman Aretz | Design and correctness proof of an emulation of the floating-point operations of the Electrologica X8. A case study |

| | | |
|---|---|---|
| 10/03 | Luca Aceto, Matteo Cimini, Anna Ingolfsdottir, MohammadReza Mousavi and Michel A. Reniers | On Rule Formats for Zero and Unit Elements |
| 10/04 | Hamid Reza Asaadi, Ramtin Khosravi, MohammadReza Mousavi, Neda Noroozi | Towards Model-Based Testing of Electronic Funds Transfer Systems |
| 10/05 | Reinder J. Bril, Uğur Keskin, Moris Behnam, Thomas Nolte | Schedulability analysis of synchronization protocols based on overrun without payback for hierarchical scheduling frameworks revisited |
| 10/06 | Zvezdan Protić | Locally unique labeling of model elements for state-based model differences |
| 10/07 | C.G.U. Okwudire and R.J. Bril | Converting existing analysis to the EDP resource model |
| 10/08 | Muhammed Atif, Sjoerd Cranen, MohammadReza Mousavi | Reconstruction and verification of group membership protocols |
| 10/09 | Sjoerd Cranen, Jan Friso Groote, Michel Reniers | A linear translation from LTL to the first-order modal μ-calculus |
| 10/10 | Mike Holenderski, Wim Cools Reinder J. Bril, Johan J. Lukkien | Extending an Open-source Real-time Operating System with Hierarchical Scheduling |
| 10/11 | Eric van Wyk and Steffen Zschaler | 1st Doctoral Symposium of the International Conference on Software Language Engineering (SLE) |
| 10/12 | Pre-Proceedings | 3rd International Software Language Engineering Conference |
| 10/13 | Faisal Kamiran, Toon Calders and Mykola Pechenizkiy | Discrimination Aware Decision Tree Learning |
| 10/14 | J.F. Groote, T.W.D.M. Kouters and A.A.H. Osaiweran | Specification Guidelines to avoid the State Space Explosion Problem |
| 10/15 | Daniel Trivellato, Nicola Zannone and Sandro Etalle | GEM: a Distributed Goal Evaluation Algorithm for Trust Management |
| 10/16 | L. Aceto, M. Cimini, A.Ingolfsdottir, M.R. Mousavi and M. A. Reniers | Rule Formats for Distributivity |
| 10/17 | L. Aceto, A. Birgisson, A. Ingolfsdottir, and M.R. Mousavi | Decompositional Reasoning about the History of Parallel Processes |
| 10/18 | P.D. Mosses, M.R. Mousavi and M.A. Reniers | Robustness os Behavioral Equivalence on Open Terms |
| 10/19 | Harsh Beohar and Pieter Cuijpers | Desynchronisability of (partial) closed loop systems |
| 11/01 | Kees M. van Hee, Natalia Sidorova and Jan Martijn van der Werf | Refinement of Synchronizable Places with Multi-workflow Nets - Weak termination preserved! |
| 11/02 | M.F. van Amstel, M.G.J. van den Brand and L.J.P. Engelen | Using a DSL and Fine-grained Model Transformations to Explore the boundaries of Model Verification |
| 11/03 | H.R. Mahrooghi and M.R. Mousavi | Reconciling Operational and Epistemic Approaches to the Formal Analysis of Crypto-Based Security Protocols |
| 11/04 | J.F. Groote, A.A.H. Osaiweran and J.H. Wesselius | Benefits of Applying Formal Methods to Industrial Control Software |
| 11/05 | Jan Friso Groote and Jan Lanik | Semantics, bisimulation and congruence results for a general stochastic process operator |
| 11/06 | P.J.L. Cuijpers | Moore-Smith theory for Uniform Spaces through Asymptotic Equivalence |
| 11/07 | F.P.M. Stappers, M.A. Reniers and S. Weber | Transforming SOS Specifications to Linear Processes |
| 11/08 | Debjyoti Bera, Kees M. van Hee, Michiel van Osch and Jan Martijn van der Werf | A Component Framework where Port Compatibility Implies Weak Termination |
| 11/09 | Tseesuren Batsuuri, Reinder J. Bril and Johan Lukkien | Model, analysis, and improvements for inter-vehicle communication using one-hop periodic broadcasting based on the 802.11p protocol |

| | | |
|---|---|---|
| 11/10 | Neda Noroozi, Ramtin Khosravi, MohammadReza Mousavi and Tim A.C. Willemse | Synchronizing Asynchronous Conformance Testing |
| 11/11 | Jeroen J.A. Keiren and Michel A. Reniers | Type checking mCRL2 |
| 11/12 | Muhammad Atif, MohammadReza Mousavi and Ammar Osaiweran | Formal Verification of Unreliable Failure Detectors in Partially Synchronous Systems |
| 11/13 | J.F. Groote, A.A.H. Osaiweran and J.H. Wesselius | Experience report on developing the Front-end Client unit under the control of formal methods |
| 11/14 | J.F. Groote, A.A.H. Osaiweran and J.H. Wesselius | Ananlyzing a Controller of a Power Distribution Unit Using Formal Methods |
| 11/15 | John Businge, Alexander Serebrenik and Mark van den Brand | Eclipse API Usage: The Good and The Bad |
| 11/16 | J.F. Groote, A.A.H. Osaiweran, M.T.W. Schuts and J.H. Wesselius | Investigating the Effects of Designing Control Software using Push and Poll Strategies |
| 11/17 | M.F. van Amstel, A. Serebrenik And M.G.J. van den Brand | Visualizing Traceability in Model Transformation Compositions |
| 11/18 | F.P.M. Stappers, M.A. Reniers, J.F. Groote and S. Weber | Dogfooding the Structural Operational Semantics of mCRL2 |
| 12/01 | S. Cranen | Model checking the FlexRay startup phase |
| 12/02 | U. Khadim and P.J.L. Cuijpers | Appendix C / G of the paper: Repairing Time-Determinism in the Process Algebra for Hybrid Systems ACP |
| 12/03 | M.M.H.P. van den Heuvel, P.J.L. Cuijpers, J.J. Lukkien and N.W. Fisher | Revised budget allocations for fixed-priority-scheduled periodic resources |
| 12/04 | Ammar Osaiweran, Tom Fransen, Jan Friso Groote and Bart van Rijnsoever | Experience Report on Designing and Developing Control Components using Formal Methods |
| 12/05 | Sjoerd Cranen, Jeroen J.A. Keiren and Tim A.C. Willemse | A cure for stuttering parity games |
| 12/06 | A.P. van der Meer | CIF MSOS type system |
| 12/07 | Dirk Fahland and Robert Prüfer | Data and Abstraction for Scenario-Based Modeling with Petri Nets |
| 12/08 | Luc Engelen and Anton Wijs | Checking Property Preservation of Refining Transformations for Model-Driven Development |