Science of Computer Programming 196 (2020) 102493

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Compositional model checking with divergence preserving branching bisimilarity is lively

Sander de Putter [a,1], Frédéric Lang [b], Anton Wijs [a,*]

[a] Eindhoven University of Technology, PO BOX 513, 5600 MB, Eindhoven, the Netherlands
[b] Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP [2], LIG, 38000 Grenoble, France

ARTICLE INFO

Article history:
Received 15 March 2018
Received in revised form 24 March 2020
Accepted 21 May 2020
Available online 27 May 2020

Keywords:
Divergence-preserving branching bisimilarity
Congruence
Parallel composition
Synchronisation
Compositional model checking

ABSTRACT

Compositional model checking approaches attempt to limit state space explosion by iteratively combining the behaviour of the components in a concurrent system and reducing the result modulo an appropriate equivalence relation. In this article, we consider *Labelled Transition Systems* (LTSs), in which transitions are labelled by actions, to describe component behaviour, and *LTS networks* to combine the behaviour of all components in a system. For an equivalence relation to be useful for the compositional model checking of LTS networks, it should be a congruence for the parallel composition operator that is used to combine component behaviour. Such an operator may define synchronisations between the actions of component transitions.

An equivalence relation preserving both safety and liveness properties is divergence-preserving branching bisimilarity (DPBB). It has long been generally assumed that DPBB is a congruence for parallel composition. Fokkink, Van Glabbeek and Luttik recently proposed a congruence format that implies that this is the case. In parallel, we were the first to prove, by means of the Coq proof assistant, that DPBB is a congruence for the parallel composition of two LTS networks with synchronisation on transition labels. In the current article, we also consider an instance of our parallel composition operator that is both associative and commutative, which are two essential properties for the compositional construction of state spaces.

Furthermore, we show that DPBB is a congruence for LTS networks in which many LTSs are composed in parallel at once with support for multi-party synchronisation. Additionally, we discuss how to safely decompose an existing LTS network into components such that their recomposition is equivalent to the original LTS network.

Finally, to demonstrate the effectiveness of compositional model checking with intermediate DPBB reductions, we discuss the results we obtained after having conducted a number of experiments.

© 2020 Elsevier B.V. All rights reserved.

* Corresponding author.
E-mail addresses: s.m.j.d.putter@tue.nl (S. de Putter), frederic.lang@inra.fr (F. Lang), a.j.wijs@tue.nl (A. Wijs).

[1] This work is supported by ARTEMIS Joint Undertaking project EMC2 (grant nr. 621429).

[2] Institute of Engineering Univ. Grenoble Alpes.

https://doi.org/10.1016/j.scico.2020.102493
0167-6423/© 2020 Elsevier B.V. All rights reserved.

1. Introduction

Model checking [1,2] is one of the most successful approaches for the analysis and verification of the behaviour of concurrent systems. However, a major issue is the so-called *state space explosion problem*: the state space of a concurrent system tends to increase exponentially as the number of parallel processes increases linearly. Often, it is difficult or infeasible to verify realistic large scale concurrent systems. Over time, several methods have been proposed to tackle the state space explosion problem. Prominent approaches are the application of some form of on-the-fly reduction, such as Partial Order Reduction [3] or Symmetry Reduction [4], and compositionally verifying the system, for instance using Compositional Reasoning [5] or Partial Model Checking [6,7].

The key operations in compositional approaches are the composition and decomposition of systems. First a system is decomposed into two or more components. Then, one or more of these components is manipulated (e.g., reduced). Finally, the components are re-composed. Comparison modulo an appropriate equivalence relation is applied to ensure that the manipulations preserve properties of interest (for instance, expressed in the modal μ-calculus [8]). These manipulations are sound if and only if the equivalence relation 1) is guaranteed to preserve the formalisations of the properties, and 2) is a congruence for the composition expression.

Labelled Transition Systems (LTSs) are often used to formalise the semantics of concurrent systems, and the semantics of the individual components in such a system. Two prominent equivalences for LTSs are branching bisimilarity and divergence-preserving branching bisimilarity (DPBB) [9,10]. Branching bisimilarity preserves safety properties, while DPBB preserves both safety and liveness properties.

In [11], it is proven that DPBB preserves properties expressed in a particular fragment of the modal μ-calculus if the actions referred to in those properties are not abstracted away (renamed to τ, which represents internal actions).

In [12] it is proven that DPBB is a congruence for parallel composition without synchronisation between the parallel components. However, in general, parallel composition involves some synchronisation mechanism, and compositional reasoning requires equivalences that are a congruence for parallel composition in which parallel components may synchronise their behaviour. Languages to model concurrent systems, such as the process algebras CCS [13], CSP [14], ACP [15], mCRL2 [16], and LOTOS [17], include a parallel composition operator that supports synchronisation. Therefore, in the following, when we refer to parallel composition, we imply, unless stated otherwise, that synchronisation is involved.

It is known that branching bisimilarity is a congruence for the parallel composition of LTSs. This follows from the fact that parallel composition of LTSs can be expressed in a so-called *BB cool* (Branching Bisimulation cool) language [18]. The authors of [19] have proposed a congruence format for DPBB, from which it follows that DPBB is a congruence for parallel composition of LTSs. We place such a congruence proof in the setting of *networks of LTSs* [20] (or LTS networks for short) and compositional model checking. The current article extends earlier work [21] that was conducted in parallel to [19]. Most of the proofs in the current article have been constructed using the Coq proof assistant [22]. Apart from our earlier work, no results obtained with the use of a proof assistant have been reported so far.

A popular toolbox that offers a selection of compositional approaches is Cadp [23]. Cadp offers both *property-independent* approaches (e.g., compositional model generation, smart reduction, and compositional reasoning via behavioural interfaces) and *property-dependent* approaches (e.g., property-dependent reductions [11] and Partial Model Checking [6]). The formal semantics of concurrent systems are described using LTS networks. An LTS network consists of n LTSs representing the parallel processes. A set of synchronisation laws $\mathcal{V}$ is used to describe the possible communication, i.e., synchronisation, between the process LTSs. With this synchronisation mechanism, the usual parallel composition operators of standard process algebras, such as ACP, CCS, CSP, mCRL2, and LOTOS, can be encoded.

Contributions The current article considers parallel composition of LTS networks. First, given two LTS networks $\mathcal{M}$ and $\mathcal{M}'$ of size n related via a DPBB relation B, another LTS network $\mathcal{N}$ of size m, and a parallel composition operator $\|_\sigma$ with a partial function σ that specifies synchronisation between components, we show there is a DPBB relation C such that

$$\mathcal{M}\ \mathsf{B}\ \mathcal{M}' \implies \mathcal{M} \|_\sigma \mathcal{N}\ \mathsf{C}\ \mathcal{M}' \|_\sigma \mathcal{N}$$

In words, we prove that DPBB is a congruence for the parallel composition of two LTS networks. This result subsumes the composition of two individual LTSs via composition of LTS networks of size one.

Second, we present a method to safely decompose an LTS network in components such that the composition of the components is equivalent to the original LTS network. All proofs for the first and second contribution have been mechanically verified using the Coq proof assistant and are available online.[3]

Third, we discuss the effectiveness of compositionally constructing state spaces with intermediate LTS minimisations modulo DPBB in comparison with the classical, non-compositional state space construction. The discussion is based on results we obtained after having conducted a number of experiments using the Cadp toolbox.

This article extends previous work [21], which originally presented the first three contributions, in a number of ways.

First, we prove for a particular definition of σ that the resulting parallel composition operator, which we refer to as $\|_\cap$, is both associative and commutative. That σ-definition defines that the transition labels in the LTSs that the components have

[3] http://www.win.tue.nl/mdse/composition/DPBB_is_a_congruence_for_synchronizing_LTSs.zip.

in common need to synchronise, i.e., it defines synchronisation on the common alphabet of the components. Therefore, this parallel composition operator coincides with the CSP parallel composition operator $\|$ [14]. Associativity and commutativity are desirable properties as they indicate that composition of more than two LTS networks may be done in any order. Synchronisation on the common alphabet is therefore often used in practice for compositional model checking (for instance, see [24–26]). In this article, we show that this can be done safely, by proving that DPBB is a congruence for LTS networks, as defined in [23], if parallel composition is performed with synchronisation on the common alphabet of the components.

Second, we prove that it is actually unnecessary to require synchronisation on the common alphabet for DPBB to be a congruence for LTS networks. This is a useful result, as that restriction excludes many LTS networks in practice. In this case, we generalise the congruence definition of [23].

Third, we have extended the number of test cases originally presented in [21] showing a higher variety in the effectiveness of the approach. Crouzen and Lang [27] report on experiments comparing the run-time and memory performance of three compositional verification techniques. As opposed to these experiments, our experiments concern the comparison of compositional and classical, non-compositional state space construction.

Structure of the article Related work is discussed in Section 2. In Section 3, we discuss the notions of LTS, LTS network, so-called *LTS network admissibility*, and DPBB. Next, the formal composition of LTS networks is presented in Section 4. We prove that DPBB is a congruence for the composition of two LTS networks. Section 5 is on the decomposition of an LTS network. Decomposition allows the redefinition of a system as a set of components. Section 6 introduces an instance of the composition operator that is both associative and commutative. In Section 7, we prove that DPBB is a congruence for LTS networks, as defined in [23], if the set of synchronisation laws implements synchronisation on the common alphabet of the components. Furthermore, we prove that a generalisation of that definition holds as well, which demonstrates that synchronisation on the common alphabet of the components is, in fact, not required. In Section 8, we apply the theoretical results to a set of use cases comparing a compositional construction approach with non-compositional state space construction. In Section 9 we present the conclusions and directions for future work.

2. Related work

Networks of LTSs are introduced in [28]. The authors mention that strong and branching bisimilarity are congruences for the operations supported by LTS networks. Among these operations is the parallel composition with synchronisation on equivalent labels. A congruence proof for branching bisimilarity has been verified in PVS by van de Pol and a textual proof was written, but both the textual proof and the PVS proof have not been made public [29]. An axiomatisation for a rooted version of divergence-preserving branching bisimulation has been performed in a Master graduation project [30]. However, the considered language does not include parallel composition. In this article, we formally show that DPBB is also a congruence for parallel composition with synchronisations between components. As DPBB is a branching bisimulation relation with an extra case for explicit divergence, the proof we present also formally shows that branching bisimilarity is a congruence for parallel composition with synchronisations between components.

Another approach supporting compositional verification is presented in [20]. Given an LTS network and a component selected from the network, the approach automatically generates an interface LTS from the remainder of the network. This remainder is called the environment. The interface LTS represents the synchronisation possibilities that are offered by the environment. This requires the construction and reduction of the system LTS of the environment. The advantage of this method is that transitions and states that do not contribute to the system LTS can be removed. In our approach only the system LTS of the considered component must be constructed. The environment is left out of scope until the components are composed.

Many process algebras support parallel composition with synchronisation on labels. Often a proof is given showing that some bisimilarity is a congruence for these operators [16,31–33]. To generalize the congruence proofs a series of meta-theories has been proposed for algebras with parallel composition [18,34,35]. In [35] the *panth* format is proposed. The authors show that strong bisimilarity is a congruence for algebras that adhere to the panth format. The focus of the work is on the expressiveness of the format. The author of [18] proposes so-called *cool* formats for four bisimilarities: weak bisimilarity, rooted weak bisimilarity, branching bisimilarity, and rooted branching bisimilarity. It is shown that these bisimilarities are congruences for the corresponding formats. In [34] similar formats are proposed for eager bisimilarity and branching bisimilarity. Eager bisimilarity is a kind of weak bisimilarity which is sensitive to divergence. The above mentioned formats do not consider DPBB.

Recently, the authors of [19] have proposed such a format for DPBB. In parallel to that work, we have proven that DPBB is a congruence for the parallel composition of two LTS networks with synchronisation on transition labels, using the Coq proof assistant [21]. In the current article, we additionally prove that DPBB is a congruence for LTS networks, as defined in [23]. Furthermore, we define how to compose and decompose LTS networks, and demonstrate the practical effectiveness of compositional model checking based on DPBB using a larger set of benchmarks than the one used previously in [21].

In earlier work, we presented decomposition for LTS transformation systems of LTS networks [36]. The work aims to verify the transformation of a component that may synchronise with other components. The paper proposes to calculate so called detaching laws which are similar to our interface laws. The approach can be modelled with our method. In fact, we

show that the derivation of these detaching laws does not amount to a desired decomposition, i.e., the re-composition of the decomposition is *not* equivalent to the original system (see Example 5.1 discussed in Section 5).

A projection of an LTS network given a set of indices is presented in [23]. Their projection operator is similar to the consistent decomposition of LTS networks that we propose. In fact, with a suitable operator for the reordering of LTS networks our decomposition operator is equivalent to their projection operator. The current article contributes to these results that admissibility properties of the LTS network are indeed preserved for such consistent decompositions.

3. Preliminaries

In this section, we introduce the notions of LTS, LTS network, and divergence-preserving branching bisimilarity of LTSs. The potential behaviour of processes is described by means of LTSs. The behaviour of a concurrent system is described by a *network of LTSs* [20], or *LTS network* for short. The semantics of an LTS network $\mathcal{M}$ is defined by an LTS $\mathcal{G}_{\mathcal{M}}$, which describes the global behaviour of the network. To compare the behaviour of processes and systems, the notion of divergence-preserving branching bisimilarity (DPBB) is used. DPBB is often used to reduce the state space of system specifications while preserving safety and liveness properties, or to compare the observable behaviour of two systems.

The semantics of a process, or a composition of several processes, can be formally expressed by an LTS as presented in Definition 3.1.

Definition 3.1 *(Labelled transition system).* An *LTS* $\mathcal{G}$ is a tuple $(\mathcal{S}_{\mathcal{G}}, \mathcal{A}_{\mathcal{G}}, \mathcal{T}_{\mathcal{G}}, \mathcal{I}_{\mathcal{G}})$, with

- $\mathcal{S}_{\mathcal{G}}$ a non-empty, finite set of states;
- $\mathcal{A}_{\mathcal{G}}$ a set of action labels;
- $\mathcal{T}_{\mathcal{G}} \subseteq \mathcal{S}_{\mathcal{G}} \times \mathcal{A}_{\mathcal{G}} \times \mathcal{S}_{\mathcal{G}}$ a transition relation;
- $\mathcal{I}_{\mathcal{G}} \subseteq \mathcal{S}_{\mathcal{G}}$ a (non-empty) set of initial states.

Action labels in $\mathcal{A}_{\mathcal{G}}$ are denoted by a, b, c, etc. Additionally, there is a special action label $\tau \in \mathcal{A}_{\mathcal{G}}$ that represents internal, or hidden, system steps. A transition $(s, a, s') \in \mathcal{T}_{\mathcal{G}}$, or $s \xrightarrow{a}_{\mathcal{G}} s'$ for short, denotes that LTS $\mathcal{G}$ can move from state s to state s' by performing the a-action. The transitive reflexive closure of $\xrightarrow{a}_{\mathcal{G}}$ is denoted as $\xrightarrow{a}{}^{*}_{\mathcal{G}}$, and the transitive closure is denoted as $\xrightarrow{a}{}^{+}_{\mathcal{G}}$.

LTS network An LTS network, presented in Definition 3.2, describes a system consisting of a finite number of concurrent process LTSs and a set of synchronisation laws that define the possible interaction between the processes. We write $1..n$ for the set of integers ranging from 1 to n. A vector $\bar{v}$ of size n contains n elements indexed from 1 to n. For all $i \in 1..n$, $\bar{v}_i$ represents the ith element of vector $\bar{v}$. The *concatenation* of two vectors $\bar{v}$ and $\bar{w}$ of size n and m, respectively, is denoted by $\bar{v} \parallel \bar{w}$. In the context of composition of LTS networks, this concatenation of vectors corresponds to the parallel composition of the behaviour referred to by the two vectors.

Definition 3.2 *(LTS network).* An LTS network $\mathcal{M}$ of size n is a pair $(\Pi, \mathcal{V})$, where

- Π is a vector of n concurrent LTSs. For each $i \in 1..n$, we write $\Pi_i = (\mathcal{S}_i, \mathcal{A}_i, \mathcal{T}_i, \mathcal{I}_i)$.
- $\mathcal{V}$ is a finite set of synchronisation laws. A *synchronisation law* is a tuple $(\bar{v}, a)$, where $\bar{v}$ is a vector of size n, called the *synchronisation vector*, containing synchronising action labels, and a is an action label representing the result of successful synchronisation. We have $\forall i \in 1..n.\ \bar{v}_i \in \mathcal{A}_i \cup \{\bullet\}$, where $\bullet$ is a special symbol denoting that Π_i performs no action. The *set of result actions* of a set of synchronisation laws $\mathcal{V}$ is defined as $\mathcal{A}_{\mathcal{V}} = \{a \mid (\bar{v}, a) \in \mathcal{V}\}$.

The semantics of an LTS network $\mathcal{M}$ is defined by the LTS $\mathcal{G}_{\mathcal{M}}$, which is obtained by combining the processes in Π according to the synchronisation laws in $\mathcal{V}$ as specified by Definition 3.3. The LTS network model subsumes most hiding, renaming, cutting, and parallel composition operators present in process algebras. For instance, hiding can be applied by replacing the a component in a law by τ.

Definition 3.3 *(LTS network semantics).* Given an LTS network $\mathcal{M} = (\Pi, \mathcal{V})$, its *semantics* is defined by $\mathcal{G}_{\mathcal{M}} = (\mathcal{S}_{\mathcal{M}}, \mathcal{A}_{\mathcal{M}}, \mathcal{T}_{\mathcal{M}}, \mathcal{I}_{\mathcal{M}})$, with

- $\mathcal{I}_{\mathcal{M}} = \{\langle s_1, \ldots, s_n \rangle \mid s_i \in \mathcal{I}_i\}$;
- $\mathcal{T}_{\mathcal{M}}$ and $\mathcal{S}_{\mathcal{M}}$ are the smallest relation and set, respectively, satisfying $\mathcal{I}_{\mathcal{M}} \subseteq \mathcal{S}_{\mathcal{M}}$ and for all $\bar{s} \in \mathcal{S}_{\mathcal{M}}$, $a \in \mathcal{A}_{\mathcal{V}}$, we have $\bar{s} \xrightarrow{a}_{\mathcal{M}} \bar{s}'$ and $\bar{s}' \in \mathcal{S}_{\mathcal{M}}$ iff there exists $(\bar{v}, a) \in \mathcal{V}$ such that for all $i \in 1..n$:

$$\begin{cases} \bar{s}_i = \bar{s}'_i & \text{if } \bar{v}_i = \bullet \\ \bar{s}_i \xrightarrow{\bar{v}_i}_{\Pi_i} \bar{s}'_i & \text{otherwise} \end{cases}$$

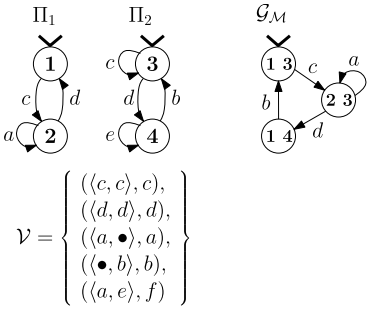

Fig. 1. An LTS network $\mathcal{M} = (\Pi, \mathcal{V})$ (left) and its system LTS $\mathcal{G}_\mathcal{M}$ (right).

- $\mathcal{A}_\mathcal{M} = \{a \mid \exists \bar{s}, \bar{s}' \in \mathcal{S}_\mathcal{M}. \bar{s} \xrightarrow{a}_\mathcal{M} \bar{s}'\}$.

In Fig. 1, an example of an LTS network $\mathcal{M} = (\langle \Pi_1, \Pi_2\rangle, \mathcal{V})$ with four synchronisation laws is shown on the left, and its semantics in the form of an LTS $\mathcal{G}_\mathcal{M}$ is shown on the right. Initial states are indicated with an incoming arrow. The states of the system LTS $\mathcal{G}_\mathcal{M}$ are constructed by combining the states of Π_1 and Π_2. In this example, we have $\langle 1, 3\rangle, \langle 1, 4\rangle, \langle 2, 3\rangle \in \mathcal{S}_\mathcal{M}$, of which $\langle 1, 3\rangle$ is the single initial state of $\mathcal{G}_\mathcal{M}$.

The transitions of $\mathcal{G}_\mathcal{M}$ in Fig. 1 are constructed by combining the transitions of Π_1 and Π_2 according to the set of synchronisation laws $\mathcal{V}$. Law $(\langle c, c\rangle, c)$ specifies that the process LTSs can synchronise on their c-transitions, resulting in c-transitions in $\mathcal{G}_\mathcal{M}$. Similarly, the process LTSs can synchronise on their d-transitions, resulting in a d-transition in $\mathcal{G}_\mathcal{M}$. Furthermore, law $(\langle a, \bullet\rangle, a)$ specifies that process Π_1 can perform an a-transition independently resulting in an a-transition in $\mathcal{G}_\mathcal{M}$. Likewise, law $(\langle \bullet, b\rangle, b)$ specifies that the b-transition can be fired independently by process Π_2. Because Π_1 does not participate in this law, it remains in state $\langle 1\rangle$ in $\mathcal{G}_\mathcal{M}$. The last law states that a- and e-transitions can synchronise, resulting in f-transitions, however, in this example the a- and e-transitions in Π_1 and Π_2 are never able to synchronise since state $\langle 2, 4\rangle$ is unreachable.

An LTS network is called *admissible* iff the synchronisation laws of the network do not synchronise, rename, or cut τ-transitions [20] as defined in Definition 3.4. The intuition behind this is that internal, i.e., hidden, behaviour should not be restricted by any operation. Partial model checking and compositional construction rely on LTS networks being admissible [23]. Hence, in this article, we also restrict ourselves to admissible LTS networks when presenting our composition and decomposition methods.

Definition 3.4 *(LTS network admissibility).* An LTS network $\mathcal{M} = (\Pi, \mathcal{V})$ of size n is called admissible iff the following properties hold:

1. $\forall(\bar{\nu}, a) \in \mathcal{V}. \exists i \in 1..n.\ \bar{\nu}_i = \tau \implies \forall j \neq i.\ \bar{\nu}_j = \bullet$; (no synchronisation of τ's)
2. $\forall(\bar{\nu}, a) \in \mathcal{V}. \exists i \in 1..n.\ \bar{\nu}_i = \tau \implies a = \tau$; (no renaming of τ's)
3. $\forall i \in 1..n.\ \tau \in \mathcal{A}_i \implies \exists(\bar{\nu}, a) \in \mathcal{V}.\ \bar{\nu}_i = \tau$. (no cutting of τ's)

Divergence-preserving branching bisimilarity To compare LTSs, we use DPBB, also called *branching bisimilarity with explicit divergence* [9,10]. DPBB supports abstraction from actions and preserves both safety and liveness properties. To simplify proofs we use DPBB with the weakest divergence condition (D_4) presented in [10] as presented in Definition 3.5. This definition is equivalent to the standard definition of DPBB [10]. The smallest infinite ordinal is denoted by ω.

Definition 3.5 *(Divergence-preserving branching bisimulation).* A binary relation B between two LTSs $\mathcal{G}_1$ and $\mathcal{G}_2$ is a *divergence-preserving branching bisimulation* iff for all $s \in \mathcal{S}_{\mathcal{G}_1}$ and $t \in \mathcal{S}_{\mathcal{G}_2}$, $s\ \mathsf{B}\ t$ implies:

1. if $s \xrightarrow{a}_{\mathcal{G}_1} s'$ then
   (a) either $a = \tau$ with $s'\ \mathsf{B}\ t$;
   (b) or $t \xrightarrow{\tau}{}^*_{\mathcal{G}_2} \hat{t} \xrightarrow{a}_{\mathcal{G}_2} t'$ with $s\ \mathsf{B}\ \hat{t}$ and $s'\ \mathsf{B}\ t'$.
2. symmetric to 1.
3. if there is an infinite sequence of states $(s^k)_{k\in\omega}$ such that $s = s^0$, $s^k \xrightarrow{\tau}_{\mathcal{G}_1} s^{k+1}$ and $s^k\ \mathsf{B}\ t$ for all $k \in \omega$, then there exists a state t' such that $t \xrightarrow{\tau}{}^+_{\mathcal{G}_2} t'$ and $s^k\ \mathsf{B}\ t'$ for some $k \in \omega$.
4. symmetric to 3.

Condition 3 (and its symmetric case) is illustrated in Fig. 2. For every state t that is related by B to an infinite number of states along an infinite path of τ-transitions, there exists at least one state t' reachable from t via at least one τ-transition

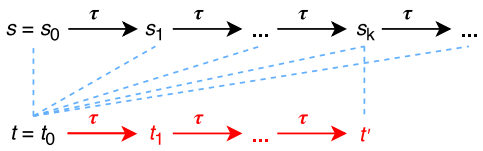

Fig. 2. Condition 3 of Definition 3.5.

that is related by B to one of those states. In other words, the infinite sequence of τ-transitions cannot be simulated in t by zero τ-transitions. Van Glabbeek et al. [10] have proven that this condition coincides with the preservation of divergence.

Two states $s \in \mathcal{S}_{\mathcal{G}_1}$ and $t \in \mathcal{S}_{\mathcal{G}_2}$ are *divergence-preserving branching bisimilar*, denoted by $s \underline{\leftrightarrow}_b^{\Delta} t$, iff there is a DPBB relation B such that $s \mathrel{B} t$. We say that two LTSs $\mathcal{G}_1$ and $\mathcal{G}_2$ are divergence-preserving branching bisimilar, denoted by $\mathcal{G}_1 \underline{\leftrightarrow}_b^{\Delta} \mathcal{G}_2$, iff $\forall s_1 \in \mathcal{I}_{\mathcal{G}_1}.\exists s_2 \in \mathcal{I}_{\mathcal{G}_2}.\, s_1 \underline{\leftrightarrow}_b^{\Delta} s_2$ and vice versa. Finally, we say that two LTS networks $\mathcal{M}$, $\mathcal{M}'$ are divergence-preserving branching bisimilar, denoted by $\mathcal{M} \underline{\leftrightarrow}_b^{\Delta} \mathcal{M}'$, iff $\mathcal{G}_{\mathcal{M}} \underline{\leftrightarrow}_b^{\Delta} \mathcal{G}_{\mathcal{M}'}$.

4. Composition of LTS networks

This section introduces the compositional construction of LTS networks. Such a composition achieves the parallel composition of process LTSs, resulting in an LTS network $\mathcal{M}$. For that network, $\mathcal{G}_{\mathcal{M}}$ tends to grow exponentially as processes are added.

An LTS network can be seen as being composed of several *components*, each of which consists of a number of individual processes in parallel composition, with *intra-component* synchronisation laws describing how the processes inside a component should synchronise with each other. Furthermore, *inter-component* synchronisation laws define how the components as a whole should synchronise with each other. Compositional construction of a minimal version of the LTS defining the semantics of the LTS network may then be performed by first constructing the LTSs of the different components, subsequently minimising these, and finally combining their behaviour. Example 4.1 presents an example of a network with two components and an inter-component synchronisation law.

Example 4.1 *(Component).* Consider an LTS network $\mathcal{M} = (\Pi, \mathcal{V})$ with processes $\Pi = \langle \Pi_1, \Pi_2, \Pi_3 \rangle$ and synchronisation laws $\mathcal{V} = \{(\langle a, \bullet, \bullet \rangle, a), (\langle \bullet, b, b \rangle, b), (\langle c, c, c \rangle, c)\}$. We may split up the network in two *components*, say $M_1 = (\langle \Pi_1 \rangle, \mathcal{V}_1)$ and $\mathcal{M}_{\{2,3\}} = (\langle \Pi_2, \Pi_3 \rangle, \mathcal{V}_{\{2,3\}})$. Then, $(\langle c, c, c \rangle, c)$ is an *inter-component law* describing synchronisation between $\mathcal{M}_1$ and $\mathcal{M}_{\{2,3\}}$. The component $\mathcal{M}_1$ consists of process Π_1, and the set of intra-component synchronisation laws $\mathcal{V}_1 = \{(\langle a, \bullet, \bullet \rangle, a)\}$ operating solely on Π_1. Similarly, component $\mathcal{M}_{\{2,3\}}$ consists of Π_2 and Π_3, and the set of intra-component synchronisation laws $\mathcal{V}_{\{2,3\}} = \{(\langle \bullet, b, b \rangle, b)\}$ operating solely on Π_2 and Π_3.

The challenge of compositional construction is to allow manipulation of the components while guaranteeing that the observable behaviour of the system as a whole remains equivalent modulo DPBB. Even though synchronisation laws of a component may be changed, we must somehow preserve synchronisations with the other components. Such a change of synchronisation laws occurs, for instance, when reordering the processes in a component, or renaming actions that are part of inter-component synchronisation laws.

In this and the following section, we limit ourselves to the composition of two components: a left and a right component. This simplifies notations and proofs. However, the approach can be generalised to splitting networks given two sets of indices indicating which processes are part of which component, i.e., a projection operator can be used to project distinct parts of a network into components.

In the remainder of this section, first, we formalise LTS networks composition. Then, we show that admissibility is preserved when two admissible networks are composed. Finally, we prove that DPBB is a congruence for composition of two LTS networks.

Composing LTS networks Before defining the composition of two networks, we introduce a partial function indicating how the inter-component laws should be constructed from the interfaces of the two networks. An inter-component law can then be constructed by combining the interface vectors of the components and adding a result action. This is achieved through a given *interface function*, presented in Definition 4.1, relating interface actions to result actions.

Definition 4.1 *(Interface function).* Consider two LTS networks $\mathcal{M}_\Pi = (\Pi, \mathcal{V})$ and $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \mathcal{W})$ of size n and m, respectively. An *interface function* between $\mathcal{M}_\Pi$ and $\mathcal{M}_\mathrm{P}$ is a partial function $\sigma : \mathcal{A}_\mathcal{V} \cup \mathcal{A}_\mathcal{W} \setminus \{\tau\} \times \mathcal{A}_\mathcal{V} \cup \mathcal{A}_\mathcal{W} \setminus \{\tau\} \nrightarrow \mathcal{A}$ describing how the interface actions of $\mathcal{M}_\Pi$ should be combined with interface actions of $\mathcal{M}_\mathrm{P}$, and what the action label should be resulting from successful synchronisation. The set $\mathcal{A}$ is the set of actions resulting from successful synchronisation between Π and P. The actions related by σ are considered the interface actions.

The reason σ can be defined for pairs of actions from $\mathcal{A}_\mathcal{V} \cup \mathcal{A}_\mathcal{W}$, as opposed to an action from $\mathcal{A}_\mathcal{V}$ with an action from $\mathcal{A}_\mathcal{W}$, is to allow for the cutting of actions, i.e., allowing to define that an action of one component has to synchronise with some action from the other component that can actually never be executed.

An interface function implicitly defines how inter-component synchronisation laws should be represented in the separate components. These local representatives are called the *interface synchronisation laws*. An interface function σ for LTS networks $\mathcal{M}_\Pi = (\Pi, \mathcal{V})$ and $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \mathcal{W})$ implies the following sets of interface synchronisation laws:

$$\mathcal{V}_\sigma = \{(\bar{v}, a) \in \mathcal{V} \mid \exists b \in \mathcal{A}_\mathcal{V} \cup \mathcal{A}_\mathcal{W} \setminus \{\tau\}, c \in \mathcal{A}.\sigma(a, b) = c\}$$
$$\mathcal{W}_\sigma = \{(\bar{w}, b) \in \mathcal{W} \mid \exists a \in \mathcal{A}_\mathcal{V} \cup \mathcal{A}_\mathcal{W} \setminus \{\tau\}, c \in \mathcal{A}.\sigma(a, b) = c\}$$

An interface synchronisation law makes a component's potential to synchronise with other components explicit. An interface synchronisation law has a synchronisation vector, called the *interface vector*. The result action of an interface synchronisation law is called an *interface action*. These notions are clarified further in Example 4.2.

Example 4.2 *(Interface vector and interface law).* Let $\mathcal{M} = (\langle \Pi_1, \Pi_2, \Pi_3 \rangle, \mathcal{V})$ be a network with inter-component synchronisation law $(\langle a, a, b \rangle, c) \in \mathcal{V}$, a component $M_{\{1,2\}} = (\langle \Pi_1, \Pi_2 \rangle, \mathcal{V}_{\{1,2\}})$, and a component $M_3 = (\langle \Pi_3 \rangle, \mathcal{V}_3)$. Then, $\langle a, a \rangle$ is an *interface vector* of $M_{\{1,2\}}$, and given a corresponding *interface action* α for $\mathcal{M}_{\{1,2\}}$ ($\alpha \in \mathcal{A}_{\mathcal{M}_1} \cup \mathcal{A}_{\mathcal{M}_2} \setminus \{\tau\}$), an *interface law* of $\mathcal{M}_{\{1,2\}}$ is $(\langle a, a \rangle, \alpha)$.

Together, the interface laws and interface function define the possible synchronisations between two components, i.e., the interface laws and interface function define inter-component synchronisation laws. Given two sets of laws $\mathcal{V}$ and $\mathcal{W}$ and an interface function σ, the inter-component synchronisation laws are defined by the following function $\mathcal{L}_\sigma$:

$$\mathcal{L}_\sigma(\mathcal{V}, \mathcal{W}) = \{(\bar{v} \parallel \bar{w}, a) \mid \exists \alpha, \beta.(\bar{v}, \alpha) \in \mathcal{V} \wedge (\bar{w}, \beta) \in \mathcal{W} \wedge \sigma(\alpha, \beta) = a\}$$

The interface function σ partitions both $\mathcal{V}$ and $\mathcal{W}$ into two sets of synchronisation laws: the interface and non-interface synchronisation laws.

The application of the interface function, i.e., the formal composition of two LTS networks, is presented in Definition 4.2. We show that a component may be exchanged with a divergence-preserving branching bisimilar component iff the interface actions are not hidden. The interface between the component and the remainder of the network is respected when the interface actions remain observable.

Definition 4.2 *(Composition of LTS networks).* Consider LTS networks $\mathcal{M}_\Pi = (\Pi, \mathcal{V})$ of size n and $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \mathcal{W})$ of size m. Let $\sigma : \mathcal{A}_\mathcal{V} \cup \mathcal{A}_\mathcal{W} \setminus \{\tau\} \times \mathcal{A}_\mathcal{V} \cup \mathcal{A}_\mathcal{W} \setminus \{\tau\} \nrightarrow \mathcal{A}$ be an interface function describing the synchronisations between $\mathcal{M}_\Pi$ and $\mathcal{M}_\mathrm{P}$. The *composition* of $\mathcal{M}_\Pi$ and $\mathcal{M}_\mathrm{P}$, denoted by $\mathcal{M}_\Pi \parallel_\sigma \mathcal{M}_\mathrm{P}$, is defined as the LTS network $(\Pi \parallel \mathrm{P}, \mathcal{V} \parallel \mathcal{W})$, where $\mathcal{V} \parallel \mathcal{W} = (\mathcal{V} \setminus \mathcal{V}_\sigma)^\bullet \cup {}^\bullet(\mathcal{W} \setminus \mathcal{W}_\sigma) \cup \mathcal{L}_\sigma(\mathcal{V}, \mathcal{W})$ with $(\mathcal{V} \setminus \mathcal{V}_\sigma)^\bullet = \{(\bar{v} \parallel \bullet^m, a) \mid (\bar{v}, a) \in \mathcal{V} \setminus \mathcal{V}_\sigma\}$ and ${}^\bullet(\mathcal{W} \setminus \mathcal{W}_\sigma) = \{(\bullet^n \parallel \bar{w}, a) \mid (\bar{w}, a) \in \mathcal{W} \setminus \mathcal{W}_\sigma\}$ the sets of synchronisation laws $\mathcal{V} \setminus \mathcal{V}_\sigma$ postfixed with m $\bullet$'s and $\mathcal{W} \setminus \mathcal{W}_\sigma$ prefixed with n $\bullet$'s, respectively.

Our parallel composition operator $\parallel_\sigma$ subsumes parallel composition operators from process algebras such as CCS, CSP, ACP, mCRL2 and LOTOS. For instance, consider the LOTOS parallel composition operator $|[A]|$, which defines that two components have to synchronise on actions (or 'gates' in LOTOS) in A, while all other actions can be executed independently (with the rule that the internal action, similar to our τ, is not in A). For two LTS networks $\mathcal{M}_\Pi = (\Pi, \mathcal{V})$, $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \mathcal{W})$, we can achieve the same by defining that $\sigma(a, a) = a$ for all $a \in A \cap (\mathcal{A}_\mathcal{V} \cup \mathcal{A}_\mathcal{W})$, and leaving σ undefined for all other pairs of actions.

As presented in Proposition 4.1, LTS networks that are composed (according to Definition 4.2) from two admissible networks are admissible as well.

Proposition 4.1. *Let $\mathcal{M}_\Pi = (\Pi, \mathcal{V})$ and $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \mathcal{W})$ be admissible LTS networks of size n and m, respectively. Furthermore, let $\sigma : \mathcal{A}_\mathcal{V} \cup \mathcal{A}_\mathcal{W} \setminus \{\tau\} \times \mathcal{A}_\mathcal{V} \cup \mathcal{A}_\mathcal{W} \setminus \{\tau\} \nrightarrow \mathcal{A}$ be an interface function. Then, the network $\mathcal{M} = \mathcal{M}_\Pi \parallel_\sigma \mathcal{M}_\mathrm{P}$ is also admissible.*

Proof. We show that $\mathcal{M}$ satisfies Definition 3.4:

- *No synchronisation and renaming of τ's.* Let $(\bar{v}, a) \in (\mathcal{V} \setminus \mathcal{V}_\sigma)^\bullet \cup {}^\bullet(\mathcal{W} \setminus \mathcal{W}_\sigma) \cup \mathcal{L}_\sigma(\mathcal{V}, \mathcal{W})$ be a synchronisation law with $\bar{v}_i = \tau$ for some $i \in 1..(n+m)$. We distinguish two cases:
  - $(\bar{v}, a) \in (\mathcal{V} \setminus \mathcal{V}_\sigma)^\bullet \cup {}^\bullet(\mathcal{W} \setminus \mathcal{W}_\sigma)$. By construction of $(\mathcal{V} \setminus \mathcal{V}_\sigma)^\bullet$ and ${}^\bullet(\mathcal{W} \setminus \mathcal{W}_\sigma)$, and admissibility of $\mathcal{M}_\Pi$ and $\mathcal{M}_\mathrm{P}$, we have $\forall j \in 1..n.\ \bar{v}_j \neq \bullet \implies i = j$, $\forall j \in (n+1)..(n+m).\ \bar{v}_j \neq \bullet \implies i = j$ and $a = \tau$. Hence, it holds that $\forall j \in 1..(n+m).\ \bar{v}_j \neq \bullet \implies i = j$ (no synchronisation of τ's) and $a = \tau$ (no renaming of τ's).
  - $(\bar{v}, a) \in \mathcal{L}_\sigma(\mathcal{V}, \mathcal{W})$. By definition of $\mathcal{L}_\sigma(\mathcal{V}, \mathcal{W})$, there are interface laws $(\bar{v}', \alpha) \in \mathcal{V}$ and $(\bar{v}'', \beta) \in \mathcal{W}$ such that $\sigma(\alpha, \beta) = a$. Hence, either $1 \leq i \leq n$ with $\bar{v}'_i = \tau$ or $n < i \leq n+m$ with $\bar{v}''_{i-n} = \tau$. Since $\mathcal{M}_\Pi$ and $\mathcal{M}_\mathrm{P}$ are admissible, we must have $\alpha = \tau$ or $\beta = \tau$, respectively. However, the interface function does not allow τ as interface actions, therefore, the proof follows by contradiction.

It follows that $\mathcal{M}$ does not allow synchronisation and renaming of τ's.

- *No cutting of τ's.* Let $(\Pi \parallel \mathrm{P})_i$ be a process $(i \in 1..(n+m))$ with $\tau \in \mathcal{A}_{(\Pi \parallel \mathrm{P})_i}$. We distinguish the two cases $1 \leq i \leq n$ and $n < i \leq n+m$. It follows that $\tau \in \mathcal{A}_{\Pi_i}$ for the former case and $\tau \in \mathcal{A}_{\mathrm{P}_{i-n}}$ for the latter case. Since both $\mathcal{M}_\Pi$ and $\mathcal{M}_\mathrm{P}$ are admissible and no actions are removed when constructing $(\mathcal{V} \setminus \mathcal{V}_\sigma)^\bullet$ and ${}^\bullet(\mathcal{W} \setminus \mathcal{W}_\sigma)$ from $\mathcal{V}$ and $\mathcal{W}$, respectively, in both cases there exists a $(\bar{v}, a) \in (\mathcal{V} \setminus \mathcal{V}_\sigma)^\bullet \cup {}^\bullet(\mathcal{W} \setminus \mathcal{W}_\sigma) \cup \mathcal{L}_\sigma(\mathcal{V}, \mathcal{W})$ such that $\bar{v}_i = \tau$. Hence, the composite network $\mathcal{M}$ does not allow cutting of τ's.

Since the three admissibility properties hold, the composed network $\mathcal{M}$ satisfies Definition 3.4. □

DPBB is a congruence for the composition of two LTS networks Proposition 4.2 shows that DPBB is a congruence for the composition of two LTS networks according to Definition 4.2. It is worth noting that an interface function does not relate τ's, i.e., synchronisation of τ-actions is not allowed.

Note that Proposition 4.2 subsumes the composition of two LTSs, via composition of LTS networks of size one with trivial sets of intra-component synchronisation laws.

Proposition 4.2. *Consider LTS networks $\mathcal{M}_\Pi = (\Pi, \mathcal{V})$ and $\mathcal{M}_{\Pi'} = (\Pi', \mathcal{V}')$, both of size n, and $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \mathcal{W})$ of size m. Let σ be an interface function defining the synchronisation of actions in $\mathcal{A}_\mathcal{V} \cap \mathcal{A}_{\mathcal{V}'}$ and $\mathcal{A}_\mathcal{W}$. DPBB is a* congruence *for the composition of two LTS networks, i.e., it holds that*

$$\mathcal{M}_\Pi \underline{\leftrightarrow}_b^\Delta \mathcal{M}_{\Pi'} \implies \mathcal{M}_\Pi \parallel_\sigma \mathcal{M}_\mathrm{P} \underline{\leftrightarrow}_b^\Delta \mathcal{M}_{\Pi'} \parallel_\sigma \mathcal{M}_\mathrm{P}$$

Proof. Intuitively, we have $\mathcal{M}_\Pi \parallel_\sigma \mathcal{M}_\mathrm{P} \underline{\leftrightarrow}_b^\Delta \mathcal{M}_{\Pi'} \parallel_\sigma \mathcal{M}_\mathrm{P}$ because $\mathcal{M}_\Pi \underline{\leftrightarrow}_b^\Delta \mathcal{M}_{\Pi'}$ and the interface with $\mathcal{M}_\mathrm{P}$ is respected. Since $\mathcal{M}_\Pi \underline{\leftrightarrow}_b^\Delta \mathcal{M}_{\Pi'}$, whenever a transition labelled with an interface action α in $\mathcal{M}_\Pi$ is able to perform a transition together with $\mathcal{M}_\mathrm{P}$, then $\mathcal{M}_{\Pi'}$ is able to simulate the interface α-transition, i.e., perform the transition, possibly after having performed a sequence of τ-transitions in which the visited states are all divergence-preserving branching bisimilar with each other, and synchronise with $\mathcal{M}_\mathrm{P}$ as well. It follows that the branching structure and divergence is preserved. For the sake of brevity we define the following shorthand notations: $\mathcal{M} = \mathcal{M}_\Pi \parallel_\sigma \mathcal{M}_\mathrm{P}$ and $\mathcal{M}' = \mathcal{M}_{\Pi'} \parallel_\sigma \mathcal{M}_\mathrm{P}$. We show $\mathcal{M}_\Pi \underline{\leftrightarrow}_b^\Delta \mathcal{M}_{\Pi'} \implies \mathcal{M} \underline{\leftrightarrow}_b^\Delta \mathcal{M}'$.

Let B be a DPBB relation between the semantics of $\mathcal{M}_\Pi$ and $\mathcal{M}_{\Pi'}$, i.e., $\mathcal{G}_{\mathcal{M}_\Pi} \,\mathsf{B}\, \mathcal{G}_{\mathcal{M}_{\Pi'}}$. By definition, we have $\mathcal{M} \underline{\leftrightarrow}_b^\Delta \mathcal{M}'$ iff $\mathcal{G}_\mathcal{M} \underline{\leftrightarrow}_b^\Delta \mathcal{G}_{\mathcal{M}'}$, which holds iff there exists a DPBB relation C with $\mathcal{I}_\mathcal{M} \,\mathsf{C}\, \mathcal{I}_{\mathcal{M}'}$. We define C as follows, for states $\bar{p} \in \mathcal{S}_{\mathcal{M}_\Pi}$, $\bar{q} \in \mathcal{S}_{\mathcal{M}_{\Pi'}}$, $\bar{r} \in \mathcal{S}_{\mathcal{M}_\mathrm{P}}$:

$$\mathsf{C} = \{(\bar{p} \parallel \bar{r}, \bar{q} \parallel \bar{r}) \mid \bar{p} \,\mathsf{B}\, \bar{q}\}$$

Relation B relates the states of $\mathcal{G}_{\mathcal{M}_\Pi}$ and $\mathcal{G}_{\mathcal{M}_{\Pi'}}$, addressing the component that is subject to change. The unchanged component of the network is related via the shared state vector $\bar{r}$, i.e., it relates the states of $\mathcal{G}_{\mathcal{M}_\mathrm{P}}$ to themselves.

To prove the proposition we have to show that C is a DPBB relation. This requires proving that C relates the initial states of $\mathcal{G}_\mathcal{M}$ and $\mathcal{G}_{\mathcal{M}'}$ and that C satisfies the conditions of Definition 3.5.

Initial. C *relates the initial states of $\mathcal{G}_\mathcal{M}$ and $\mathcal{G}_{\mathcal{M}'}$, i.e., $\mathcal{I}_\mathcal{M} \,\mathsf{C}\, \mathcal{I}_{\mathcal{M}'}$.* We prove that $\forall \bar{s} \in \mathcal{I}_\mathcal{M}.\ \exists \bar{t} \in \mathcal{I}_{\mathcal{M}'}.\ \bar{s} \,\mathsf{C}\, \bar{t}$. The symmetric case can be proven similarly. Take an initial state $\bar{s} = \bar{p} \parallel \bar{r} \in \mathcal{I}_\mathcal{M}$, with $\bar{p} \in \mathcal{I}_{\mathcal{M}_\Pi}$ and $\bar{r} \in \mathcal{I}_{\mathcal{M}_\mathrm{P}}$. Since $\mathcal{I}_{\mathcal{M}_\Pi} \,\mathsf{B}\, \mathcal{I}_{\mathcal{M}_{\Pi'}}$ and $\bar{p} \in \mathcal{I}_{\mathcal{M}_\Pi}$, there exists a $\bar{t} \in \mathcal{I}_{\mathcal{M}_{\Pi'}}$ such that $\bar{p} \,\mathsf{B}\, \bar{t}$. Therefore, we have $\bar{p} \parallel \bar{r} \,\mathsf{C}\, \bar{t} \parallel \bar{r}$. Since $\bar{p} \parallel \bar{r}$ is an arbitrary initial state in $\mathcal{I}_\mathcal{M}$ the proof holds for all states in $\mathcal{I}_\mathcal{M}$. Furthermore, since the other case is symmetrical it follows that $\mathcal{I}_\mathcal{M} \,\mathsf{C}\, \mathcal{I}_{\mathcal{M}'}$.

Def. 3.5, case 1. *For $\bar{s} \in \mathcal{S}_\mathcal{M}$ and $\bar{t} \in \mathcal{S}_{\mathcal{M}'}$, if $\bar{s} \,\mathsf{C}\, \bar{t}$ and $\bar{s} \xrightarrow{a}_\mathcal{M} \bar{s}'$ then either $a = \tau \wedge \bar{s}' \,\mathsf{C}\, \bar{t}$, or $\bar{t} \xrightarrow{\tau}{}^*_{\mathcal{M}'} \hat{\bar{t}} \xrightarrow{a}_{\mathcal{M}'} \bar{t}' \wedge \bar{s} \,\mathsf{C}\, \hat{\bar{t}} \wedge \bar{s}' \,\mathsf{C}\, \bar{t}'$.* To better distinguish between the two parts of the networks, we 'unfold' C and reformulate the proof obligation as follows, with $\bar{s} = \bar{p} \parallel \bar{r}$ and $\bar{t} = \bar{q} \parallel \bar{r}$: *If $\bar{p} \,\mathsf{B}\, \bar{q}$ and $\bar{p} \parallel \bar{r} \xrightarrow{a}_\mathcal{M} \bar{p}' \parallel \bar{r}'$ then either $a = \tau \wedge \bar{p}' \,\mathsf{B}\, \bar{q} \wedge \bar{r} = \bar{r}'$, or $\bar{q} \parallel \bar{r} \xrightarrow{\tau}{}^*_{\mathcal{M}'} \hat{\bar{q}} \parallel \bar{r} \xrightarrow{a}_{\mathcal{M}'} \bar{q}' \parallel \bar{r}' \wedge \bar{p} \,\mathsf{B}\, \hat{\bar{q}} \wedge \bar{p}' \,\mathsf{B}\, \bar{q}'$.* Consider synchronisation law $(\bar{v} \parallel \bar{w}, a) \in (\mathcal{V} \setminus \mathcal{V}_\sigma)^\bullet \cup {}^\bullet(\mathcal{W} \setminus \mathcal{W}_\sigma) \cup \mathcal{L}_\sigma(\mathcal{V}, \mathcal{W})$ enabling the transition $\bar{p} \parallel \bar{r} \xrightarrow{a}_\mathcal{M} \bar{p}' \parallel \bar{r}'$. We distinguish three cases:

1. $(\bar{v} \parallel \bar{w}, a) \in (\mathcal{V} \setminus \mathcal{V}_\sigma)^\bullet$. It follows that $\bar{w} = \bullet^m$, and thus, subsystem $\mathcal{M}_\mathrm{P}$ does not participate. Hence, we have $\bar{r} = \bar{r}'$ and $(\bar{v}, a) \in \mathcal{V}$ enables a transition $\bar{p} \xrightarrow{a}_{\mathcal{M}_\Pi} \bar{p}'$. Since $\bar{p} \,\mathsf{B}\, \bar{q}$, by Definition 3.5, we have:
   - $a = \tau$ with $\bar{p}' \,\mathsf{B}\, \bar{q}$. Because $\bar{p}' \,\mathsf{B}\, \bar{q}$ and $\bar{r} = \bar{r}'$, the proof trivially follows.
   - $\bar{q} \xrightarrow{\tau}{}^*_{\mathcal{M}_{\Pi'}} \hat{\bar{q}} \xrightarrow{a}_{\mathcal{M}_{\Pi'}} \bar{q}'$ with $\bar{p} \,\mathsf{B}\, \hat{\bar{q}}$ and $\bar{p}' \,\mathsf{B}\, \bar{q}'$. These transitions are enabled by laws in $\mathcal{V}' \setminus \mathcal{V}'_\sigma$. The set of derived laws are of the form $(\bar{v}' \parallel \bullet^m, \tau) \in (\mathcal{V}' \setminus \mathcal{V}'_\sigma)^\bullet$ enabling a τ-path from $\bar{q} \parallel \bar{r}$ to $\hat{\bar{q}} \parallel \bar{r}$, and there is a law $(\bar{v}' \parallel \bullet^m, a) \in (\mathcal{V}' \setminus \mathcal{V}'_\sigma)^\bullet$ enabling $\hat{\bar{q}} \parallel \bar{r} \xrightarrow{a}_{\mathcal{M}'} \bar{q}' \parallel \bar{r}$. From this and $\bar{r}' = \bar{r}$ it follows that the proof obligation is satisfied.
2. $(\bar{v} \parallel \bar{w}, a) \in {}^\bullet(\mathcal{W} \setminus \mathcal{W}_\sigma)$. It follows that $\bar{v} = \bullet^n$, and thus, subsystems $\mathcal{M}_\Pi$ and $\mathcal{M}_{\Pi'}$ do not participate. We have $\bar{p} = \bar{p}'$ and $\bar{r} \xrightarrow{a}_{\mathcal{M}_\mathrm{P}} \bar{r}'$. We take $\bar{q}' = \bar{q}$. Hence, we can conclude $\bar{q} \parallel \bar{r} \xrightarrow{\tau}{}^*_{\mathcal{M}'} \bar{q} \parallel \bar{r} \xrightarrow{a}_\mathcal{M} \bar{q}' \parallel \bar{r}'$, $\bar{p} \,\mathsf{B}\, \bar{q}$, and $\bar{p}' \,\mathsf{B}\, \bar{q}'$.
3. $(\bar{v} \parallel \bar{w}, a) \in \mathcal{L}_\sigma(\mathcal{V}, \mathcal{W})$. Both parts of the network participate in the transition $\bar{p} \parallel \bar{r} \xrightarrow{a}_\mathcal{M} \bar{p}' \parallel \bar{r}'$. By definition of $\mathcal{L}_\sigma(\mathcal{V}, \mathcal{W})$, there are $(\bar{v}, \alpha) \in \mathcal{V}$, $(\bar{w}, \beta) \in \mathcal{W}$ and $\sigma(\alpha, \beta) = a$ such that $(\bar{v}, \alpha)$ enables a transition $\bar{p} \xrightarrow{\alpha}_{\mathcal{M}_\Pi} \bar{p}'$

and $(\bar{u}, \beta)$ enables a transition $\bar{q} \xrightarrow{\beta} \bar{q}'$. Since $\bar{p}$ B $\bar{q}$ and $\alpha \in \mathcal{A}_{\mathcal{V}} \setminus \{\tau\}$, by Definition 3.5, we must have that $\bar{q} \xrightarrow{\tau}{}^*_{\mathcal{M}'_{\Pi'}} \hat{\bar{q}} \xrightarrow{\alpha}_{\mathcal{M}'_{\Pi'}} \bar{q}'$ with $\bar{p}$ B $\hat{\bar{q}}$ and $\bar{p}'$ B $\bar{q}'$. Since τ actions are not related by the interface function, we must have a set of synchronisation laws of the form $(\bar{v}' \parallel \bullet^m, \tau) \in (\mathcal{V}' \setminus \mathcal{V}'_\sigma)^\bullet$ enabling a τ-path $\bar{q} \parallel \bar{r} \xrightarrow{\tau}{}^*_{\mathcal{M}'} \hat{\bar{q}} \parallel \bar{r}$.
Let $(\bar{v}', \alpha) \in \mathcal{V}'$ be the synchronisation law enabling the α-transition. Since $\sigma(\alpha, \beta) = a$, α is an interface action and does not occur in $\mathcal{V}' \setminus \mathcal{V}'_\sigma$. It follows that $(\bar{v}', \alpha) \in \mathcal{V}'_\sigma$, and consequently $(\bar{v}' \parallel \bar{w}, a) \in \mathcal{L}_\sigma(\mathcal{V}', \mathcal{W})$. Law $(\bar{v}' \parallel \bar{w}, a)$ enables the transition $\hat{\bar{q}} \parallel \bar{r} \xrightarrow{a}_{\mathcal{M}'} \bar{q}' \parallel \bar{r}'$, and the proof follows.

Def. 3.5, case 2. *If $\bar{s}$ C $\bar{t}$ and $\bar{t} \xrightarrow{a}_{\mathcal{M}'} \bar{t}'$ then either $a = \tau \wedge \bar{s}'$ C $\bar{t}$, or $\bar{s} \xrightarrow{\tau}{}^*_{\mathcal{M}} \hat{\bar{s}} \xrightarrow{a}_{\mathcal{M}} \bar{s}' \wedge \bar{s}$ C $\hat{\bar{t}} \wedge \bar{s}'$ C $\bar{t}'$.* This case is symmetric to the previous case.

Def. 3.5, case 3. *If $\bar{s}$ C $\bar{t}$ and there is an infinite sequence of states $(\bar{s}^k)_{k\in\omega}$ such that $\bar{s} = \bar{s}^0$, $\bar{s}^k \xrightarrow{\tau}_{\mathcal{M}} \bar{s}^{k+1}$ and $\bar{s}^k$ C $\bar{t}$ for all $k \in \omega$, then there exists a state $\bar{t}'$ such that $\bar{t} \xrightarrow{\tau}{}^+_{\mathcal{M}'} \bar{t}'$ and $\bar{s}^k$ C $\bar{t}'$ for some $k \in \omega$.* Again we reformulate the proof obligation to better distinguish between the two components, with $\bar{s} = \bar{p} \parallel \bar{r}$ and $\bar{t} = \bar{q} \parallel \bar{r}$: *if $\bar{p} \parallel \bar{r}$ C $\bar{q} \parallel \bar{r}$ and there is an infinite sequence of states $(\bar{p}^k \parallel \bar{r}^k)_{k\in\omega}$ such that $\bar{p} \parallel \bar{r} = \bar{p}^0 \parallel \bar{r}^0$, $\bar{p}^k \parallel \bar{r}^k \xrightarrow{\tau}_{\mathcal{M}} \bar{p}^{k+1} \parallel \bar{r}^{k+1}$ and $\bar{p}^k$ B $\bar{q}$ for all $k \in \omega$, then there exist states $\bar{q}'$ and $\bar{r}'$ such that $\bar{q} \parallel \bar{r} \xrightarrow{\tau}{}^+_{\mathcal{M}'} \bar{q}' \parallel \bar{r}'$ and $\bar{p}^k$ B $\bar{q}'$ for some $k \in \omega$.*
We distinguish two cases:

1. All steps in the τ-sequence are enabled in $\mathcal{M}_\Pi$, i.e., $\forall k \in \omega.\ \bar{p}^k \xrightarrow{\tau}_{\mathcal{M}_\Pi} \bar{p}^{k+1}$. Since $\bar{p}$ B $\bar{q}$, by condition 3 of Definition 3.5, it follows that there is a state $\bar{q}'$ with $\bar{q} \xrightarrow{\tau}{}^+ \bar{q}'$ and $\bar{p}^k$ B $\bar{q}'$ for some $k \in \omega$. Since τ is not an interface action, the synchronisation laws enabling $\bar{q} \xrightarrow{\tau}{}^+ \bar{q}'$ are also present in $\mathcal{M}'$. Hence, we have $\bar{q} \parallel \bar{r} \xrightarrow{\tau}{}^+ \bar{q}' \parallel \bar{r}$ and $\bar{p}^k$ B $\bar{q}'$ for $k \in \omega$.
2. There is a $k \in \omega$ with $\neg \bar{p}^k \xrightarrow{\tau}_{\mathcal{M}_\Pi} \bar{p}^{k+1}$. We do have $\bar{p}^k \parallel \bar{r}^k \xrightarrow{\tau}_{\mathcal{M}} \bar{p}^{k+1} \parallel \bar{r}^{k+1}$ with $\bar{p}^k$ B $\bar{q}$ (see the antecedent at the start of the 'divergence' case). Since the τ-transition is not enabled in $\mathcal{M}_\Pi$ the transition must be enabled by a synchronisation law $(\bar{v} \parallel \bar{w}, \tau) \in {}^\bullet(\mathcal{W} \setminus \mathcal{W}_\sigma) \cup \mathcal{L}_\sigma(\mathcal{V}, \mathcal{W})$. We distinguish two cases:
   * $(\bar{v} \parallel \bar{w}, \tau) \in {}^\bullet(\mathcal{W} \setminus \mathcal{W}_\sigma)$. The transition $\bar{p}^k \parallel \bar{r}^k \xrightarrow{\tau}_{\mathcal{M}} \bar{p}^{k+1} \parallel \bar{r}^{k+1}$ is enabled by $(\bar{v} \parallel \bar{w}, \tau) \in {}^\bullet(\mathcal{W} \setminus \mathcal{W}_\sigma)$. Therefore, there is a transition $\bar{r}^k \xrightarrow{\tau}_{\mathcal{M}_\mathrm{P}} \bar{r}^{k+1}$ enabled by $(\bar{w}, \tau) \in \mathcal{W} \setminus \mathcal{W}_\sigma$. Since this transition is part of an infinite τ-sequence, there is a path $\bar{p} \parallel \bar{r} \xrightarrow{\tau}{}^*_{\mathcal{M}} \bar{p}^k \parallel \bar{r}^k$. Furthermore, condition 1b of Definition 3.5 holds for C, hence, there is a state $\bar{q}' \in \mathcal{S}_{\mathcal{M}_{\Pi'}}$ and a transition $\bar{q} \parallel \bar{r} \xrightarrow{\tau}{}^*_{\mathcal{M}_\mathrm{P}} \bar{q}' \parallel \bar{r}^k$ with $\bar{p}^k \parallel \bar{r}^k$ C $\bar{q}' \parallel \bar{r}^k$. Therefore, we have $\bar{q} \parallel \bar{r} \xrightarrow{\tau}{}^+_{\mathcal{M}'} \bar{q}' \parallel \bar{r}^{k+1}$. Finally, since $\bar{p}^k \parallel \bar{r}^k$ C $\bar{q}' \parallel \bar{r}^k$, it follows that $\bar{p}^k$ B $\bar{q}'$.
   * $(\bar{v} \parallel \bar{w}, \tau) \in \mathcal{L}_\sigma(\mathcal{V}, \mathcal{W})$. By definition of $\mathcal{L}_\sigma(\mathcal{V}, \mathcal{W})$, there are two laws $(\bar{v}, \alpha) \in \mathcal{V}$ and $(\bar{u}, \beta) \in \mathcal{W}$ with $\sigma(\alpha, \beta) = \tau$. The laws enable transitions $\bar{p}^k \xrightarrow{\alpha}_{\mathcal{M}_\Pi} \bar{p}^{k+1}$ and $\bar{r}^k \xrightarrow{\beta}_{\mathcal{M}_\mathrm{P}} \bar{r}^{k+1}$, respectively. Since $\bar{p}^k$ B $\bar{q}$ and $\alpha \neq \tau$, by Definition 3.5, there are states $\hat{\bar{q}}, \bar{q}' \in \mathcal{S}_{\mathcal{M}_{\Pi'}}$ such that there is a sequence $\bar{q} \xrightarrow{\tau}{}^*_{\mathcal{M}_{\Pi'}} \hat{\bar{q}} \xrightarrow{\alpha}_{\mathcal{M}_{\Pi'}} \bar{q}'$ with $\bar{p}$ B $\hat{\bar{q}}$ and $\bar{p}^{k+1}$ B $\bar{q}'$. Let $(\bar{v}', \alpha) \in \mathcal{V}'$ be the law enabling the α-transition. Since $\sigma(\alpha, \beta) = \tau$, and consequently $(\bar{v}' \parallel \bar{w}, \tau) \in \sigma(\mathcal{X}', \mathcal{Y})$. Furthermore, the τ-path from $\bar{q}$ to $\hat{\bar{q}}$ is enabled by laws of the form $(\bar{v}'', \tau) \in \mathcal{V}' \setminus \mathcal{V}'_\sigma$. Hence, there is a series of transitions $\bar{q} \parallel \bar{r} \xrightarrow{\tau}{}^*_{\mathcal{M}'} \hat{\bar{q}} \parallel \bar{r}^k \xrightarrow{\tau}_{\mathcal{M}'} \bar{q}' \parallel \bar{r}^{k+1}$. Finally, recall that $\bar{p}^{k+1}$ B $\bar{q}'$. Hence, also in this case the proof obligation is satisfied.

Def. 3.5, case 4. *If $\bar{s}$ C $\bar{t}$ and there is an infinite sequence of states $(\bar{t}^k)_{k\in\omega}$ such that $\bar{t} = \bar{t}^0$, $\bar{t}^k \xrightarrow{\tau}_{\mathcal{M}'} \bar{t}^{k+1}$ and $\bar{s}$ C $\bar{t}^k$ for all $k \in \omega$, then there exists a state $\bar{s}'$ such that $\bar{s} \xrightarrow{\tau}{}^+_{\mathcal{M}} \bar{s}'$ and $\bar{s}'$ C $\bar{t}^k$ for some $k \in \omega$.* This case is symmetric to the previous case. □

5. Decomposition of LTS networks

In Section 4, we discussed the composition of LTS networks, in which a system is constructed by combining two subnetworks. However, for compositional model checking approaches, it should also be possible to correctly decompose LTS networks. In this case the inter-component laws are already known. Therefore, we can derive a set of interface laws and an interface function specifying how the system is decomposed into components.

Consider the decomposition of an LTS network $\mathcal{M} = (\Pi \parallel \mathrm{P}, \mathcal{V})$ into components $\mathcal{M}_\Pi$ and $\mathcal{M}_\mathrm{P}$ according to some interface function σ. We denote the size of Π by n and the size of P by m. First, the set of synchronisation laws $\mathcal{Z}$ is partitioned into three disjoint sets: 1) $\mathtt{ll}(\mathcal{V})$, the laws only applicable on the processes in Π (the function `ll` selects the subset of laws in $\mathcal{V}$ that have a vector in which the last m entries are $\bullet$, and); 2) $\mathtt{rl}(\mathcal{W})$, the laws only applicable on the processes in P (the function `rl` selects the subset of laws in $\mathcal{W}$ that have a vector in which the first n entries are $\bullet$); and 3) $\mathtt{il}(\mathcal{V})$, the inter-component laws (the laws not in $\mathtt{ll}(\mathcal{V})$ and $\mathtt{rl}(\mathcal{V})$).

Next, consider two functions $f, g : \mathtt{il}(\mathcal{V}) \to \mathcal{A} \setminus \{\tau\}$ from inter-component laws to interface actions. Using those functions, the inter-component laws can be decomposed into sets $\overleftarrow{\mathtt{il}(\mathcal{V})} = \{(\bar{v}, f(x)) \mid x = (\bar{v} \parallel \bar{w}, a) \in \mathtt{il}(\mathcal{V})\}$ and $\overrightarrow{\mathtt{il}(\mathcal{V})} = \{(\bar{w}, g(x)) \mid x = (\bar{v} \parallel \bar{w}, a) \in \mathtt{il}(\mathcal{V})\}$ of interface laws over Π and P, respectively. In a similar way, $\mathtt{ll}(\mathcal{V})$ and $\mathtt{rl}(\mathcal{V})$ can be decomposed into sets $\overleftarrow{\mathtt{ll}(\mathcal{V})}$ and $\overrightarrow{\mathtt{ll}(\mathcal{V})}$, and $\overleftarrow{\mathtt{rl}(\mathcal{V})}$ and $\overrightarrow{\mathtt{rl}(\mathcal{V})}$, respectively, with the vectors of $\overrightarrow{\mathtt{ll}(\mathcal{V})}$ and $\overleftarrow{\mathtt{rl}(\mathcal{V})}$ only consisting of $\bullet$ entries.

Finally, the components are defined as $\mathcal{M}_\Pi = (\Pi, \overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})})$ and $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})})$.

To be able to apply Proposition 4.2 for compositional state space construction, the composition of the decomposed LTS networks must be equivalent to the original system. If this holds we say a decomposition is *consistent* with respect to $\mathcal{M}$.

Definition 5.1 *(Consistent decomposition).* Consider an LTS network $\mathcal{M} = (\Pi \parallel \mathrm{P}, \mathcal{V})$. Say LTS network $\mathcal{M}$ is decomposed into components $\mathcal{M}_\Pi = (\Pi, \overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})})$ and $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})})$ according to some interface function σ. Such a decomposition of $\mathcal{M}$ into components $\mathcal{M}_\Pi$ and $\mathcal{M}_\mathrm{P}$ is called *consistent* with respect to $\mathcal{M}$ iff $\mathcal{M} = \mathcal{M}_\Pi \parallel_\sigma \mathcal{M}_\mathrm{P}$.

To show that a decomposition is consistent with the original system it is sufficient to show that the set of inter-component laws of the original system is equivalent to the set of inter-component laws generated by the interface function:

Lemma 5.1. *Consider an LTS network $\mathcal{M} = (\Pi \parallel \mathrm{P}, \mathcal{V})$. A* consistent *decomposition of $\mathcal{M}$ into components $\mathcal{M}_\Pi = (\Pi, \overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})})$ and $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})})$ with interface function $\sigma = \{(f(\bar{v}, a), g(\bar{v}, a), a) \mid (\bar{v}, a) \in \mathtt{il}(\mathcal{V})\}$ is guaranteed if $\mathtt{il}(\mathcal{V}) = \mathcal{L}_\sigma(\overleftarrow{\mathtt{il}(\mathcal{V})}, \overrightarrow{\mathtt{il}(\mathcal{V})})$, $\mathcal{A}_{\overleftarrow{\mathtt{ll}(\mathcal{V})}} \cap \mathcal{A}_{\overleftarrow{\mathtt{il}(\mathcal{V})}} = \emptyset$, and $\mathcal{A}_{\overrightarrow{\mathtt{rl}(\mathcal{V})}} \cap \mathcal{A}_{\overrightarrow{\mathtt{il}(\mathcal{V})}} = \emptyset$.*

Proof. The decomposition of $\mathcal{M} = (\Pi \parallel \mathrm{P}, \mathcal{V})$ into components $\mathcal{M}_\Pi = (\Pi, \overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})})$ and $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})})$ is consistent iff $\mathcal{M} = \mathcal{M}_\Pi \parallel_\sigma \mathcal{M}_\mathrm{P}$, which is the case iff $\mathcal{V} = \overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})} \parallel \overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})}$ (Definition 4.2). This means that we must have $\mathcal{V} = ((\overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})}) \setminus (\overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})})_\sigma)^\bullet \cup {}^\bullet((\overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})}) \setminus (\overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})})_\sigma) \cup \mathcal{L}_\sigma(\overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})}, \overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})})$. Before we prove that this holds, let us number the antecedent propositions of the lemma: $\mathtt{il}(\mathcal{V}) = \mathcal{L}_\sigma(\overleftarrow{\mathtt{il}(\mathcal{V})}, \overrightarrow{\mathtt{il}(\mathcal{V})})$ (1), $\mathcal{A}_{\overleftarrow{\mathtt{ll}(\mathcal{V})}} \cap \mathcal{A}_{\overleftarrow{\mathtt{il}(\mathcal{V})}} = \emptyset$ (2), and $\mathcal{A}_{\overrightarrow{\mathtt{rl}(\mathcal{V})}} \cap \mathcal{A}_{\overrightarrow{\mathtt{il}(\mathcal{V})}} = \emptyset$ (3). As $\mathcal{V}$ consists of three disjoint sets $\mathtt{ll}(\mathcal{V})$, $\mathtt{rl}(\mathcal{V})$ and $\mathtt{il}(\mathcal{V})$, we show that $\mathtt{ll}(\mathcal{V}) = ((\overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})}) \setminus (\overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})})_\sigma)^\bullet$, $\mathtt{rl}(\mathcal{V}) = {}^\bullet((\overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})}) \setminus (\overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})})_\sigma)$, and $\mathtt{il}(\mathcal{V}) = \mathcal{L}_\sigma(\overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})}, \overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})})$.

By construction of $\overleftarrow{\mathtt{il}(\mathcal{V})}$ and the definition of $\mathcal{V}_\sigma$ and $\mathcal{W}_\sigma$ (see Section 4), we have $\mathcal{A}_{\overleftarrow{\mathtt{il}(\mathcal{V})}} = \mathcal{A}_{(\overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})})_\sigma}$ and $\mathcal{A}_{\overrightarrow{\mathtt{il}(\mathcal{V})}} = \mathcal{A}_{(\overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})})_\sigma}$ (4). Furthermore, from (2) and (3) it follows that $\overleftarrow{\mathtt{ll}(\mathcal{V})}$ and $\overrightarrow{\mathtt{rl}(\mathcal{V})}$ are disjoint from $\overleftarrow{\mathtt{il}(\mathcal{V})}$ and $\overrightarrow{\mathtt{il}(\mathcal{V})}$, respectively. Thus, $\overleftarrow{\mathtt{ll}(\mathcal{V})}$ and $\overrightarrow{\mathtt{rl}(\mathcal{V})}$ are disjoint from $(\overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})})_\sigma$ and $(\overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})})_\sigma$ (5), respectively, implying that $\overleftarrow{\mathtt{il}(\mathcal{V})} = (\overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})})_\sigma$ and $\overrightarrow{\mathtt{il}(\mathcal{V})} = (\overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})})_\sigma$ (6). It follows that $\mathtt{ll}(\mathcal{V}) \overset{(5,6)}{=} ((\overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})}) \setminus \overleftarrow{\mathtt{il}(\mathcal{V})})^\bullet \overset{(6)}{=} ((\overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})}) \setminus (\overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})})_\sigma)^\bullet$ and, symmetrically, $\mathtt{rl}(\mathcal{V}) \overset{(5,6)}{=} {}^\bullet((\overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})}) \setminus (\overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})})_\sigma)$.

Recall that $\overleftarrow{\mathtt{ll}(\mathcal{V})}$ and $\overrightarrow{\mathtt{rl}(\mathcal{V})}$ do not have any result actions in common with $\overleftarrow{\mathtt{il}(\mathcal{V})}$ and $\overrightarrow{\mathtt{il}(\mathcal{V})}$, respectively (2,3), and that interface actions defined by σ are produced by the same functions f and g that are used to produce the result actions of sets $\overleftarrow{\mathtt{il}(\mathcal{V})}$ and $\overrightarrow{\mathtt{il}(\mathcal{V})}$, respectively. These two facts and Definition 4.2 (synchronisation via σ) imply that $\mathcal{L}_\sigma(\overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})}, \overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})}) \overset{(5,6,\sigma)}{=} \mathcal{L}_\sigma(\overleftarrow{\mathtt{il}(\mathcal{V})}, \overrightarrow{\mathtt{il}(\mathcal{V})}) \overset{(1)}{=} \mathtt{il}(\mathcal{V})$. Hence, the decomposition of $\mathcal{M}$ is consistent if $\mathtt{il}(\mathcal{V}) = \mathcal{L}_\sigma(\overleftarrow{\mathtt{il}(\mathcal{V})}, \overrightarrow{\mathtt{il}(\mathcal{V})})$ (1), $\mathcal{A}_{\overleftarrow{\mathtt{ll}(\mathcal{V})}} \cap \mathcal{A}_{\overleftarrow{\mathtt{il}(\mathcal{V})}} = \emptyset$ (2), and $\mathcal{A}_{\overrightarrow{\mathtt{rl}(\mathcal{V})}} \cap \mathcal{A}_{\overrightarrow{\mathtt{il}(\mathcal{V})}} = \emptyset$ (3). □

Indeed, it is possible to derive an inconsistent decomposition as shown in Example 5.1.

Example 5.1 *(Inconsistent decomposition).* For an LTS network $\mathcal{M} = (\Pi \parallel \mathrm{P}, \mathcal{V})$, consider a set of inter-component laws $\mathtt{il}(\mathcal{V}) = \{(\langle a, b\rangle, c), (\langle b, a\rangle, c)\}$. To generate interface actions for a decomposition into LTS networks $\mathcal{M}_\Pi = (\Pi, \overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})})$ and $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})})$, consider the functions f, g that relate inter-component laws with interface actions solely based on the result action of the input law, i.e., $\forall(\bar{v}, a), (\bar{w}, b) \in \mathtt{il}(\mathcal{V}).\, a = b \iff f((\bar{v}, a)) = f((\bar{w}, b)) \vee g((\bar{v}, a)) = g((\bar{w}, b))$. We define that $f((a, b)) = f((b, a)) = \alpha$ and $g((a, b)) = g((b, a)) = \beta$, with $\alpha, \beta \in \mathcal{A} \setminus \{\tau\}$. Partitioning the laws $\mathtt{il}(\mathcal{V})$ results in the sets of interface laws $\overleftarrow{\mathtt{il}(\mathcal{V})} = \{(\langle a\rangle, \alpha), (\langle b\rangle, \alpha)\}$ and $\overrightarrow{\mathtt{il}(\mathcal{V})} = \{(\langle b\rangle, \beta), (\langle a\rangle, \beta)\}$. This system implies the interface function $\sigma(\alpha, \beta) = c$. The derived set of inter-component laws is then $\mathcal{L}_\sigma(\overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})}, \overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})}) = \{(\langle a, a\rangle, c), (\langle a, b\rangle, c), (\langle b, a\rangle, c), (\langle b, b\rangle, c)\} \neq \mathtt{il}(\mathcal{V})$. Hence, this decomposition is not consistent with the original system.

However, a consistent decomposition can *always* be derived. Propositions 5.1 and 5.2 give functions f and g that guarantee a consistent decomposition. Consider a synchronisation law $(\bar{v} \parallel \bar{w}, a)$. The idea is to encode this synchronisation law directly in the interface function, by making sure that f and g are injections: for no two different interface vectors $\bar{v}$, $\bar{v}'$, we have $f(\bar{v}) = f(\bar{v}')$, and for no two different interface vectors $\bar{w}$, $\bar{w}'$, we have $g(\bar{w}) = g(\bar{w}')$. This way it is explicit which interface law corresponds to which inter-component law.

In Proposition 5.1, for each synchronisation law $(\bar{v} \parallel \bar{w}, a) \in \mathtt{il}(\mathcal{V})$, we create unique interface actions $\alpha_{\bar{v}}$ and $\alpha_{\bar{w}}$ and define that $\sigma(\alpha_{\bar{v}}, \alpha_{\bar{w}}) = a$.

Proposition 5.1. *Consider an LTS network $\mathcal{M} = (\Pi \parallel \mathrm{P}, \mathcal{V})$. To decompose $\mathcal{M}$ into components $\mathcal{M}_\Pi = (\Pi, \overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})})$ and $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})})$, first, we partition $\mathcal{V}$ into $\mathtt{ll}(\mathcal{V})$, $\mathtt{rl}(\mathcal{V})$ and $\mathtt{il}(\mathcal{V})$. For all $(\bar{v} \parallel \bar{w}, a) \in \mathtt{il}(\mathcal{V})$, we define the functions*

producing interface actions as $f((\bar{v} \parallel \bar{w}, a)) = \alpha_{\bar{v}}$ *and* $g((\bar{v} \parallel \bar{w}, a)) = \alpha_{\bar{w}}$*, where* $\alpha_{\bar{v}} \notin \mathcal{A}_{\overleftarrow{\texttt{ll}(\mathcal{V})}} \cup \{\tau\}$ *and* $\alpha_{\bar{w}} \notin \mathcal{A}_{\overrightarrow{\texttt{rl}(\mathcal{V})}} \cup \{\tau\}$ *are unique interface actions identified by the corresponding interface law, that is,* $\forall (\bar{v} \parallel \bar{w}, a), (\bar{v}' \parallel \bar{w}', b) \in \texttt{il}(\mathcal{V})$. $\bar{v} = \bar{v}' \iff \alpha_{\bar{v}} = \alpha_{\bar{v}'}$ *and* $\forall (\bar{v} \parallel \bar{w}, a), (\bar{v}' \parallel \bar{w}', b) \in \texttt{il}(\mathcal{V})$. $\bar{w} = \bar{w}' \iff \alpha_{\bar{w}} = \alpha_{\bar{w}'}$*. The decomposition of* $\mathcal{M}$ *into* $\mathcal{M}_\Pi = (\Pi, \overleftarrow{\texttt{ll}(\mathcal{V})} \cup \overleftarrow{\texttt{il}(\mathcal{V})})$ *and* $\mathcal{M}_P = (P, \overrightarrow{\texttt{rl}(\mathcal{V})} \cup \overrightarrow{\texttt{il}(\mathcal{V})})$ *given by* f *and* g *is* consistent*.*

Proof. Functions f and g imply the interface function $\sigma = \{(\alpha_{\bar{v}}, \alpha_{\bar{w}}, a) \mid (\bar{v} \parallel \bar{w}, a) \in \texttt{il}(\mathcal{V})\}$, and sets of interface laws $\overleftarrow{\texttt{il}(\mathcal{V})} = \{(\bar{v}, \alpha_{\bar{v}}) \mid (\bar{v} \parallel \bar{w}, a) \in \texttt{il}(\mathcal{V})\}$ and $\overrightarrow{\texttt{il}(\mathcal{V})} = \{(\bar{w}, \alpha_{\bar{w}}) \mid (\bar{v} \parallel \bar{w}, a) \in \texttt{il}(\mathcal{V})\}$.

By Lemma 5.1, we have to show:

- $\texttt{il}(\mathcal{V}) = \mathcal{L}_\sigma(\overleftarrow{\texttt{il}(\mathcal{V})}, \overrightarrow{\texttt{il}(\mathcal{V})})$: By (1) the definition of $\mathcal{L}_\sigma$, (2) $(\overleftarrow{\texttt{ll}(\mathcal{V})} \cup \overleftarrow{\texttt{il}(\mathcal{V})})_\sigma = \overleftarrow{\texttt{il}(\mathcal{V})}_\sigma$, (3) $(\overrightarrow{\texttt{rl}(\mathcal{V})} \cup \overrightarrow{\texttt{il}(\mathcal{V})})_\sigma = \overrightarrow{\texttt{il}(\mathcal{V})}_\sigma$, and (4) the construction of $\overleftarrow{\texttt{il}(\mathcal{V})}$, $\overrightarrow{\texttt{il}(\mathcal{V})}$, and σ, it follows that $\mathcal{L}_\sigma(\overleftarrow{\texttt{il}(\mathcal{V})}, \overrightarrow{\texttt{il}(\mathcal{V})}) \overset{(1)}{=} \{(\bar{v} \parallel \bar{w}, a) \mid (\bar{v}, \alpha_{\bar{v}}) \in (\overleftarrow{\texttt{ll}(\mathcal{V})} \cup \overleftarrow{\texttt{il}(\mathcal{V})})_\sigma \wedge (\bar{w}, \alpha_{\bar{w}}) \in (\overrightarrow{\texttt{rl}(\mathcal{V})} \cup \overrightarrow{\texttt{il}(\mathcal{V})})_\sigma \wedge \sigma(\alpha_{\bar{v}}, \alpha_{\bar{w}}) = a\} \overset{(2,3)}{=} \{(\bar{v} \parallel \bar{w}, a) \mid (\bar{v}, \alpha_{\bar{v}}) \in \overleftarrow{\texttt{il}(\mathcal{V})}_\sigma \wedge (\bar{w}, \alpha_{\bar{w}}) \in \overrightarrow{\texttt{il}(\mathcal{V})}_\sigma \wedge \sigma(\alpha_{\bar{v}}, \alpha_{\bar{w}}) = a\} \overset{(4)}{=} \{(\bar{v} \parallel \bar{w}, a) \mid (\bar{v} \parallel \bar{w}, a) \in \texttt{il}(\mathcal{V})\} = \texttt{il}(\mathcal{V})$.
- $\mathcal{A}_{\overleftarrow{\texttt{ll}(\mathcal{V})}} \cap \mathcal{A}_{\overleftarrow{\texttt{il}(\mathcal{V})}} = \emptyset$: Since for all $\alpha_{\bar{v}} \in \mathcal{A}_{\overleftarrow{\texttt{il}(\mathcal{V})}}$, we have $\alpha_{\bar{v}} \notin \mathcal{A}_{\overleftarrow{\texttt{ll}(\mathcal{V})}} \cup \{\tau\}$, $\mathcal{A}_{\overleftarrow{\texttt{ll}(\mathcal{V})}}$ and $\mathcal{A}_{\overleftarrow{\texttt{il}(\mathcal{V})}}$ are disjoint.
- $\mathcal{A}_{\overrightarrow{\texttt{rl}(\mathcal{V})}} \cap \mathcal{A}_{\overrightarrow{\texttt{il}(\mathcal{V})}} = \emptyset$: Since for all $\alpha_{\bar{w}} \in \mathcal{A}_{\overrightarrow{\texttt{il}(\mathcal{V})}}$, we have $\alpha_{\bar{w}} \notin \mathcal{A}_{\overrightarrow{\texttt{rl}(\mathcal{V})}} \cup \{\tau\}$, $\mathcal{A}_{\overrightarrow{\texttt{rl}(\mathcal{V})}}$ and $\mathcal{A}_{\overrightarrow{\texttt{il}(\mathcal{V})}}$ are disjoint. □

Example 5.2. Consider an admissible LTS network with the following synchronisation laws:

$$\mathcal{V} = \{(\langle a, \bullet, a\rangle, a), (\langle a, a, \bullet\rangle, a), (\langle b, b, b\rangle, \tau), (\langle c, \bullet, c\rangle, c)\}$$

If we want to decompose the associated LTS network into a component containing the first two processes and a component containing the third process, we first partition $\mathcal{V}$ into $\texttt{ll}(\mathcal{V}) = \{(\langle a, a, \bullet\rangle, a)\}$, $\texttt{rl}(\mathcal{V}) = \emptyset$, and $\texttt{il}(\mathcal{V}) = \{(\langle a, \bullet, a\rangle, a), (\langle b, b, b\rangle, \tau), (\langle c, \bullet, c\rangle, c)\}$. From these sets, along the lines of Proposition 5.1, we can derive the following sets of synchronisation laws:

$$\begin{aligned}
&\overleftarrow{\texttt{ll}(\mathcal{V})} = \{(\langle a, a\rangle, a)\}\\
&\overrightarrow{\texttt{rl}(\mathcal{V})} = \emptyset\\
&\overleftarrow{\texttt{il}(\mathcal{V})} = \{(\langle a, \bullet\rangle, \alpha_{\langle a,\bullet\rangle}), (\langle b, b\rangle, \alpha_{\langle b,b\rangle}), (\langle c, \bullet\rangle, \alpha_{\langle c,\bullet\rangle})\}\\
&\overrightarrow{\texttt{il}(\mathcal{V})} = \{(\langle a\rangle, \alpha_{\langle a\rangle}), (\langle b\rangle, \alpha_{\langle b\rangle}), (\langle c\rangle, \alpha_{\langle c\rangle})\}\\
&\sigma = \{(\alpha_{\langle a,\bullet\rangle}, \alpha_{\langle a\rangle}, a), (\alpha_{\langle b,b\rangle}, \alpha_{\langle b\rangle}, \tau), (\alpha_{\langle c,\bullet\rangle}, \alpha_{\langle c\rangle}, c)\}
\end{aligned}$$

Proposition 5.2 proposes an alternative decomposition that is implemented in CADP's smart reduction [15]. The idea is (1) to generate only interface synchronisation laws of the form (a, a, b), so that components always synchronise through a common label a, while (2) keeping a equal to b whenever possible, in which cases we avoid the introduction of new interface actions. Laws in this simple form make the decomposition more straightforward.

Proposition 5.2. *Consider an LTS network* $\mathcal{M} = (\Pi \parallel P, \mathcal{V})$*. To decompose* $\mathcal{M}$ *into components* $\mathcal{M}_\Pi = (\Pi, \overleftarrow{\texttt{ll}(\mathcal{V})} \cup \overleftarrow{\texttt{il}(\mathcal{V})})$ *and* $\mathcal{M}_P = (P, \overrightarrow{\texttt{rl}(\mathcal{V})} \cup \overrightarrow{\texttt{il}(\mathcal{V})})$*, we can partition* $\mathcal{V}$ *into* $\texttt{ll}(\mathcal{V})$*,* $\texttt{rl}(\mathcal{V})$ *and* $\texttt{il}(\mathcal{V})$*. For all* $(\bar{v}, a) \in \texttt{il}(\mathcal{V})$*, we define the functions* f*,* g *producing interface actions as*

$$f(\bar{v}, a) = g(\bar{v}, a) = \begin{cases} a & \text{if } \texttt{visible-unique}(a)\\ \alpha_{(\bar{v},a)} & \text{otherwise}\end{cases}$$

where each $\alpha_{(\bar{v},a)} \notin \mathcal{A}_{\overleftarrow{\texttt{ll}(\mathcal{V})}} \cup \mathcal{A}_{\overrightarrow{\texttt{rl}(\mathcal{V})}} \cup \{\tau\}$ *is a unique interface action identified by the corresponding inter-component synchronisation law, that is,* $\forall (\bar{v}, a), (\bar{w}, b) \in \texttt{il}(\mathcal{V})$. $\bar{v} = \bar{w} \iff \alpha_{\bar{v}} = \alpha_{\bar{w}}$*, and where* $\texttt{visible-unique}(a)$ *is defined as the following predicate:*

$$\texttt{visible-unique}(a) \triangleq a \neq \tau \wedge \forall (\bar{v}, a), (\bar{w}, a) \in \texttt{il}(\mathcal{V}).\ \bar{v} = \bar{w}$$

The decomposition of $\mathcal{M}$ *into* $\mathcal{M}_\Pi = (\Pi, \overleftarrow{\texttt{ll}(\mathcal{V})} \cup \overleftarrow{\texttt{il}(\mathcal{V})})$ *and* $\mathcal{M}_P = (P, \overrightarrow{\texttt{rl}(\mathcal{V})} \cup \overrightarrow{\texttt{il}(\mathcal{V})})$ *using* f *and* g *is* consistent*.*

The proof of Proposition 5.2 is similar to the proof of Proposition 5.1. The most relevant difference is the fact that $\sigma(a, a) = a$ if $(\bar{v} \parallel \bar{w}, a) \in \texttt{il}(\mathcal{V})$ and a is unique in $\texttt{il}(\mathcal{V})$, i.e., $\texttt{visible-unique}(a)$ holds. In this case we must also have $\mathcal{A}_{\overleftarrow{\texttt{ll}(\mathcal{V})}} \cap \mathcal{A}_{\overleftarrow{\texttt{il}(\mathcal{V})}} = \emptyset$ and $\mathcal{A}_{\overrightarrow{\texttt{rl}(\mathcal{V})}} \cap \mathcal{A}_{\overrightarrow{\texttt{il}(\mathcal{V})}} = \emptyset$, otherwise a contradiction with $\texttt{visible-unique}(a)$ can be derived.

The decomposition of Proposition 5.2 implies the interface function:

$$\sigma = \{(\alpha_{\bar{v}\parallel\bar{w}}, \alpha_{\bar{v}\parallel\bar{w}}, a) \mid (\bar{v} \parallel \bar{w}, a) \in \mathtt{il}(\mathcal{V}) \wedge \neg\mathtt{visible\text{-}unique}(a)\}$$
$$\cup \{(a, a, a) \mid (\bar{v} \parallel \bar{w}, a) \in \mathtt{il}(\mathcal{V}) \wedge \mathtt{visible\text{-}unique}(a)\}$$

Furthermore, it implies the following sets of interface synchronisation laws:

$$\overleftarrow{\mathtt{il}(\mathcal{V})} = \{(\bar{v}, \alpha_{\bar{v}\parallel\bar{w}}) \mid (\bar{v} \parallel \bar{w}, a) \in \mathtt{il}(\mathcal{V}) \wedge \neg\mathtt{visible\text{-}unique}(a)\}$$
$$\cup \{(\bar{v}, a) \mid (\bar{v} \parallel \bar{w}, a) \in \mathtt{il}(\mathcal{V}) \wedge \mathtt{visible\text{-}unique}(a)\}$$

$$\overrightarrow{\mathtt{il}(\mathcal{V})} = \{(\bar{w}, \alpha_{\bar{v}\parallel\bar{w}}) \mid (\bar{v} \parallel \bar{w}, a) \in \mathtt{il}(\mathcal{V}) \wedge \neg\mathtt{visible\text{-}unique}(a)\}$$
$$\cup \{(\bar{w}, a) \mid (\bar{v} \parallel \bar{w}, a) \in \mathtt{il}(\mathcal{V}) \wedge \mathtt{visible\text{-}unique}(a)\}$$

Example 5.3. Consider again the admissible LTS network of Example 5.2 and its intended decomposition. Its set of synchronisation laws $\mathcal{V}$ can be decomposed along the lines of Proposition 5.2 as follows, using the same definitions of $\mathtt{ll}(\mathcal{V})$, $\mathtt{rl}(\mathcal{V})$ and $\mathtt{il}(\mathcal{V})$:

$$\overleftarrow{\mathtt{il}(\mathcal{V})} = \{(\langle a, \bullet\rangle, \alpha_{\langle a,\bullet,a\rangle}), (\langle b, b\rangle, \alpha_{\langle b,b,b\rangle}), (\langle c, \bullet\rangle, c)\}$$
$$\overrightarrow{\mathtt{il}(\mathcal{V})} = \{(\langle a\rangle, \alpha_{\langle a,\bullet,a\rangle}), (\langle b\rangle, \alpha_{\langle b,b,b\rangle}), (\langle c\rangle, c)\}$$
$$\sigma = \{(\alpha_{\langle a,\bullet,a\rangle}, \alpha_{\langle a,\bullet,a\rangle}, a), (\alpha_{\langle b,b,b\rangle}, \alpha_{\langle b,b,b\rangle}, \tau), (c, c, c)\}$$

Preservation of Admissibility Proposition 5.3 shows that LTS networks resulting from the consistent decomposition of an admissible LTS network are also admissible. Hence, consistent decomposition is compatible with the compositional verification approaches presented in [23].

Proposition 5.3. *Consider an admissible LTS network $\mathcal{M} = (\Pi \parallel \mathrm{P}, \mathcal{V})$, with Π of size n and P of size m. If the decomposition (Proposition 5.1 and Proposition 5.2) into components $\mathcal{M}_\Pi = (\Pi, \overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})})$ and $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})})$ is consistent, then $\mathcal{M}_\Pi$ and $\mathcal{M}_\mathrm{P}$ are also admissible.*

Proof. We show that $\mathcal{M}_\Pi$ satisfies Definition 3.4. The proof for $\mathcal{M}_\mathrm{P}$ is similar.
No synchronisation and renaming of τ's. Let $(\bar{v}, a) \in \overleftarrow{\mathtt{ll}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})}$ be a synchronisation law such that $\bar{v}_i = \tau$ for some $i \in 1..n$. We distinguish two cases:

- $(\bar{v}, a) \in \overleftarrow{\mathtt{il}(\mathcal{V})}$. Since $(\bar{v}, a)$ is an interface law and the decomposition is consistent, its result action a may not be τ. However, since $\mathcal{M}$ is admissible, no renaming of τ's is allowed. By contradiction it follows that $(\bar{v}, a) \notin \overleftarrow{\mathtt{il}(\mathcal{V})}$, completing this case.
- $(\bar{v}, a) \in \overleftarrow{\mathtt{ll}(\mathcal{V})}$. By construction of $\overleftarrow{\mathtt{ll}(\mathcal{V})}$, there exists a law $(\bar{v} \parallel \bullet^m, a) \in \mathtt{ll}(\mathcal{V})$. Since $\mathtt{ll}(\mathcal{V}) \subseteq \mathcal{V}$, by admissibility of $\mathcal{M}$, we have $\forall j \in 1..n.\ \bar{v}_j \neq \bullet \implies i = j$ (no synchronisation of τ's) and $a = \tau$ (no renaming of τ's).

No cutting of τ's. Let Π_i be a process with $i \in 1..n$ such that $\tau \in \mathcal{A}_{\Pi_i}$. Since $\mathcal{M}$ is admissible there exists a law $(\bar{v} \parallel \bar{w}, a) \in \mathtt{ll}(\mathcal{V}) \cup \mathtt{rl}(\mathcal{V}) \cup \mathtt{il}(\mathcal{V})$ such that $(\bar{v} \parallel \bar{u})_i = \tau$. We distinguish three cases:

- $(\bar{v} \parallel \bar{w}, a) \in \mathtt{ll}(\mathcal{V})$. Since $(\bar{v} \parallel \bar{w})_i = \tau$ and $i \leq n$ it follows that $\bar{v}_i = \tau$. By construction of $\overleftarrow{\mathtt{ll}(\mathcal{V})}$, there is a $(\bar{v}, a) \in \overleftarrow{\mathtt{ll}(\mathcal{V})}$ with $\bar{v}_i = \tau$.
- $(\bar{v} \parallel \bar{w}, a) \in \mathtt{rl}(\mathcal{V})$. In this case we must have $i > n$ which contradicts our assumption that $i \in 1..n$. The proof follows by contradiction.
- $(\bar{v} \parallel \bar{w}, a) \in \mathtt{il}(\mathcal{V})$. Then, $(\bar{v} \parallel \bar{w}, a)$ is an inter-component law with at least one participating process for each component. Hence, there exists a $j \in (n+1)..m$ such that $(\bar{v} \parallel \bar{w})_j \neq \bullet$. Moreover, since $\mathcal{M}$ is admissible, no synchronisation of τ's is allowed. Therefore, since $(\bar{v} \parallel \bar{w})_j \neq \bullet$, we must have $j = i$. However, this would mean $j \in 1..n$, contradicting $j \in (n+1)..m$. By contradiction the proof follows.

We conclude that $\mathcal{M}_\Pi$ does not cut τ's.
All three admissibility properties hold for $\mathcal{M}_\Pi$ and $\mathcal{M}_\mathrm{P}$. Hence, the LTS networks resulting from the decomposition satisfy Definition 3.4. □

6. Associative and commutative LTS network composition

In this section we create an instance of the composition operator that is commutative and associative. This operator uses an interface function that synchronises the actions of components that they have in common, i.e., their common alphabet.

It is desirable for a parallel composition operator to be both associative and commutative when used for partial model checking or compositional state space construction as then the composition order is irrelevant with respect to the resulting LTS. It allows users to select a composition order. In practice, often, the chosen composition order has a big impact on the effectiveness of compositional state space construction [27,37]. Hence, synchronisation on the common alphabet of components is frequently used in practice (for instance, see [24–26]).

Definition 6.1 *(Composition with synchronisation on the common alphabet).* Consider the LTS networks $\mathcal{M}_\Pi = (\Pi, \mathcal{V})$ of size n and $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \mathcal{W})$ of size m. The *composition with synchronisation on the common alphabet* of $\mathcal{M}_\Pi$ and $\mathcal{M}_\mathrm{P}$ is defined as $\mathcal{M}_\Pi \parallel_\cap \mathcal{M}_\mathrm{P} = \mathcal{M}_\Pi \parallel_\sigma \mathcal{M}_\mathrm{P}$, with $\sigma = \{(a, a, a) \mid a \in (\mathcal{A}_\mathcal{V} \cap \mathcal{A}_\mathcal{W}) \setminus \{\tau\}\}$.

Associativity The intuition behind the associativity of LTS network composition is that vector concatenation is associative and synchronisation on the common alphabet is insensitive to the order of composition. Thus, the concatenation of process vectors and synchronisation vectors enjoy the associativity property. The challenge, however, is to show that the $^\bullet$ operations and σ support the mathematical properties needed for associativity of the composition of sets of synchronisation laws.

Given two LTS networks $\mathcal{M}_\Pi = (\Pi, \mathcal{V})$ and $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \mathcal{W})$, the composition of the set of synchronisation laws $\mathcal{V} \parallel \mathcal{W}$ consists of the union of three sets: two describing independent behaviour of each of the components, i.e., $(\mathcal{V} \setminus \mathcal{V}_\sigma)^\bullet$ and $^\bullet(\mathcal{W} \setminus \mathcal{W}_\sigma)$, and one describing synchronising behaviour, i.e., $\mathcal{L}_\sigma(\mathcal{V}, \mathcal{W})$ (Definition 4.2). When three LTS networks are composed one (inner) composition is performed before the other. The outer composition applies the $^\bullet$ operators and σ on a union of sets. We show how these operators distribute over set union.

Lemma 6.1. *Consider sets of synchronisation laws $\mathcal{V}$ and $\mathcal{W}$ with synchronisation vectors of the same size. Padding of $\bullet$'s distributes over set union:*

$$^\bullet(\mathcal{V} \cup \mathcal{W}) = {}^\bullet\mathcal{V} \cup {}^\bullet\mathcal{W}, \text{ and } (\mathcal{V} \cup \mathcal{W})^\bullet = \mathcal{V}^\bullet \cup \mathcal{W}^\bullet.$$

Proof. The proof that $(\mathcal{V} \cup \mathcal{W})^\bullet = \mathcal{V}^\bullet \cup \mathcal{W}^\bullet$ is analog to the proof that $^\bullet(\mathcal{V} \cup \mathcal{W}) = {}^\bullet\mathcal{V} \cup {}^\bullet\mathcal{W}$. We only prove $^\bullet(\mathcal{V} \cup \mathcal{W}) = {}^\bullet\mathcal{V} \cup {}^\bullet\mathcal{W}$ here. The proof follows from (1) application of the definition of $^\bullet$ and (2) splitting of the set $\mathcal{V} \cup \mathcal{W}$ into $\mathcal{V}$ and $\mathcal{W}$:

$$\begin{aligned}
^\bullet(\mathcal{V} \cup \mathcal{W}) &\overset{(1)}{=} \{(\bullet^n \parallel \bar{v}, a) \mid (\bar{v}, a) \in \mathcal{V} \cup \mathcal{W}\} \\
&\overset{(2)}{=} \{(\bullet^n \parallel \bar{v}, a) \mid (\bar{v}, a) \in \mathcal{V}\} \cup \{(\bullet^n \parallel \bar{v}, a) \mid (\bar{v}, a) \in \mathcal{W}\} \\
&\overset{(1)}{=} {}^\bullet\mathcal{V} \cup {}^\bullet\mathcal{W} \quad \square
\end{aligned}$$

Lemma 6.2. *Consider an interface function σ and sets of synchronisation laws $\mathcal{V}$, $\mathcal{W}$, and $\mathcal{X}$. Application of $\mathcal{L}_\sigma$ distributes over set union as follows:*

$$\mathcal{L}_\sigma(\mathcal{V}, \mathcal{W} \cup \mathcal{X}) = \mathcal{L}_\sigma(\mathcal{V}, \mathcal{W}) \cup \mathcal{L}_\sigma(\mathcal{V}, \mathcal{X})$$

Proof. The proof follows from (1) application of the definition of $\mathcal{L}_\sigma$ and (2) splitting of the set $\mathcal{W} \cup \mathcal{X}$ into $\mathcal{W}$ and $\mathcal{X}$:

$$\begin{aligned}
\mathcal{L}_\sigma(\mathcal{V}, \mathcal{W} \cup \mathcal{X}) &\overset{(1)}{=} \{(\bar{v} \parallel \bar{w}, a) \mid (\bar{v}, \alpha) \in \mathcal{V} \wedge (\bar{w}, \beta) \in \mathcal{W} \cup \mathcal{X} \wedge \sigma(\alpha, \beta) = a\} \\
&\overset{(2)}{=} \{(\bar{v} \parallel \bar{w}, a) \mid (\bar{v}, \alpha) \in \mathcal{V} \wedge (\bar{w}, \beta) \in \mathcal{W} \wedge \sigma(\alpha, \beta) = a\} \cup \\
&\quad \{(\bar{v} \parallel \bar{w}, a) \mid (\bar{v}, \alpha) \in \mathcal{V} \wedge (\bar{w}, \beta) \in \mathcal{X} \wedge \sigma(\alpha, \beta) = a\} \\
&\overset{(1)}{=} \mathcal{L}_\sigma(\mathcal{V}, \mathcal{W}) \cup \mathcal{L}_\sigma(\mathcal{V}, \mathcal{X}) \quad \square
\end{aligned}$$

Next, we prove that the composition of LTS networks with synchronisation on the common alphabet is associative.

Proposition 6.1. *For all LTS networks $\mathcal{M}_\Pi = (\Pi, \mathcal{V})$, $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \mathcal{W})$, and $\mathcal{M}_\Sigma = (\Sigma, \mathcal{X})$ of sizes n, m, and o, respectively, the composition of LTS networks following Definition 6.1 is* associative*, i.e., it holds that*

$$(\mathcal{M}_\Pi \parallel_\cap \mathcal{M}_\mathrm{P}) \parallel_\cap \mathcal{M}_\Sigma = \mathcal{M}_\Pi \parallel_\cap (\mathcal{M}_\mathrm{P} \parallel_\cap \mathcal{M}_\Sigma)$$

Proof. If the LTS networks $(\mathcal{M}_\Pi \parallel_\cap \mathcal{M}_\mathrm{P}) \parallel_\cap \mathcal{M}_\Sigma$ and $\mathcal{M}_\Pi \parallel_\cap (\mathcal{M}_\mathrm{P} \parallel_\cap \mathcal{M}_\Sigma)$ are equivalent, then this means that their process vectors are equivalent and their sets of synchronisation laws are equivalent.

First of all, the process vectors are equivalent due to associativity of the vector concatenation operator $\|$:

$$\Pi \parallel (\mathrm{P} \parallel \Sigma) = (\Pi \parallel \mathrm{P}) \parallel \Sigma$$

Second of all, we show that $\mathcal{V} \parallel (\mathcal{W} \parallel \mathcal{X}) = (\mathcal{V} \parallel \mathcal{W}) \parallel \mathcal{X}$. Before we do so, for sets of synchronisation laws $\mathcal{Y}$, $\mathcal{Z}$, we introduce an alternative notation for $\mathcal{Y} \setminus \mathcal{Y}_\sigma$ in the context of composing $\mathcal{Y}$ with $\mathcal{Z}$, i.e., $\mathcal{Y} \parallel \mathcal{Z}$. We write $\mathcal{Y} \setminus \mathcal{Z}_\mathcal{A}$ to emphasise the relevance of the alphabet of $\mathcal{Y}$ that is in common with that of $\mathcal{Z}$. We define $\mathcal{Y} \setminus \mathcal{Z}_\mathcal{A} = \{(\bar{y}, a) \in \mathcal{Y} \mid a \notin \mathcal{A}_\mathcal{Z}\}$. The set $\mathcal{Y} \setminus \mathcal{Z}_\mathcal{A}$ is equivalent to $\mathcal{Y} \setminus \mathcal{Y}_\sigma$:

$$\mathcal{Y} \setminus \mathcal{Z}_\mathcal{A} = \{(\bar{y}, a) \in \mathcal{Y} \mid a \notin \mathcal{A}_\mathcal{Z}\} = \mathcal{Y} \setminus \{(\bar{y}, a) \in \mathcal{Y} \mid (a, a, a) \in \sigma\} = \mathcal{Y} \setminus \mathcal{Y}_\sigma$$

The set $\mathcal{Z} \setminus \mathcal{Y}_\mathcal{A}$ is defined similarly.

The associativity proof proceeds as follows. Following Definition 4.2, we partition both the set $\mathcal{V} \parallel (\mathcal{W} \parallel \mathcal{X})$ and the set $(\mathcal{V} \parallel \mathcal{W}) \parallel \mathcal{X}$, each into seven subsets, and show that there is a one-to-one mapping of each of the seven subsets of one partition with one of the subsets of the other partition. Each of the rewrite equations consists of four steps:

(1) unfolding or applying the outer definition of $^\bullet$ or σ, respectively;
(2) unfolding or applying the inner definition of $^\bullet$ or σ, respectively;
(3) applying associativity of vector concatenation and the inner definition of $^\bullet$ or σ;
(4) applying the outer definition of $^\bullet$ or σ.

Furthermore, in cases 2*a* and 3*c* below, the following property of composition of sets of laws is applied in steps (2) and (3), respectively: $a \notin \mathcal{A}_{\mathcal{W} \parallel \mathcal{X}} = a \notin \mathcal{A}_\mathcal{W} \cup \mathcal{A}_\mathcal{X} = a \notin \mathcal{A}_\mathcal{W} \wedge a \notin \mathcal{A}_\mathcal{X}$.

The partitioning and partition mapping proceed as follows.

1. $\mathcal{L}_\sigma(\mathcal{V}, \mathcal{W} \parallel \mathcal{X}) = \mathcal{L}_\sigma(\mathcal{V}, (\mathcal{W} \setminus \mathcal{W}_\sigma)^\bullet \cup {}^\bullet(\mathcal{X} \setminus \mathcal{X}_\sigma) \cup \mathcal{L}_\sigma(\mathcal{W}, \mathcal{X}))$. According to Lemma 6.2, this can be partitioned into:
   (a) $\mathcal{L}_\sigma(\mathcal{V}, \mathcal{L}_\sigma(\mathcal{W}, \mathcal{X}))$, the set of laws specifying synchronisations involving all LTS networks.

$$\begin{aligned}
&\mathcal{L}_\sigma(\mathcal{V}, \mathcal{L}_\sigma(\mathcal{W}, \mathcal{X})) \\
&\quad \overset{(1)}{=} \{\bar{v} \parallel (\bar{w} \parallel \bar{x}) \mid (\bar{v}, a) \in \mathcal{V} \wedge (\bar{w} \parallel \bar{x}, a) \in \mathcal{L}_\sigma(\mathcal{W}, \mathcal{X})\} \\
&\quad \overset{(2)}{=} \{\bar{v} \parallel (\bar{w} \parallel \bar{x}) \mid (\bar{v}, a) \in \mathcal{V} \wedge (\bar{w}, a) \in \mathcal{W} \wedge (\bar{x}, a) \in \mathcal{X}\} \\
&\quad \overset{(3)}{=} \{(\bar{v} \parallel \bar{w}) \parallel \bar{x} \mid (\bar{v} \parallel \bar{w}, a) \in \mathcal{L}_\sigma(\mathcal{V}, \mathcal{W}) \wedge (\bar{x}, a) \in \mathcal{X}\} \\
&\quad \overset{(4)}{=} \mathcal{L}_\sigma(\mathcal{L}_\sigma(\mathcal{V}, \mathcal{W}), \mathcal{X}))
\end{aligned}$$

   (b) $\mathcal{L}_\sigma(\mathcal{V}, (\mathcal{W} \setminus \mathcal{X}_\mathcal{A})^\bullet)$, the set of laws synchronising only $\mathcal{M}_\Pi$ and $\mathcal{M}_\mathrm{P}$.

$$\begin{aligned}
&\sigma(\mathcal{V}, (\mathcal{W} \setminus \mathcal{X}_\mathcal{A})^\bullet) \\
&\quad \overset{(1)}{=} \{\bar{v} \parallel (\bar{w} \parallel \bullet^o) \mid (\bar{v}, a) \in \mathcal{V} \wedge (\bar{w} \parallel \bullet^o, a) \in (\mathcal{W} \setminus \mathcal{X}_\mathcal{A})^\bullet\} \\
&\quad \overset{(2)}{=} \{\bar{v} \parallel (\bar{w} \parallel \bullet^o) \mid (\bar{v}, a) \in \mathcal{V} \wedge (\bar{w}, a) \in \mathcal{W} \wedge a \notin \mathcal{A}_\mathcal{X}\} \\
&\quad \overset{(3)}{=} \{(\bar{v} \parallel \bar{w}) \parallel \bullet^o \mid (\bar{v} \parallel \bar{w}, a) \in \mathcal{L}_\sigma(\mathcal{V}, \mathcal{W}) \wedge a \notin \mathcal{A}_\mathcal{X}\} \\
&\quad \overset{(4)}{=} (\mathcal{L}_\sigma(\mathcal{V}, \mathcal{W}) \setminus \mathcal{X}_\mathcal{A})^\bullet
\end{aligned}$$

   (c) $\mathcal{L}_\sigma(\mathcal{V}, {}^\bullet(\mathcal{X} \setminus \mathcal{W}_\mathcal{A}))$, the set of laws synchronising only $\mathcal{M}_\Pi$ and $\mathcal{M}_\Sigma$.

$$\begin{aligned}
&\sigma(\mathcal{V}, {}^\bullet(\mathcal{X} \setminus \mathcal{W}_\mathcal{A})) \\
&\quad \overset{(1)}{=} \{\bar{v} \parallel (\bullet^m \parallel \bar{x}) \mid (\bar{v}, a) \in \mathcal{V} \wedge (\bullet^m \parallel \bar{x}, a) \in \mathcal{X} \setminus \mathcal{W}_\mathcal{A}\} \\
&\quad \overset{(2)}{=} \{\bar{v} \parallel (\bullet^m \parallel \bar{x}) \mid (\bar{v}, a) \in \mathcal{V} \wedge (\bar{x}, a) \in \mathcal{X} \wedge a \notin \mathcal{A}_\mathcal{W}\} \\
&\quad \overset{(3)}{=} \{(\bar{v} \parallel \bullet^m) \parallel \bar{x} \mid (\bar{v} \parallel \bullet^m) \in (\mathcal{V} \setminus \mathcal{W}_\mathcal{A})^\bullet \wedge (\bar{x}, a) \in \mathcal{X}\} \\
&\quad \overset{(4)}{=} \mathcal{L}_\sigma((\mathcal{V} \setminus \mathcal{W}_\mathcal{A})^\bullet, \mathcal{X})
\end{aligned}$$

2. $(\mathcal{V} \setminus (\mathcal{W} \parallel \mathcal{X})_\mathcal{A})^\bullet$ requires no partitioning:

(a) $(\mathcal{V} \setminus (\mathcal{W} \parallel \mathcal{X})_{\mathcal{A}})^\bullet$, the set of laws specifying the independent behaviour of $\mathcal{M}_\Pi$.

$$
\begin{aligned}
&(\mathcal{V} \setminus (\mathcal{W} \parallel \mathcal{X})_{\mathcal{A}})^\bullet \\
&\quad \overset{(1)}{=} \{\bar{v} \parallel \bullet^{m+o} \mid (\bar{v}, a) \in \mathcal{V} \wedge a \notin \mathcal{A}_{\mathcal{W} \parallel \mathcal{X}}\} \\
&\quad \overset{(2)}{=} \{\bar{v} \parallel (\bullet^m \parallel \bullet^o) \mid (\bar{v}, a) \in \mathcal{V} \wedge a \notin \mathcal{A}_{\mathcal{W}} \wedge a \notin \mathcal{A}_{\mathcal{X}}\} \\
&\quad \overset{(3)}{=} \{(\bar{v} \parallel \bullet^m) \parallel \bullet^o \mid (\bar{v} \parallel \bullet^m, a) \in (\mathcal{V} \setminus \mathcal{W}_{\mathcal{A}})^\bullet \wedge a \notin \mathcal{A}_{\mathcal{X}}\} \\
&\quad \overset{(4)}{=} ((\mathcal{V} \setminus \mathcal{W}_{\mathcal{A}})^\bullet \setminus \mathcal{X}_{\mathcal{A}})^\bullet
\end{aligned}
$$

3. $^\bullet((\mathcal{W} \parallel \mathcal{X}) \setminus \mathcal{V}_{\mathcal{A}})$ is partitioned, applying Lemma 6.1, into:
(a) $^\bullet(\sigma(\mathcal{W}, \mathcal{X}) \setminus \mathcal{V}_{\mathcal{A}})$, the set of laws synchronising only $\mathcal{M}_{\mathrm{P}}$ and $\mathcal{M}_\Sigma$.

$$
\begin{aligned}
&{}^\bullet(\sigma(\mathcal{W}, \mathcal{X}) \setminus \mathcal{V}_{\mathcal{A}}) \\
&\quad \overset{(1)}{=} \{(\bullet^n \parallel (\bar{w} \parallel \bar{x}), a) \mid (\bar{w} \parallel \bar{x}, a) \in \mathcal{L}_\sigma(\mathcal{W}, \mathcal{X}) \wedge a \notin \mathcal{A}_{\mathcal{V}}\} \\
&\quad \overset{(2)}{=} \{(\bullet^n \parallel (\bar{w} \parallel \bar{x}), a) \mid (\bar{w}, a) \in \mathcal{W} \wedge (\bar{x}, a) \in \mathcal{X} \wedge a \notin \mathcal{A}_{\mathcal{V}}\} \\
&\quad \overset{(3)}{=} \{((\bullet^n \parallel \bar{w}) \parallel \bar{x}, a) \mid (\bullet^n \parallel \bar{w}, a) \in {}^\bullet(\mathcal{W} \setminus \mathcal{V}_{\mathcal{A}}) \wedge (\bar{x}, a) \in \mathcal{X}\} \\
&\quad \overset{(4)}{=} \mathcal{L}_\sigma({}^\bullet(\mathcal{W} \setminus \mathcal{V}_{\mathcal{A}}), \mathcal{X})
\end{aligned}
$$

(b) $^\bullet((\mathcal{W} \setminus \mathcal{X}_{\mathcal{A}})^\bullet \setminus \mathcal{V}_{\mathcal{A}})$, the set of laws regarding independent behaviour of $\mathcal{M}_{\mathrm{P}}$.

$$
\begin{aligned}
&{}^\bullet((\mathcal{W} \setminus \mathcal{X}_{\mathcal{A}})^\bullet \setminus \mathcal{V}_{\mathcal{A}}) \\
&\quad \overset{(1)}{=} \{(\bullet^n \parallel (\bar{w} \parallel \bullet^o), a) \mid (\bar{w} \parallel \bullet^o, a) \in (\mathcal{W} \setminus \mathcal{X}_{\mathcal{A}})^\bullet \wedge a \notin \mathcal{A}_{\mathcal{V}}\} \\
&\quad \overset{(2)}{=} \{(\bullet^n \parallel (\bar{w} \parallel \bullet^o), a) \mid (\bar{w}, a) \in \mathcal{W} \wedge a \notin \mathcal{A}_{\mathcal{V}} \wedge a \notin \mathcal{A}_{\mathcal{X}}\} \\
&\quad \overset{(3)}{=} \{((\bullet^n \parallel \bar{w}) \parallel \bullet^o, a) \mid (\bullet^n \parallel \bar{w}, a) \in {}^\bullet(\mathcal{W} \setminus \mathcal{V}_{\mathcal{A}}) \wedge a \notin \mathcal{A}_{\mathcal{X}}\} \\
&\quad \overset{(4)}{=} ({}^\bullet(\mathcal{W} \setminus \mathcal{V}_{\mathcal{A}}) \setminus \mathcal{X}_{\mathcal{A}})^\bullet
\end{aligned}
$$

(c) $^\bullet({}^\bullet(\mathcal{X} \setminus \mathcal{W}_{\mathcal{A}}) \setminus \mathcal{V}_{\mathcal{A}})$, the set of laws specifying independent behaviour of $\mathcal{M}_\Sigma$.

$$
\begin{aligned}
&{}^\bullet({}^\bullet(\mathcal{X} \setminus \mathcal{W}_{\mathcal{A}}) \setminus \mathcal{V}_{\mathcal{A}}) \\
&\quad \overset{(1)}{=} \{(\bullet^n \parallel (\bullet^m \parallel \bar{x}), a) \mid (\bullet^m \parallel \bar{x}, a) \in {}^\bullet(\mathcal{X} \setminus \mathcal{W}_{\mathcal{A}}) \wedge a \notin \mathcal{A}_{\mathcal{V}}\} \\
&\quad \overset{(2)}{=} \{(\bullet^n \parallel (\bullet^m \parallel \bar{x}), a) \mid (\bar{x}, a) \in \mathcal{X} \wedge a \notin \mathcal{A}_{\mathcal{W}} \wedge a \notin \mathcal{A}_{\mathcal{V}}\} \\
&\quad \overset{(3)}{=} \{(\bullet^{n+m} \parallel \bar{x}, a) \mid (\bar{x}, a) \in \mathcal{X} \wedge a \notin \mathcal{A}_{\mathcal{V} \parallel \mathcal{W}}\} \\
&\quad \overset{(4)}{=} {}^\bullet(\mathcal{X} \setminus (\mathcal{V} \parallel \mathcal{W})_{\mathcal{A}})
\end{aligned}
$$

These equations constitute a one-to-one mapping between the subsets of the partition of $\mathcal{V} \parallel (\mathcal{W} \parallel \mathcal{X})$ and those of the partition of $(\mathcal{V} \parallel \mathcal{W}) \parallel \mathcal{X}$. Therefore, we have $\mathcal{V} \parallel (\mathcal{W} \parallel \mathcal{X}) = (\mathcal{V} \parallel \mathcal{W}) \parallel \mathcal{X}$.

Since both $\Pi \parallel (\mathrm{P} \parallel \Sigma) = (\Pi \parallel \mathrm{P}) \parallel \Sigma$ and $\mathcal{V} \parallel (\mathcal{W} \parallel \mathcal{X}) = (\mathcal{V} \parallel \mathcal{W}) \parallel \mathcal{X}$ it follows that $(\mathcal{M}_\Pi \parallel_\cap \mathcal{M}_{\mathrm{P}}) \parallel_\cap \mathcal{M}_\Sigma = \mathcal{M}_\Pi \parallel_\cap (\mathcal{M}_{\mathrm{P}} \parallel_\cap \mathcal{M}_\Sigma)$. □

Commutativity It is clear that composition of LTS networks with synchronisation on the common alphabet of the components is not commutative w.r.t. LTS network equivalence, as is indicated by Example 6.1.

Example 6.1. Let $\mathcal{M}_\Pi = (\Pi, \mathcal{V})$ and $\mathcal{M}_{\mathrm{P}} = (\mathrm{P}, \mathcal{W})$ be two LTS networks. Furthermore, consider compositions $\mathcal{M}_1 = \mathcal{M}_\Pi \parallel_\cap \mathcal{M}_{\mathrm{P}}$ and $\mathcal{M}_2 = \mathcal{M}_{\mathrm{P}} \parallel_\cap \mathcal{M}_\Pi$. The LTS network $\mathcal{M}_1$ has process vector $\Pi \parallel \mathrm{P}$ while $\mathcal{M}_2$ has process vector $\mathrm{P} \parallel \Pi$. Unless $\mathcal{M}_\Pi = \mathcal{M}_{\mathrm{P}}$, $\mathcal{M}_1$ and M_2 are strictly not equivalent. Similarly, the synchronisation laws of both composite LTS networks are in a different order.

However, LTS network composition as defined in Definition 6.1 is commutative with respect to the semantics of the constructed LTS network. That is, for the semantics of the composition of LTS networks, it does not matter in which order the LTS networks are composed. We first prove that such a composition is commutative with respect to (strong) bisimulation [2]

in Proposition 6.2. Afterwards, we propose an adaption of the definition of LTS network, fixing the ordering issue by replacing vectors with indexed families gaining a commutative operator for composition of LTS networks with synchronisation on the common alphabet.

Proposition 6.2. *Let $\mathcal{M}_\Pi = (\Pi, \mathcal{V})$ and $\mathcal{M}_\mathrm{P} = (\mathrm{P}, \mathcal{W})$ be LTS networks of sizes n and m respectively. Composition of LTS networks according to Definition 6.1 is* commutative with respect to (strong) bisimulation*, i.e., it holds that $\mathcal{G}_{\mathcal{M}_\Pi \parallel_\cap \mathcal{M}_\mathrm{P}} \leftrightarrows \mathcal{G}_{\mathcal{M}_\mathrm{P} \parallel_\cap \mathcal{M}_\Pi}$.*

Proof. Take the relation $\mathsf{C} = \{(\bar{s} \parallel \bar{t}, \bar{t} \parallel \bar{s}) \mid \bar{s} \in \mathcal{G}_{\mathcal{M}_\Pi} \wedge \bar{t} \in \mathcal{G}_{\mathcal{M}_\mathrm{P}}\}$. The relation C is a (strong) bisimulation relation.

- C *relates the initial states of $\mathcal{M}_\Pi$ and $\mathcal{M}_\mathrm{P}$.* Since every state $\bar{s} \parallel \bar{t} \in \mathcal{I}_{\mathcal{M}_\Pi \parallel_\cap \mathcal{M}_\mathrm{P}}$ is related by C to state $\bar{t} \parallel \bar{s} \in \mathcal{I}_{\mathcal{M}_\mathrm{P} \parallel_\cap \mathcal{M}_\Pi}$ and vice versa.
- *If $\bar{s} \parallel \bar{t}\ \mathsf{C}\ \bar{t} \parallel \bar{s}$ and $\bar{s} \parallel \bar{t} \xrightarrow{a}_{\mathcal{M}_\Pi \parallel_\cap \mathcal{M}_\mathrm{P}} \bar{s}' \parallel \bar{t}'$ then $\bar{t} \parallel \bar{s} \xrightarrow{a}_{\mathcal{M}_\mathrm{P} \parallel_\cap \mathcal{M}_\Pi} \bar{t}'' \parallel \bar{s}'' \wedge \bar{s}' \parallel \bar{t}'\ \mathsf{C}\ \bar{t}'' \parallel \bar{s}''$.* Let $(\bar{v} \parallel \bar{w}, a) \in \mathcal{V} \parallel \mathcal{W}$ be the law enabling the transition $\bar{s} \parallel \bar{t} \xrightarrow{a}_{\mathcal{M}_\Pi \parallel_\cap \mathcal{M}_\mathrm{P}} \bar{s}' \parallel \bar{t}'$. It follows that there is a law $(\bar{w} \parallel \bar{v}, a) \in \mathcal{W} \parallel \mathcal{V}$ that enables the transition $\bar{t} \parallel \bar{s} \xrightarrow{a}_{\mathcal{M}_\mathrm{P} \parallel_\cap \mathcal{M}_\Pi} \bar{t}' \parallel \bar{s}'$. As $\bar{s}' \parallel \bar{t}'\ \mathsf{C}\ \bar{t}' \parallel \bar{s}'$, the proof follows by taking $\bar{t}'$ for $\bar{t}''$, and $\bar{s}'$ for $\bar{s}''$.
- *If $\bar{s} \parallel \bar{t}\ \mathsf{C}\ \bar{t} \parallel \bar{s}$ and $\bar{t} \parallel \bar{s} \xrightarrow{a}_{\mathcal{M}_\mathrm{P} \parallel_\cap \mathcal{M}_\Pi} \bar{t}' \parallel \bar{s}'$ then $\bar{s} \parallel \bar{t} \xrightarrow{a}_{\mathcal{M}_\Pi \parallel_\cap \mathcal{M}_\mathrm{P}} \bar{s}'' \parallel \bar{t}'' \wedge \bar{s}'' \parallel \bar{t}''\ \mathsf{C}\ \bar{t}' \parallel \bar{s}'$.* This case is symmetric to the previous case. □

To avoid the issues discussed in Example 6.1, an alternative definition of LTS network can be designed. Both process vectors and synchronisation vectors may be replaced by indexed families. An *indexed family* consists of a set of objects (the process LTSs or synchronisation vectors), an index set, and a surjective function mapping elements from the index set to elements of the set of objects. When the index sets of two LTS networks are disjoint, then the union of sets can be applied, where we previously would use vector concatenation, to compose the collections of process LTSs and synchronisation laws. The union of two indexed families is commutative, and as such, commutativity of composition of LTS networks with indexed families is also commutative.

7. Two more congruence results for DPBB and LTS networks

In this section we first prove that DPBB is a congruence for the parallel composition of an arbitrary number of LTS networks using the parallel composition operator with synchronisation on the common alphabet. After that, we prove that synchronisation on the common alphabet is actually not a requirement for DPBB to be a congruence for LTS networks. For this, we generalise the definition of congruence as given by [23].

That DPBB is a congruence for the parallel composition of multiple LTS networks using the parallel composition operator with synchronisation on the common alphabet of components (Definition 6.1) follows from the associativity and commutativity of that parallel composition operator:

Proposition 7.1. *Consider two vectors of LTSs Π and P, both of size n, where for some $i \in 1..n$, $\Pi_i \leftrightarrows_b^\Delta \mathrm{P}_i$, and for all $j \in 1..n \setminus \{i\}$, $\Pi_j = \mathrm{P}_j$. Furthermore, consider a set of synchronisation laws $\mathcal{V}$ with vectors of size n. If $\mathcal{V}$ does not rename, cut, or synchronise τ-transitions, then we have*

$$\mathcal{G}_{(\Pi, \mathcal{V})} \leftrightarrows_b^\Delta \mathcal{G}_{(\mathrm{P}, \mathcal{V})}$$

Proof. LTS network $\mathcal{M}_\Pi = (\Pi, \mathcal{V})$ can be decomposed, by Proposition 5.2, into $\mathcal{M}_\Pi^0 \parallel_\cap \mathcal{M}_\Pi^1$, with $\mathcal{M}_\Pi^0 = (\langle \Pi_0, \ldots, \Pi_{i-1} \rangle, \overleftarrow{\mathtt{il}(\mathcal{V})} \cup \overleftarrow{\mathtt{il}(\mathcal{V})})$ and $\mathcal{M}_\Pi^1 = (\langle \Pi_i, \ldots, \Pi_n \rangle, \overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})})$, since the decomposition method described by Proposition 5.2 introduces for all inter-component synchronisation laws $\mathtt{il}(\mathcal{V})$ of $\mathcal{M}_\Pi$ common actions in the alphabets of $\mathcal{M}_\Pi^0$ and $\mathcal{M}_\Pi^1$ to synchronise on. This follows from the fact that for all laws $(\bar{v}, a) \in \mathtt{il}(\mathcal{V})$, $f(\bar{v}, a) = g(\bar{v}, a)$.

In turn, $\mathcal{M}_\Pi^1$ can be decomposed into $\mathcal{M}_\Pi^2 \parallel_\cap \mathcal{M}_\Pi^3$, with $\mathcal{W} = \overrightarrow{\mathtt{rl}(\mathcal{V})} \cup \overrightarrow{\mathtt{il}(\mathcal{V})}$, $\mathcal{M}_\Pi^2 = (\langle \Pi_i \rangle, \overleftarrow{\mathtt{il}(\mathcal{W})} \cup \overleftarrow{\mathtt{il}(\mathcal{W})})$, and $\mathcal{M}_\Pi^3 = (\langle \Pi_{i+1}, \ldots, \Pi_n \rangle, \overrightarrow{\mathtt{rl}(\mathcal{W})} \cup \overrightarrow{\mathtt{il}(\mathcal{W})})$.

This decomposition leads to $M_\Pi^0 \parallel_\cap (\mathcal{M}_\Pi^2 \parallel_\cap \mathcal{M}_\Pi^3)$. By associativity (Proposition 6.1) and commutativity w.r.t. DPBB (Proposition 6.2) of $\parallel_\cap$, we have $M_\Pi^0 \parallel_\cap (\mathcal{M}_\Pi^2 \parallel_\cap \mathcal{M}_\Pi^3) = (M_\Pi^0 \parallel_\cap \mathcal{M}_\Pi^2) \parallel_\cap \mathcal{M}_\Pi^3 = (M_\Pi^2 \parallel_\cap \mathcal{M}_\Pi^0) \parallel_\cap \mathcal{M}_\Pi^3$.

Similarly to $\mathcal{M}_\Pi$, we can decompose $\mathcal{M}_\mathrm{P}$ into $\mathcal{M}_\mathrm{P}^0 \parallel_\cap (\mathcal{M}_\mathrm{P}^2 \parallel_\cap \mathcal{M}_\mathrm{P}^3)$, which is equivalent to $(\mathcal{M}_\mathrm{P}^2 \parallel_\cap \mathcal{M}_\mathrm{P}^0) \parallel_\cap \mathcal{M}_\mathrm{P}^3$.

The fact that $\Pi_i \leftrightarrows_b^\Delta \mathrm{P}_i$ implies that $\mathcal{M}_\Pi^2 \leftrightarrows_b^\Delta \mathcal{M}_\mathrm{P}^2$, since the sets of synchronisation laws of those LTS networks are equivalent (those sets have both been derived in the same way from $\mathcal{V}$ for the i-th element in their respective process vectors), and those laws do not rename, cut or synchronise τ's. Because of this, and the fact that $\parallel_\cap$ is an instance of $\parallel_\sigma$, we have, by Proposition 4.2, that $(\mathcal{M}_\Pi^2 \parallel_\cap \mathcal{M}_\Pi^0) \leftrightarrows_b^\Delta (\mathcal{M}_\mathrm{P}^2 \parallel_\cap \mathcal{M}_\Pi^0)$, and, in turn, that $(\mathcal{M}_\Pi^2 \parallel_\cap \mathcal{M}_\Pi^0) \parallel_\cap \mathcal{M}_\Pi^3 \leftrightarrows_b^\Delta (\mathcal{M}_\mathrm{P}^2 \parallel_\cap \mathcal{M}_\Pi^0) \parallel_\cap \mathcal{M}_\mathrm{P}^3$. By definition of $\leftrightarrows_b^\Delta$ for LTS networks, this means that $\mathcal{G}_{(\mathcal{M}_\Pi^2 \parallel_\cap \mathcal{M}_\Pi^0) \parallel_\cap \mathcal{M}_\Pi^3} \leftrightarrows_b^\Delta \mathcal{G}_{(\mathcal{M}_\mathrm{P}^2 \parallel_\cap \mathcal{M}_\mathrm{P}^0) \parallel_\cap \mathcal{M}_\mathrm{P}^3}$. Again, due to the associativity and commutativity w.r.t. DPBB of $\parallel_\cap$, this is equal to $\mathcal{G}_{\mathcal{M}_\Pi^0 \parallel_\cap (\mathcal{M}_\Pi^2 \parallel_\cap \mathcal{M}_\Pi^3)} \leftrightarrows_b^\Delta \mathcal{G}_{\mathcal{M}_\mathrm{P}^0 \parallel_\cap (\mathcal{M}_\mathrm{P}^2 \parallel_\cap \mathcal{M}_\mathrm{P}^3)}$, which, because decomposition according to Proposition 5.2 is consistent, means that $\mathcal{G}_{\mathcal{M}_\Pi} \leftrightarrows_b^\Delta \mathcal{G}_{\mathcal{M}_\mathrm{P}} = \mathcal{G}_{(\Pi, \mathcal{V})} \leftrightarrows_b^\Delta \mathcal{G}_{(\mathrm{P}, \mathcal{V})}$. □

However, it is unnecessary to require that the set of synchronisation laws implements synchronisation on the common alphabet of the components. As this requirement excludes many LTS networks in practice, we discuss next an alternative proof, for a reformulated version of Proposition 7.1 in the form of Proposition 7.2. In this new proposition, *each* component is DPBB-related to another component. Note that Proposition 7.1 is a special case of Proposition 7.2, in which all but one component are DPBB related to themselves. The proof for Proposition 7.2 does not require synchronisation on the common alphabet.

Proposition 7.2. *Consider two vectors of LTSs* Π *and* P*, and a set of synchronisation laws* $\mathcal{V}$*. Furthermore, assume that* τ*-transitions are not renamed, cut, or synchronised. It holds that*

$$(\forall i \in 1..n.\ \Pi_i \leftrightarrow_b^{\Delta} \mathrm{P}_i) \implies \mathcal{G}_{(\Pi,\mathcal{V})} \leftrightarrow_b^{\Delta} \mathcal{G}_{(\mathrm{P},\mathcal{V})}$$

Proof. Given two vectors of LTSs Π and P such that for all $i \in 1..n$ there is a DPBB relation B_i with $\Pi_i\ \mathsf{B}_i\ \mathrm{P}_i$. We define the bisimulation relation C as follows:

$$\mathsf{C} = \{(\bar{s}, \bar{t}) \mid \bar{s} \in \mathcal{S}_{(\Pi,\mathcal{V})} \wedge \bar{t} \in \mathcal{S}_{(\mathrm{P},\mathcal{V})} \wedge \forall i \in 1..n.\ \bar{s}_i\ \mathsf{B}_i\ \bar{t}_i\}$$

We prove that C is a DPBB relation as defined in Definition 3.5. We will use $Ac(\bar{\nu}) = \{i \mid i \in 1..n \wedge \bar{\nu}_i \neq \bullet\}$ as a shorthand for the set of indices of processes participating in a synchronisation law $(\bar{\nu}, a)$; e.g., $Ac(\langle c, b, \bullet\rangle) = \{1, 2\}$.

- C *relates the initial states of* $\mathcal{M}_\Pi$ *and* $\mathcal{M}_\mathrm{P}$. Consider a state $\bar{s} \in \mathcal{I}_{(\Pi,\mathcal{V})}$. For each $i \in 1..n$ there is a state $q_i \in \mathcal{I}_{\mathrm{P}_i}$ such that $\bar{s}_i\ \mathsf{B}_i\ t_i$, since B_i is a DPBB relation between Π_i and P_i. Let $\bar{t}$ be the state constructed from these q_i, i.e., for all $i \in 1..n$, we have $\bar{t}_i = q_i$. Then, $\bar{t} \in \mathcal{I}_{(\mathrm{P},\mathcal{V})}$ and $\bar{s}\ \mathsf{C}\ \bar{t}$. The symmetric case follows similarly.
- *If* $\bar{s}\ \mathsf{C}\ \bar{t}$ *and* $\bar{s} \xrightarrow{a}_{(\Pi,\mathcal{V})} \bar{s}'$ *then either* $a = \tau \wedge \bar{s}'\ \mathsf{C}\ \bar{t}$*, or* $\bar{t} \xrightarrow{\tau}{}^{*}_{(\mathrm{P},\mathcal{V})} \hat{\bar{t}} \xrightarrow{a}_{(\mathrm{P},\mathcal{V})} \bar{t}' \wedge \bar{s}\ \mathsf{C}\ \hat{\bar{t}} \wedge \bar{s}'\ \mathsf{C}\ \bar{t}'$. Consider a law $(\bar{\nu}, a) \in \mathcal{V}$ enabling transition $\bar{s} \xrightarrow{a}_{(\Pi,\mathcal{V})} \bar{s}'$. We distinguish two cases:
  1. There is a τ-action in synchronisation vector $\bar{\nu}$, i.e., $\exists i \in 1..n.\ \bar{\nu}_i = \tau$. Therefore, there is a transition $\bar{s}_i \xrightarrow{\tau}_i \bar{s}'_i$. Since τ-transitions do not synchronise it follows that it is the only action in the synchronisation vector, i.e., $Ac(\bar{\nu}) = \{i\}$. Hence, $a = \tau$ as renaming τ-transitions is not allowed. Furthermore, by Definition 3.3, for all $j \in 1..n \setminus \{i\}$ it holds that $\bar{s}_j = \bar{s}'_j$. As we also have $\bar{s}_j\ \mathsf{B}_j\ \bar{t}_j$, it follows that $\bar{s}'_j\ \mathsf{B}_j\ \bar{t}_j$.

     Because $\bar{s}_i\ \mathsf{B}_i\ \bar{t}_i$ and $\bar{s}_i \xrightarrow{\tau}_i \bar{s}'_i$, by Definition 3.5, two cases can occur:
     - $a = \tau$ with $\bar{s}'_i\ \mathsf{B}_i\ \bar{t}_i$. Hence, for all $j \in 1..n$ we have $\bar{s}_j\ \mathsf{B}_j\ \bar{t}_j$. By definition of C, it follows that $\bar{s}'\ \mathsf{C}\ \bar{t}$.
     - $\bar{t}_i \xrightarrow{\tau}{}^{*}_i \hat{t} \xrightarrow{a}_i t'$ with $\bar{s}_i\ \mathsf{B}_i\ \hat{t}$ and $\bar{s}'_i\ \mathsf{B}_i\ t'$. Since no τ-transitions are cut, there also exists a path $\bar{t} \xrightarrow{\tau}{}^{*}_{(\mathrm{P},\mathcal{V})} \hat{\bar{t}} \xrightarrow{a}_{(\mathrm{P},\mathcal{V})} \bar{t}'$ with $\hat{\bar{t}}_i = \hat{t}$, $\bar{t}'_i = t'$, and for all $j \in 1..n \setminus \{i\}$ we have $\bar{t}'_j = \hat{\bar{t}}_j = \bar{t}_j$. Therefore, from $\bar{s}_i\ \mathsf{B}_i\ \hat{t}$, $\bar{s}'_i\ \mathsf{B}_i\ t'$, and $\forall j \in 1..n \setminus \{i\}.\ \bar{s}_j\ \mathsf{B}_j\ \bar{t}_j$ we deduce that $\bar{s}\ \mathsf{C}\ \hat{\bar{t}}$ and $\bar{s}'\ \mathsf{C}\ \bar{t}'$.
  2. There is *no* τ-action in synchronisation vector $\bar{\nu}$, i.e., $\forall i \in 1..n.\ \bar{\nu}_i \neq \tau$. By Definition 3.3, for all $j \in 1..n \setminus Ac(\bar{\nu})$ we have $\bar{s}'_j = \bar{s}_j$. Thus, since $\bar{s}_j\ \mathsf{B}_j\ \bar{t}_j$ it follows that $\bar{s}'_j\ \mathsf{B}_j\ \bar{t}_j$. Furthermore, we have for all $i \in Ac(\bar{\nu})$ a transition $\bar{s}_i \xrightarrow{\bar{\nu}_i}_i \bar{s}'_i$. Hence, as $\bar{\nu}_i \neq \tau$ for all those $i \in Ac(\bar{\nu})$, there exists a path $\bar{t}_i \xrightarrow{\tau}{}^{*}_i \hat{\bar{t}}_i \xrightarrow{\bar{\nu}_i} \bar{t}'_i$ with $\bar{s}_i\ \mathsf{B}_i\ \hat{\bar{t}}_i$ and $\bar{s}'_i\ \mathsf{B}_i\ \bar{t}'_i$ (by Definition 3.5). From Definition 3.3 it follows that there also is a path $\bar{t} \xrightarrow{\tau}{}^{*}_{(\mathrm{P},\mathcal{V})} \hat{\bar{t}} \xrightarrow{a}_{(\mathrm{P},\mathcal{V})} \bar{t}'$ where for all $j \in 1..n \setminus Ac(\bar{\nu})$, $\hat{\bar{t}}_j$ and $\bar{t}'_j$ are defined by $\bar{t}'_j = \hat{\bar{t}}_j = \bar{t}_j$. Hence, from $\forall i \in 1..n.\ \bar{s}_i\ \mathsf{B}_i\ \bar{t}_i$, $\forall i \in Ac(\bar{\nu}).\ \bar{s}_i\ \mathsf{B}_i\ \hat{\bar{t}}_i$, and $\forall i \in Ac(\bar{\nu}).\ \bar{s}'_i\ \mathsf{B}_k\ \bar{t}'_i$ we deduce that $\bar{s}\ \mathsf{C}\ \hat{\bar{t}}$ and $\bar{s}'\ \mathsf{C}\ \hat{\bar{t}}'$.
- *If* $\bar{s}\ \mathsf{C}\ \bar{t}$ *and* $\bar{t} \xrightarrow{a}_{(\mathrm{P},\mathcal{V})} \bar{t}'$ *then either* $a = \tau \wedge \bar{s}'\ \mathsf{C}\ \bar{t}$*, or* $\bar{s} \xrightarrow{\tau}{}^{*}_{(\Pi,\mathcal{V})} \hat{\bar{s}} \xrightarrow{a}_{(\Pi,\mathcal{V})} \bar{s}' \wedge \bar{s}\ \mathsf{C}\ \hat{\bar{t}} \wedge \bar{s}'\ \mathsf{C}\ \bar{t}'$. This case is symmetric to the previous case.
- *If* $\bar{s}\ \mathsf{C}\ \bar{t}$ *and there is an infinite sequence of states* $(\bar{s}^k)_{k\in\omega}$ *such that* $\bar{s} = \bar{s}^0$*,* $\bar{s}^k \xrightarrow{\tau}_{(\Pi,\mathcal{V})} \bar{s}^{k+1}$ *and* $\bar{s}^k\ \mathsf{C}\ \bar{t}$ *for all* $k \in \omega$*, then there exists a state* $\bar{t}'$ *such that* $\bar{t} \xrightarrow{\tau}{}^{+}_{(\mathrm{P},\mathcal{V})} \bar{t}'$ *and* $\bar{s}^k\ \mathsf{C}\ \bar{t}'$ *for some* $k \in \omega$. For all $k \in \omega$, let $(\bar{\nu}^k, \tau) \in \mathcal{V}$ be the synchronisation law enabling transition $s^k \xrightarrow{\tau} s^{k+1}$.
  We distinguish two cases:
  - There is a $k \in \omega$ such that $\bar{s}^k \xrightarrow{\tau} \bar{s}^{k+1}$ is the result of the synchronisation of multiple processes in Π, i.e., $\exists k \in \omega, i \in 1..n.\ \{i\} \subset Ac(\bar{\nu}^k)$. In the τ-sequence, we have $\bar{s}^\ell\ \mathsf{C}\ \bar{t}$ for all $\ell \in \omega$, hence, we have $\bar{s}^k\ \mathsf{C}\ \bar{t}$. Furthermore, since C is a DPBB relation, it follows that there are states $\hat{\bar{t}}, \bar{t}' \in \mathcal{S}_{(\mathrm{P},\mathcal{V})}$ with a τ-path $\bar{t} \xrightarrow{\tau}{}^{*}_{(\mathrm{P},\mathcal{V})} \hat{\bar{t}} \xrightarrow{\tau}_{(\mathrm{P},\mathcal{V})} \bar{t}'$ such that $\bar{s}^{k+1}\ \mathsf{B}\ \bar{t}'$. Thus, $\bar{t} \xrightarrow{\tau}{}^{+}_{(\mathrm{P},\mathcal{V})} \bar{t}'$ and for $k+1 \in \omega$ it holds that $\bar{s}^{k+1}\ \mathsf{C}\ \bar{t}'$, thereby completing the case.
  - The τ-sequence only consists of τ-transitions performed independently by the processes in Π, i.e., $\forall k \in \omega.\ \forall i \in 1..n.\ \{i\} \notin Ac(\bar{\nu}^k)$. Since all of the τ-transitions are performed independently, there has to be at least one process of which an infinite τ-sequence is embedded in the global infinite τ-sequence starting from $\bar{s}$, otherwise the latter τ-sequence would not be infinite. Suppose the ith process has such a τ-sequence, then globally, this τ-sequence starts

from state $\bar{s}_i$. The infinite τ-sequence of Π_i is embedded in the infinite τ-sequence of the LTS network, hence, for all $k \in \omega$ it holds that $\bar{s}_i^k \; \mathsf{B}_i \; \bar{t}_i$. Since $\bar{s}_i \; \mathsf{B}_i \; \bar{t}_i$, by Definition 3.5, there is a state $t' \in \mathcal{S}_{\mathrm{P}_i}$ with $\bar{t}_i \xrightarrow{\tau}{}^{+}_{\mathrm{P}_i} t'$ and some $\ell \in \omega$ such that $\bar{s}_i^\ell \; \mathsf{B}_i \; t'$. We construct state $\bar{t}'$ such that for all $j \in 1..n$, if $j = i$, then $\bar{t}'_j = t'$, and otherwise $\bar{t}'_j = \bar{t}_j$. As local τ-transitions are not cut nor renamed, it follows that $\bar{t} \xrightarrow{\tau}{}^{+}_{(\mathrm{P},\mathcal{V})} \bar{t}'$. Moreover, since $\bar{s}^k \; \mathsf{C} \; \bar{t}$ for all $k \in \omega$, by definition of C, we have $\bar{s}_j^\ell \; \mathsf{B}_j \; \bar{t}_j$ for all $j \in 1..n$. Finally, because $\bar{s}_i^\ell \; \mathsf{B}_i \; \bar{t}'_i$ and for all $j \in 1..n \setminus \{i\}$ it holds that $\bar{s}_j^\ell \; \mathsf{B}_j \; \bar{t}_j$, by construction of $\bar{t}'$ it follows that $\bar{s}^\ell \; \mathsf{C} \; \bar{t}'$.

- *If $\bar{s} \; \mathsf{C} \; \bar{t}$ and there is an infinite sequence of states $(\bar{t}^k)_{k\in\omega}$ such that $\bar{t} = \bar{t}^0$, $\bar{t}^k \xrightarrow{\tau}_{(\mathrm{P},\mathcal{V})} \bar{t}^{k+1}$ and $\bar{s} \; \mathsf{C} \; \bar{t}^k$ for all $k \in \omega$, then there exists a state $\bar{s}'$ such that $\bar{s} \xrightarrow{\tau}{}^{+}_{(\Pi,\mathcal{V})} \bar{s}'$ and $\bar{s}' \; \mathsf{C} \; \bar{t}^k$ for some $k \in \omega$.* This case is symmetric to the previous case. □

8. Application

In order to compare compositional approaches with the classical, non-compositional approach, we have employed CADP to minimise a set of test cases modulo DPBB.

Each test case consists of a model that is minimised with respect to a given liveness property. To achieve the best minimisation we applied maximal hiding [11] in all approaches. Intuitively, maximal hiding hides all actions except for the interface actions and actions relevant for the given liveness property. In general, one can also hide fewer actions, but this will decrease the impact of DPBB reduction, both in compositional and non-compositional model checking.

As *composition strategy* we have used the *smart reduction* approach described in [27]. In CADP, the *classical approach*, where the full state space is constructed at once and no intermediate minimisations are applied, is the *root reduction* strategy.

We have measured the *running time* and the *maximum number of states and transitions* generated by the two methods.

Experimental setup To facilitate replication we briefly discuss the methods used for our experiments.

For compositional approaches, the running time and largest state space considered depend heavily on the composition order, i.e., the order in which the components are combined. The smart reduction approach uses a heuristic to determine the order in which to compose processes. In [27], it has been experimentally established that this heuristic frequently works very well. After each composition step the result is minimised.

We use the following expression from the scripting language SVL of CADP to invoke the smart reduction modulo DPBB approach:

```
smart total divbranching reduction of (<m>)
```

where `<m>` is the test model.

In the classical approach the state space of the entire system is generated before minimisation is applied. This approach is invoked as follows

```
root total divbranching reduction of (<m>)
```

where `<m>` is the test model.

The experiments were run on the DAS-5 cluster [38] machines. They have an INTEL HASWELL E5-2630-v3 2.4 GHz CPU, 64 GB memory, and run CENTOS LINUX 7.2. The running time of the two approaches was measured as the wall clock time (i.e., the real elapsed time) using the Unix time command:

```
/usr/bin/time -f "%e" svl <file>
```

The argument `-f "%e"` specifies that the time written as output should follow format `"%e"` where `%e` indicates the wall clock time. The `svl <file>` argument invokes the SVL-engine with script file `<file>`. The command measures the wall clock time of the execution of the SVL-script.

The maximum number of states and transitions that were generated were extracted from the SVL-log files after execution of the script.

The set of test cases To prevent source bias, studies were selected from four different sources. In total 19 case studies were selected: four MCRL2 [16] models distributed with its tool set, nine CADP models, three from the BEEM database [39], which consists of DVE models for the DIVINE model checker (version 3 and earlier) [40], and three from the Example Repository for Finite State Verification Tools [41]. The CADP models are expressed in the EXP language, which is the language supported by CADP to express LTS networks. All other models have been manually translated to EXP models.

The models stemming from the MCRL2 tool set distribution are the following:

1. The *1394* model, created by Luttik [42], specifies the 1394 or firewire protocol. *Property*: every `PAreq` action with parameter 'immediate' is eventually followed by a matching `PAcon` action with parameter 'won'.

2. The *1394'* model is the 1394 model scaled up with extra internal transitions. This model is our own adaptation of the 1394 model and is therefore not distributed with the tool set. *Property*: the same one as for the 1394 model.
3. The *ACS* model describes the ACS Manager that is part of the ALMA project of the European Southern Observatory. The ACS Manager is part of a system controlling a large collection of radio telescopes. The model consists of a manager and some containers and components and was created by Ploeger [43]. *Property*: every time container MT1 is locked, eventually it is freed again.
4. *Wafer Stepper* models a wafer stepper used in the manufacturing of integrated circuits. *Property*: always, eventually, all wafers in the system will be exposed.

The following CADP models were used:

1. *Cache* models a directory-based cache coherency protocol for a multi-processor architecture. The model was developed by Kahlouche et al. [44]. *Property*: there is no livelock.
2. The *DES* model describes an implementation of the data encryption standard, which allows to cipher and decipher 64-bit vectors using a 64-bit key vector [45]. *Property*: the DES can always deliver outputs.
3. *HAVi-LE* describes the asynchronous Leader Election protocol used in the HAVi (Home Audio-Video) standard, involving three device control managers. The model is fully described by Romijn [46]. *Property*: always eventually a leader is selected.
4. *HAVi-LE'* is an adaptation of the HAVi-LE model containing transitions denoting logging events. Since the model is our own adaptation it is not distributed with CADP. *Property*: the same one as for the HAVi-LE model.
5. *Le Lann* models a distributed leader election algorithm for unidirectional ring networks. The CADP model was developed by Garavel and Mounier [47]. *Property*: process `P0` is infinitely many times in the critical section.
6. *ODP* is a model of an open distributed processing trader [48]. *Property*: work is always executed eventually.
7. The *Erat. Sieve* model computes prime numbers implementing a distributed Eratosthenes Sieve; the model describes a pipeline of units, of which each unit blocks input numbers that are multiples of a given number. The model consists of four units. *Property*: if the number two is generated, then it is eventually reported as a prime number.
8. *Erat. Sieve'* is a variant of Erat. Sieve consisting of seven units. *Property*: the same one as for the Erat. Sieve model.
9. The *Transit* model describes a transit-node. It models an abstraction of a routing component of a communication network. The model was developed by Mounier [49]. *Property*: every time a message is receive, it is eventually either sent out by the node or buffered as faulty.

The following BEEM models were used:

1. The *Peterson* model describes Peterson's mutual exclusion algorithm [50] for seven processes. *Property*: every time process `P0` waits for access to the critical section, it will eventually enter it.
2. *Anderson* models Anderson's queue lock mutual exclusion algorithm [51] for three processes. *Property*: Every time a process waits for access to the critical section, it will eventually enter it.
3. *Anderson'* is a variant of the Anderson model considering four processes competing for a lock. *Property*: the same one as for the Anderson model.

From the Example Repository for Finite State Verification Tools we selected the following models:

1. The *Chiron* model describes a user interface development system with two clients. The system consists of the Chiron server, managing generic aspects of a user interface, and artists (the clients). This server is responsible for notifying artists when a user interface event occurs, while the clients listen for notifications from the server. The formal model was developed by Avrunin et al. [52]. *Property*: If an artist is registered for event `e1`, then it will eventually be notified for this event.
2. *Chiron'* is a adapted version of the Chiron model in which an additional client has been added. In total, there are three clients. *Property*: the same one as for the Chiron model.
3. The *Gas Station* problem [53] simulates a self-serve gas station. The gas station consists of two pumps, an operator, and three customers. *Property*: A charge is made eventually after a customer has started pumping.

Measurement results The results of our experiments are shown in Table 1. The *Test case* column indicates the test case model corresponding to the measurements.

The *smart* and *root* sub-columns denote the measurement for the smart reduction and root reduction approaches, respectively.

In the *Running time (sec.)* column the running time until completion of the experiment is shown in seconds. Indicated in bold are the shortest running times comparing the *smart* and *root* sub-columns. The maximum running time of an experiment was set to 80 hours, after which the experiment was discontinued (indicated with $-$).

The columns *Max. #states* and *Max. #transitions* show the largest number of states and transitions, respectively, generated during the experiment. The best result, comparing smart and root reduction, is indicated in bold.

Table 1
Experiments: smart reduction vs. root reduction.

Test case	Running time (sec.)		Max. #states		Max. #transitions		Reduced #states	Reduced #transitions
	smart	root	smart	root	smart	root		
1394	14.41	**8.25**	**102,983**	198,692	**187,714**	355,338	1	1
1394'	**47.51**	460.53	**2,832,074**	36,855,184	**5,578,078**	96,553,318	1	1
ACS	70.87	**11.22**	**1,854**	4,764	**4,760**	14,760	29	61
Anderson	26.56	**15.42**	**153,664**	384,104	**2,118,368**	5,892,964	1	1
Anderson'	**373.56**	1852.42	**15,116,544**	56,250,000	**268,738,560**	1,188,000,000	1	1
Cache	20,55	**7.84**	616	616	4,631	4,631	1	1
Chiron	22.76	**13.66**	**317,115**	481,140	**2,563,650**	3,456,675	216	1286
Chiron'	**1,171.06**	1,236.06	**49,076,280**	56,293,380	**467,536,860**	513,857,520	2376	17,974
DES	**54.61**	948.66	**1,404**	64,498,297	**3,510**	518,438,860	1	1
Erat. Sieve	63.00	**8.43**	1,156,781	**234**	2,891,692	**406**	3	2
Erat. Sieve'	–	**10.64**	–	**865**	**–**	**2,012**	3	2
Gas Station	**325.10**	362.31	**11,042,816**	11,436,032	**84,254,720**	87,105,536	432	5616
HAVi-LE	**114.27**	493.01	**970,772**	15,688,570	**5,803,552**	80,686,289	131,873	644,695
HAVi-LE'	**93.08**	5,255.56	**453,124**	190,208,728	**2,534,371**	876,008,628	159,318	849,227
Le Lann	**96.35**	5,599.15	**12,083**	160,025,986	**701,916**	944,322,648	83,502	501,573
ODP	32.90	**9.97**	**10,397**	91,394	**87,936**	641,226	432	2,268
Peterson	**63.04**	–	**9**	–	**139**	–	9	22
Transit	**25.50**	59.69	**22,928**	3,763,192	**132,712**	39,925,524	636	3,188
Wafer Stepper	74.18	**57.54**	**962,122**	3,772,753	**4,537,240**	16,977,692	905,955	4,095,389

The number of states and transitions after minimisation is shown in the *Reduced #states* and *Reduced #transitions* columns, respectively.

Discussion In terms of running time smart reduction performs best for ten out of nineteen models, whereas root reduction performs best in eight of the models. For *Wafer stepper* and *Transit* smart reduction is only a few seconds to a minute faster than root reduction. The gain is a few minutes for *1394'*, *DES*, and *HAVi-LE*. For *HAVi-LE'*, *Le Lann*, and *Peterson* smart reduction is several hours faster. In general, the smart reduction approach performs better for large models where the state space can be reduced significantly before the final composition.

Root reduction performs best in relatively small models; *1394*, *ACS*, *Anderson*, *Cache*, *Chiron* and *ODP*. However, for these cases the difference in running times is negligible. Only the *Erat. Sieve'* model is minimised significantly faster by root reduction. For smaller models the overhead of the smart reduction heuristic is too high to obtain any benefits from the nominated ordering.

In terms of state space, smart reduction is the clear winner in all cases, with the exception of the *Erat. Sieve* models. Indeed, smart reduction performs particularly badly for the *Erat. Sieve'* model. The model consists of a pipeline where data is being pushed from one end to another. While the data domain considered by the nodes in the pipeline consists of 32 elements, in the minimised state space only one element remains. As synchronising actions may not be hidden in the local process LTSs, incremental composition and minimisation leads to a state space that is several orders of magnitude larger than the final state space.

The efficiency of smart reduction seems to increase when it is more successful in keeping compositions small. For instance, *Transit* and *Wafer stepper* have a similar number of states before reduction. However, when minimising the *Transit* model the smart reduction approach is able to reduce the number of states much more significantly than in the *Wafer stepper* model (see the *Max. #states smart* column).

Lessons learned In summary, the following lessons can be learned from this experiment:

- The overhead of applying its heuristics makes smart reduction less efficient in terms of running time when applied on small models.
- In general, smart reduction produces significantly smaller state spaces, especially for larger models, compared to other approaches.
- The data domain can have a significant impact on the effectiveness of smart reduction. In particular, this is the case when data can only be eliminated at a late stage of the compositional minimisation (as demonstrated by *Erat. Sieve*).
- The efficiency of smart reduction seems to increase when it is more successful in keeping compositions small.

Threats to validity The following threats to validity must be considered when interpreting the results:

- Only one tool has been involved to conduct the experiment, hence the results may be implementation specific. This only affects measured running times, as the state spaces produced during compositional minimisation is deterministic with respect to the composition order.

- A more controlled experiment needs to be considered in order to extrapolate the results of this experiment. In particular, the effect of action hiding, number of processes, and the chosen composition order should be controlled to determine correlation of these aspects with running time and size of the state space.
- The study only considers the DPBB equivalence as minimisation relation. Results may vary depending on the chosen equivalence relation. The DPBB equivalence is the strongest equivalence relation offered by CADP that still allows abstraction. Thus, the expectation is that other relations show equal or better performance improvements.
- The number and variety of case studies is still too limited for generalising claims on the effectiveness of smart reduction in comparison with root reduction.

9. Conclusions

In this article we have shown that DPBB is a congruence for parallel composition of LTS networks, in which there is synchronisation between LTSs on given label combinations. Because of this, DPBB may be used to reduce components in the compositional verification of LTS networks, and incrementally construct the state space of an LTS network. It had already been shown that compositional verification of LTS networks is adequate for safety properties, as this follows from the fact that branching bisimilarity is a congruence for the parallel composition of synchronising LTS networks [18]. As DPBB preserves both safety and liveness properties, this article proves that compositional verification can be used to verify liveness properties as well.

Furthermore, we have discussed how to safely decompose an LTS network in the case where verification has to start from the system as a whole. Both the composition and consistent decomposition of LTS networks preserve the admissibility property of LTS networks. Hence, the composition operator remains compatible with the compositional verification approaches for LTS networks described by [23].

We have shown that parallel composition of LTS networks with synchronisation on the common alphabet of the components is associative and commutative. From this it follows that DPBB is also a congruence for LTS networks as defined by Garavel, Lang, and Mateescu [23] if the set of synchronisation laws implements synchronisation on the common alphabet. Subsequently, we have shown that the requirement to synchronise on the common alphabet is unnecessarily restrictive. This has been shown in a direct proof of DPBB being a congruence for LTS networks.

All proofs in this work, except for the two in Section 7, have been mechanically verified using the Coq proof assistant[4] and are available online.[5]

Although our work focuses on the composition of LTS networks, the results are also applicable on the composition of individual LTSs. Our parallel composition operator $\|_\sigma$ subsumes the usual parallel composition operators of standard process algebra languages such as CCS [13], CSP [54], MCRL2 [16], and LOTOS [17].

Finally, we have run a set of experiments to compare compositional and traditional DPBB reduction. The compositional approach applies CADP's smart reduction employing a heuristic to determine an efficient compositional reduction order. The traditional reduction generates the complete state space before applying reduction. The compositional approach performed better in the medium when applied on large models for which the intermediate state space can be kept small.

Future work This work has been inspired by an approach for the compositional verification of transformations of LTS networks [55,36,56–58]. We would like to apply the results of this article to the improved transformation verification algorithm [55], thus guaranteeing its correctness for the compositional verification of transformations of LTS networks.

In future experiments, we would like to involve recent advancements in the computation of branching bisimulation, and therefore also DPBB, both sequentially [59,60] and in parallel on graphics processors [61], and we may consider other equivalence relations, such as simulation equivalence [62]. It will be interesting to measure the effect of applying these algorithms to compositionally solve a model checking problem.

Finally, we can consider various extensions of the LTS network formalism. For instance, by encoding timing in the LTSs, it is possible to reason about timed system behaviour. Combining results such as those in [63–66] with our results would allow to compositionally reason about timed behaviour. Other possibilities are to distinguish *must* and *may* transitions, and explicitly involve data variables, for instance as is suggested in [67]. We plan to investigate this further.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] E.M. Clarke, O. Grumberg, D. Peled, Model Checking, The MIT Press, 1999.

[4] https://coq.inria.fr.

[5] http://www.win.tue.nl/mdse/composition/DPBB_is_a_congruence_for_synchronizing_LTSs.zip.

[2] C. Baier, J.-P. Katoen, Principles of Model Checking, The MIT Press, 2008.
[3] D. Peled, Ten years of partial order reduction, in: CAV, vol. 1427, Springer, Berlin, Heidelberg, 1998, pp. 17–28.
[4] E.M. Clarke, E.A. Emerson, S. Jha, A.P. Sistla, Symmetry reductions in model checking, in: CAV, vol. 1427, Springer, Berlin, Heidelberg, 1998, pp. 147–158.
[5] E.M. Clarke, D.E. Long, K.L. McMillan, Compositional model checking, in: LICS, IEEE Computer Society Press, 1989, pp. 353–362.
[6] H.R. Andersen, Partial model checking, in: LICS, IEEE Computer Society Press, 1995, pp. 398–407.
[7] H.R. Andersen, Partial model checking of modal equations: a survey, Int. J. Softw. Tools Technol. Transf. 2 (3) (1999) 242–259.
[8] D. Kozen, Results on the propositional μ-calculus, Theor. Comput. Sci. 27 (1983) 333–354.
[9] R.J. van Glabbeek, W.P. Weijland, Branching time and abstraction in bisimulation semantics, J. ACM 43 (3) (1996) 555–600.
[10] R.J. van Glabbeek, S.P. Luttik, N. Trčka, Branching bisimilarity with explicit divergence, Fundam. Inform. 93 (4) (2009) 371–392.
[11] R. Mateescu, A.J. Wijs, Property-dependent reductions adequate with divergence-sensitive branching bisimilarity, Sci. Comput. Program. 96 (3) (2014) 354–376.
[12] R.J. van Glabbeek, S.P. Luttik, N. Trčka, Computation tree logic with deadlock detection, Log. Methods Comput. Sci. 5 (4) (2009) 1–24.
[13] R. Milner, Communication and Concurrency, Prentice-Hall, 1989.
[14] C.A.R. Hoare, Communicating Sequential Processes, Prentice Hall, 1985.
[15] J.A. Bergstra, J.W. Klop, Process algebra for synchronous communication, Inf. Control 60 (1/3) (1984) 109–137.
[16] O. Bunte, J. Groote, J. Keiren, M. Laveaux, T. Neele, E. de Vink, W. Wesselink, A. Wijs, T. Willemse, The mCRL2 toolset for analysing concurrent systems: improvements in expressivity and usability, in: TACAS, Part II, in: LNCS, vol. 11428, Springer, 2019, pp. 21–39.
[17] ISO/IEC, LOTOS — a formal description technique based on the temporal ordering of observational behaviour, in: International Standard 8807, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, 1989.
[18] B. Bloom, Structural operational semantics for weak bisimulations, Theor. Comput. Sci. 146 (1) (1995) 25–68.
[19] W. Fokkink, R.J. van Glabbeek, S.P. Luttik, Divide and congruence III: stability & divergence, in: CONCUR, in: LIPIcs, vol. 85, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, 15.
[20] F. Lang, Refined interfaces for compositional verification, in: FORTE, in: LNCS, vol. 4229, Springer, 2006, pp. 159–174.
[21] S. de Putter, A.J. Wijs, Compositional model checking is lively, in: FACS, in: LNCS, vol. 10487, Springer, 2017, pp. 117–136.
[22] Y. Bertot, P. Castéran, Interactive Theorem Proving and Program Development, Coq' Art: The Calculus of Inductive Constructions, Texts in Theoretical Computer Science, Springer, 2004.
[23] H. Garavel, F. Lang, R. Mateescu, Compositional verification of asynchronous concurrent systems using CADP, Acta Inform. 52 (4–5) (2015) 337–392.
[24] S. Cheung, D. Giannakopoulou, J. Kramer, Verification of liveness properties using compositional reachability analysis, in: ESEC/FSE, in: LNCS, vol. 1301, Springer, 1997, pp. 227–243.
[25] K. Elkader, O. Grumberg, C. Păsăreanu, S. Shoham, Automated circular assume-guarantee reasoning with N-way decomposition and alphabet refinement, in: CAV, Part I, in: LNCS, vol. 9779, Springer, 2016, pp. 329–351.
[26] A. Wijs, T. Neele, Compositional model checking with incremental counter-example construction, in: CAV, Part I, in: LNCS, vol. 10426, Springer, 2017, pp. 570–590.
[27] P. Crouzen, F. Lang, Smart reduction, in: FASE, in: LNCS, vol. 6603, Springer, 2011, pp. 111–126.
[28] F. Lang, Exp.open 2.0: a flexible tool integrating partial order, compositional, and on-the-fly verification methods, in: IFM, in: LNCS, vol. 3771, Springer, 2005, pp. 70–88.
[29] F. Lang, Unpublished PVS proof (by Jaco van de Pol) and textual proof showing that branching bisimulation is a congruence for networks of LTSs. This proof does not consider DPBB, Personal communication, 2016.
[30] L. Spaninks, An Axiomatisation for Rooted Branching Bisimulation with Explicit Divergence, Master's thesis, Eindhoven University of Technology, 2013.
[31] J.-P. Krimm, L. Mounier, Compositional state space generation from LOTOS programs, in: TACAS, vol. 1217, Springer, Berlin, Heidelberg, 1997, pp. 239–258.
[32] F. Maraninchi, Operational and compositional semantics of synchronous automaton compositions, in: CONCUR, in: LNCS, vol. 630, Springer, 1992, pp. 550–564.
[33] M. Mazzara, I. Lanese, Towards a unifying theory for web services composition, in: WS-FM'06, in: LNCS, vol. 4184, Springer, 2006, pp. 257–272.
[34] I. Ulidowski, I. Phillips, Ordered SOS process languages for branching and eager bisimulations, Inf. Comput. 178 (1) (2002) 180–213.
[35] C. Verhoef, A congruence theorem for structured operational semantics with predicates and negative premises, in: CONCUR, vol. 836, Springer, Berlin, Heidelberg, 1994, pp. 433–448.
[36] A.J. Wijs, Define, verify, refine: correct composition and transformation of concurrent system semantics, in: FACS, in: LNCS, vol. 8348, Springer, 2013, pp. 348–368.
[37] S. de Putter, A. Wijs, To compose, or not the compose, that is the question: an analysis of compositional state space generation, in: FM, in: LNCS, vol. 10951, Springer, 2018, pp. 485–504.
[38] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, H. Wijshoff, A medium-scale distributed system for computer science research: infrastructure for the long term, IEEE Comput. 49 (5) (2016) 54–63.
[39] R. Pelánek, BEEM: benchmarks for explicit model checkers, in: SPIN'07, in: LNCS, vol. 4595, Springer, 2007, pp. 263–267.
[40] J. Barnat, L. Brim, V. Havel, J. Havlíček, J. Kriho, M. Lenčo, P. Ročkai, V. Štill, J. Weiser, DiVinE 3.0 - an explicit-state model checker for multithreaded C & C++ programs, in: CAV, in: LNCS, vol. 8044, Springer, 2013, pp. 863–868.
[41] Laboratory for Advanced Software Engineering Research, Example repository for finite state verification tools, http://laser.cs.umass.edu/verification-examples/, 8 Jan. 2003, last accessed 20 Dec. 2017.
[42] S.P. Luttik, Description and Formal Specification of the Link Layer of P1394, Tech. Rep. SEN-R9706, CWI, 1997.
[43] B. Ploeger, Analysis of ACS Using mCRL2, Tech. Rep. 09-11, Eindhoven University of Technology, 2009.
[44] H. Kahlouche, C. Viho, M. Zendri, An industrial experiment in automatic generation of executable test suites for a cache coherency protocol, in: IWTCS, Kluwer, B.V., 1998, pp. 211–226.
[45] National Institute of Standards and Technology, Data Encryption Standard (DES), Federal Information Processing Standards 46-3, 1999.
[46] J. Romijn, Model Checking a HAVi Leader Election Protocol, Tech. Rep. SEN-R9915, CWI, 1999.
[47] H. Garavel, L. Mounier, Specification and Verification of Various Distributed Leader Election Algorithm for Unidirectional Ring Networks, Research Report RR-2986, INRIA, 1996.
[48] H. Garavel, M. Sighireanu, A graphical parallel composition operator for process algebras, in: FORTE/PSTV, in: IFIP Conference Proceedings, vol. 156, Kluwer, 1999, pp. 185–202.
[49] L. Mounier, A LOTOS specification of a "transit-node", Tech. Rep. SPECTRE 94-8, VERIMAG, 1994.
[50] G.L. Peterson, Myths about the mutual exclusion problem, Inf. Process. Lett. 12 (1981) 115–116.
[51] J.H. Anderson, Y.-J. Kim, T. Herman, Shared-memory mutual exclusion: major research trends since 1986, Distrib. Comput. 16 (2–3) (2003) 75–110.
[52] G.S. Avrunin, J.C. Corbett, M.B. Dwyer, C.S. Pasareanu, S.F. Siegel, Comparing finite-state verification techniques for concurrent software, Technical Report UM-CS-1999-069, University of Massachusetts, 1999.
[53] D. Heimbold, D. Luckham, Debugging Ada tasking programs, IEEE Softw. 2 (2) (1985) 47–57.

[54] A. Roscoe, The Theory and Practice of Concurrency, Prentice-Hall, 1998.
[55] S. de Putter, A.J. Wijs, Verifying a verifier: on the formal correctness of an LTS transformation verification technique, in: FASE, in: LNCS, vol. 9633, Springer, 2016, pp. 383–400.
[56] A.J. Wijs, Confluence detection for transformations of labelled transition systems, in: Proceedings of the 2nd Graphs as Models Workshop (GaM 2015), in: EPTCS, vol. 181, Open Publishing Association, 2015, pp. 1–15.
[57] A.J. Wijs, L.J.P. Engelen, Efficient property preservation checking of model refinements, in: TACAS, in: LNCS, vol. 7795, Springer, 2013, pp. 565–579.
[58] A.J. Wijs, L.J.P. Engelen, Refiner: towards formal verification of model transformations, in: NFM, in: LNCS, vol. 8430, Springer, 2014, pp. 258–263.
[59] J.F. Groote, A.J. Wijs, An $O(m \log n)$ algorithm for stuttering equivalence and branching bisimulation, in: TACAS, in: LNCS, vol. 9636, Springer, 2016, pp. 607–624.
[60] J.F. Groote, D.N. Jansen, J.J.A. Keiren, A.J. Wijs, An $O(m \log n)$ algorithm for computing stuttering equivalence and branching bisimulation, ACM Trans. Comput. Log. 18 (2) (2017) 13.
[61] A.J. Wijs, GPU accelerated strong and branching bisimilarity checking, in: TACAS, in: LNCS, vol. 9035, Springer, 2015, pp. 368–383.
[62] G. Cécé, Foundation for a series of efficient simulation algorithms, in: LICS, IEEE, 2017, pp. 1–12.
[63] A.J. Wijs, Achieving discrete relative timing with untimed process algebra, in: ICECCS, IEEE Computer Society Press, 2007, pp. 35–44.
[64] A.J. Wijs, W.J. Fokkink, From χ_t to μCRL: combining performance and functional analysis, in: ICECCS, IEEE Computer Society Press, 2005, pp. 184–193.
[65] W.J. Fokkink, J. Pang, A.J. Wijs, Is timed branching bisimilarity a congruence indeed?, Fundam. Inform. 87 (3–4) (2008) 287–311.
[66] A.J. Wijs, What to do next? Analysing and optimising system behaviour in time, Ph.D. thesis, Vrije Universiteit Amsterdam, 2007.
[67] S.S. Bauer, K.G. Larsen, A. Legay, U. Nyman, A. Wąsowski, A model specification theory for components with data, in: FACS, in: LNCS, vol. 7253, Springer, 2011, pp. 61–78.