Enabling Preserving Bisimulation Equivalence

Rob van Glabbeek
Data61, CSIRO, Sydney, Australia
Computer Science and Engineering, University of New South Wales, Sydney, Australia

Peter Höfner
School of Computing, Australian National University, Canberra, Australia
Data61, CSIRO, Sydney, Australia

Weiyou Wang
School of Computing, Australian National University, Canberra, Australia

Abstract

Most fairness assumptions used for verifying liveness properties are criticised for being too strong or unrealistic. On the other hand, justness, arguably the minimal fairness assumption required for the verification of liveness properties, is not preserved by classical semantic equivalences, such as strong bisimilarity. To overcome this deficiency, we introduce a finer alternative to strong bisimilarity, called enabling preserving bisimilarity. We prove that this equivalence is justness-preserving and a congruence for all standard operators, including parallel composition.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Process calculi; Theory of computation → Distributed computing models

Keywords and phrases bisimilarity, liveness properties, fairness assumptions, process algebra

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.33

Related Version *Full Version*: `http://arxiv.org/abs/2108.00142` [15]

Acknowledgements We are grateful to Filippo De Bortoli.

1 Introduction

Formal verification of concurrent systems becomes more and more standard practice, in particular in safety-critical environments. Progress and fairness assumptions have to be used when verifying liveness properties, which guarantee that "something good will eventually happen". Without assumptions of this kind, no meaningful liveness property can formally be proven.

▶ **Example 1.** Consider the program `while(true) do x:=x+1 od` with `x` initialised to `0`. Intuitively, any liveness property of the form "eventually `x=n`" should be satisfied by the program. However, these properties are valid only when assuming *progress*, stating that a system will make progress when it can; otherwise the program could just stop after some computation. ⌟

Progress itself is not a strong enough assumption when concurrent systems are verified, for a system of multiple completely independent components makes progress as long as one of its components makes progress, even when others do not. For decades, researchers have developed notions of fairness and used them in both system specification and verification; the most common ones are surveyed in [13]. Two of the most popular fairness assumptions are weak and strong fairness of instructions [5].[1] They apply to systems whose behaviour is specified by some kind of code, composed out of *instructions*. A *task* is any activity of the system that stems from a particular instruction; it is *enabled* when the system is ready to do

[1] Often these notions are referred to as weak and strong fairness without mentioning instructions; here, we follow the terminology of [13], which is more precise.

some part of that task, and *executed* when the system performs some part of it. Now weak and strong fairness of instructions state that whenever a task is enabled persistently (for weak fairness) or infinitely often (for strong fairness), then it will be executed by the system. These fairness assumptions, as well as all others surveyed in [13],[2] imply progress.

Despite being commonly used, it has been argued that most fairness assumptions, including weak and strong fairness of instructions, are often too strong or unrealistic, "in the sense that run-of-the-mill implementations tend not to be fair" [13].

(Reactive) systems are often described by *labelled transition systems*, which model all activities as transitions going from state to state, labelled with *actions*. Some actions require synchronisation of the modelled system with its environment; they can occur only when both the system and the environment are ready to engage in this action. Such actions, and the transitions labelled with them, are called *blocking*.

▶ **Example 2.** Assume that every morning Alice has a choice between a slice of bread with jam or a bacon and egg roll. A corresponding transition system consists of one state with two transitions, each standing for one kind of breakfast. Both weak and strong fairness (of instructions) will force Alice to eventually have both types of breakfast, ruling out the possibility that Alice picks up jam every day as she is a vegetarian. ⌟

To address this issue, a weaker assumption, called justness, has been proposed. It has been formulated for reactive systems, modelled as labelled transition systems. *Justness* is the assumption that

> *Once a non-blocking transition is enabled that stems from a set of parallel components, one (or more) of these components will eventually partake in a transition. [13]*

Example 2 features only one component, Alice. Assuming justness, as expected, she now has the option to eat jam for the rest of her life. Let us now look at a more technical example.

▶ **Example 3.** We consider the following two programs, and assume that all variables are initialised by `0`.

```
while(true) do                          while(true) do  ||  x := 1;
  choose                                  y := y+1;     ||
    if true then y := y+1;                od            ||
    if x = 0 then x := 1;
  end
od
```

The example on the left presents an infinite loop containing an internal nondeterministic choice. The conditional write `if x = 0 then x := 1` describes an atomic read-modify-write (RMW) operation[3]. Such operators, supported by modern hardware, read a memory location and simultaneously write a new value into it. This example is similar to Example 2 in the sense that the liveness property "eventually `x=1`" should not be satisfied as the program has a choice every time the loop body is executed.

The example on the right-hand side is similar, but the handling of variables `x` and `y` are managed by different components. As the two programs are independent from each other – they could be executed on different machines – the property "eventually `x=1`" should hold.

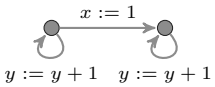
Justness differentiates these behaviour, whereas weak and strong fairness fail to do so. ⌟

[2] Many other notions of fairness are obtained by varying the definition of task. In *fairness of components* a task refers to all activity stemming from a component of a system that is a parallel composition.

[3] `https://en.wikipedia.org/wiki/Read-modify-write`

The above example illustrates that standard notions of fairness are regularly too strong, and the notion of justness may be a good replacement. When it comes to verification tasks, semantic equivalences, such as strong bisimilarity [18], are a standard tool to reduce the state space of the systems under consideration. Unfortunately, these semantic equivalences do not accord well with justness. The problem is that they are based on labelled transition systems, which do not capture the concept of components. The different behaviour of the two programs in Example 3 stems from the components involved. In fact, both programs give rise to the same transition system, depicted below. Systems featuring the same transition system cannot be distinguished by any semantic equivalence found in the literature. Consequently, the verification of the stated liveness property will fail for one of the two programs of Example 3.

To overcome this deficiency, we introduce *enabling preserving bisimilarity*, a finer alternative to strong bisimilarity, which respects justness. It is based on extended labelled transition systems that take components involved in particular transitions into account.



2 Labelled Transition Systems with Successors

As discussed in the introduction, one reason why strong bisimilarity does not preserve liveness properties under justness is that necessary information is missing, namely components.

The definition of *(parallel) components* was based on the parallel composition operator in process algebras when justness was first introduced in [4, 12], and has been generalised in later work to allow the use of justness in different contexts.

Here we define a justness-preserving semantic equivalence on an extension of labelled transition systems. Using labelled transition systems rather than process algebra as underlying concept makes our approach more general, for other models of concurrency, such as Petri nets or higher-dimensional automata, induce a semantics based on transition systems as well.

The essence of justness is that when a non-blocking transition t is enabled in a state s, eventually the system must perform an action u that *interferes* with it [13], notation $t \not\frown u$, in the sense that a component affected by u is necessary for the execution of t – or, to be more precise, for the variant t' of t that is enabled after the system has executed some actions that do not interfere with t. The present paper abstracts from the notion of component, but formalises justness, as well as our enabling preserving bisimilarity, in terms of a *successor relation* $t \rightsquigarrow_\pi t'$, marking t' as a successor of t, parametrised with the noninterfering actions π happening in between. This relation also encodes the above relation $\not\frown$. The advantage of this approach over one that uses components explicitly, is that it also applies to models like higher-dimensional automata [21, 6, 16, 8] in which the notion of a component is more fluid, and changes during execution.

A *labelled transition system* (LTS) is a tuple $(S, Tr, source, target, \ell)$ with S and Tr sets (of *states* and *transitions*), $source, target : Tr \to S$ and $\ell : Tr \to \mathscr{L}$, for some set $\mathscr{L}$ of transition labels. A transition $t \in Tr$ of an LTS is *enabled* in a state $p \in S$ if $source(t) = p$. Let $en(p)$ be the set of transitions that are enabled in p.

A *path* in an LTS is an alternating sequence $p_0\, u_0\, p_1\, u_1\, p_2 \ldots$ of states and transitions, starting with a state and either being infinite or ending with a state, such that $source(u_i) = p_i$ and $target(u_i) = p_{i+1}$ for all relevant i. The *length* $l(\pi) \in \mathbb{N} \cup \{\infty\}$ of a path π is the number of transitions in it. If π is a path, then $\hat{\pi}$ is the sequence of transitions occurring in π.

▶ **Definition 4** (LTSS). A *labelled transition system with successors* (LTSS) is a tuple $(S, Tr, source, target, \ell, \rightsquigarrow)$ with $(S, Tr, source, target, \ell)$ an LTS, and $\rightsquigarrow \subseteq Tr \times Tr \times Tr$, the *successor relation*, such that if $(t, u, v) \in \rightsquigarrow$ then $source(t) = source(u)$ and $source(v) = target(u)$. We write $t \rightsquigarrow_u v$ for $(t, u, v) \in \rightsquigarrow$.

We use this successor relation to define the concept of (un)affected transitions. Let two transitions t and u be enabled in a state p, i.e., $t, u \in en(p)$ for some $p \in S$; the *concurrency relation* $\smile\!\!\bullet$ is defined as $t \smile\!\!\bullet u :\Leftrightarrow \exists\, v.\ t \rightsquigarrow_u v$. Its negation $t \not\smile\!\!\bullet u$ says that the possible occurrence of t is *affected* by the occurrence of u. In case t is unaffected by u (i.e., $t \smile\!\!\bullet u$), each v with $t \rightsquigarrow_u v$ denotes a variant of t that can occur after u. Note that the concurrency relation can be asymmetric. Examples are traffic lights – a car passing traffic lights should be affected by them, but the lights do not care whether the car is there; and read-write operations – reading shared memory can be affected by a write action, but, depending on how the memory is implemented, the opposite might not hold. In case t and u are mutually unaffected we write $t \smile u$, i.e., $t \smile u :\Leftrightarrow t \smile\!\!\bullet u \wedge u \smile\!\!\bullet t$.

It is possible to have $t \smile\!\!\bullet t$, namely when executing transition t does not disable (a future variant of) t to occur again. This can happen when t is a signal emission, say of a traffic light shining red, for even after shining for some time it keeps on shining; or when t is a broadcast receive action, for receiving a broadcast does not invalidate a system's perpetual readiness to again accept a broadcast, either by receiving or ignoring it.

▶ **Example 5.** Consider the labelled transition system of Example 3, let t_1 and t_2 be the two transitions corresponding to `y:=y+1` in the first and second state, respectively, and let u be the transition for assignment `x:=1`. The assignments of `x` and `y` in the right-hand program are independent, hence $t_1 \rightsquigarrow_u t_2$, $u \rightsquigarrow_{t_1} u$ and $t_1 \smile u$.

For the program on the left-hand side, the situation is different. As the instructions stem from the same component (program), all transitions affect each other, i.e., $\rightsquigarrow = \smile\!\!\bullet = \emptyset$. ⌟

The successor relation relates transitions one step apart. We lift it to sequences of transitions.

▶ **Definition 6** (Successor along Path). The relation $\rightsquigarrow$ is extended to $\rightsquigarrow \subseteq Tr \times Tr^* \times Tr$ by (i) $t \rightsquigarrow_\varepsilon w$ iff $w = t$, and (ii) $t \rightsquigarrow_{\pi u} w$ iff there is a v with $t \rightsquigarrow_\pi v$ and $v \rightsquigarrow_u w$.

Here, ε denotes the empty sequence, and πu the sequence π followed by transition u. We define a concurrency relation $\smile\!\!\bullet \subseteq Tr \times Tr^*$ considering sequences of transitions by $t \smile\!\!\bullet \pi :\Leftrightarrow \exists\, v.\ t \rightsquigarrow_\pi v$. Intuitively, $t \smile\!\!\bullet \pi$ means that there exists a successor of t after π has been executed. Thus, t is unaffected by all transitions of π.

We are ready to define justness, which is parametrised by a set B of blocking actions.

▶ **Definition 7** (Justness). Given an LTSS $= (S, Tr, source, target, \ell, \rightsquigarrow)$ labelled over $\mathscr{L}$, and $B \subseteq \mathscr{L}$, a path π in is B-*just* if for each suffix π_0 of π and for each transition $t \in Tr^\bullet$ with $\ell(t) \notin B$ and $source(t)$ the first state of π_0, the path π_0 has a finite prefix ρ such that $t \not\smile\!\!\bullet \hat{\rho}$. Here $Tr^\bullet := \{t \in Tr \mid t \not\smile\!\!\bullet t\}$.

3 Enabling Preserving Bisimulation Equivalence

In this section we introduce enabling preserving bisimulation equivalence, and show how it preserves justness. In contrast to classical bisimulations, which are relations of type $S \times S$, the new equivalence is based on triples. The essence of justness is that a transition t enabled in a state s must eventually be affected by the sequence π of transitions the system performs. As long as π does not interfere with t, we obtain a transition t' with $t \rightsquigarrow_\pi t'$. This transition

t' represents the interests of t, and must eventually be affected by an extension of π. Here, executing t or t' as part of this extension is a valid way of interfering. To create a bisimulation that respects such considerations, for each related pair of states p and q we also match each enabled transition of p with one of q, and vice versa. These relations are maintained during the evolution of the two related systems, so that when one system finally interferes with a descendant of t, the related system interferes with the related descendant.

▶ **Definition 8** (Ep-bisimilarity). Given an LTSS $(S, Tr, source, target, \ell, \rightsquigarrow)$, an *enabling preserving bisimulation* (*ep-bisimulation*) is a relation $\mathscr{R} \subseteq S \times S \times \mathscr{P}(Tr \times Tr)$ satisfying

1. if $(p, q, R) \in \mathscr{R}$ then $R \subseteq en(p) \times en(q)$ such that
   - **a.** $\forall t \in en(p).\ \exists\, u \in en(q).\ t\ R\ u$,
   - **b.** $\forall u \in en(q).\ \exists\, t \in en(p).\ t\ R\ u$,
   - **c.** if $t\ R\ u$ then $\ell(t) = \ell(u)$, and
2. if $(p, q, R) \in \mathscr{R}$ and $v\ R\ w$, then $(target(v), target(w), R') \in \mathscr{R}$ for some R' such that
   - **a.** if $t\ R\ u$ and $t \rightsquigarrow_v t'$, then $\exists\, u'.\ u \rightsquigarrow_w u' \wedge t'\ R'\ u'$, and
   - **b.** if $t\ R\ u$ and $u \rightsquigarrow_w u'$, then $\exists\, t'.\ t \rightsquigarrow_v t' \wedge t'\ R'\ u'$.

Two states p and q in an LTSS are *enabling preserving bisimilar* (ep-bisimilar), $p \leftrightarrows_{ep} q$, if there exists an enabling preserving bisimulation $\mathscr{R}$ such that $(p, q, R) \in \mathscr{R}$ for some R.

Definition 8 without Items 2a and 2b is nothing else than a reformulation of the classical definition of strong bisimilarity. An ep-bisimulation additionally maintains for each pair of related states p and q a relation R between the transitions enabled in p and q. Items 2a and 2b strengthen the condition on related target states by requiring that the successors of related transitions are again related relative to these target states. It is this requirement (and in particular its implication stated in the following observation) which distinguishes the transition systems for Example 3.

▶ **Observation 9.** Ep-bisimilarity respects the concurrency relation.
For a given ep-bisimulation $\mathscr{R}$, if $(p, q, R) \in \mathscr{R}$, $t\ R\ u$ and $v\ R\ w$ then $t \smile\!\!\bullet\ v$ iff $u \smile\!\!\bullet\ w$.

▶ **Proposition 10.** $\leftrightarrows_{ep}$ is an equivalence relation.

Proof. *Reflexivity*: Let $(S, Tr, source, target, \ell, \rightsquigarrow)$ be an LTSS. The relation

$$\mathscr{R}_{Id} := \{(s, s, Id_s) \mid s \in S\}$$

is an ep-bisimulation. Here $Id_s := \{(t, t) \mid t \in en(s)\}$.
Symmetry: For a given ep-bisimulation $\mathscr{R}$, the relation

$$\mathscr{R}^{-1} := \{(q, p, R^{-1}) \mid (p, q, R) \in \mathscr{R}\}$$

is also an ep-bisimulation. Here $R^{-1} := \{(u, t) \mid (t, u) \in R\}$.
Transitivity: For given ep-bisimulations $\mathscr{R}_1$ and $\mathscr{R}_2$, the relation

$$\mathscr{R}_1; \mathscr{R}_2 := \{(p, r, R_1; R_2) \mid \exists\, q.\ (p, q, R_1) \in \mathscr{R}_1 \wedge (q, r, R_2) \in \mathscr{R}_2\}$$

is also an ep-bisimulation. Here $R_1; R_2 := \{(t, v) \mid \exists\, u.\ (t, u) \in R_1 \wedge (u, v) \in R_2\}$. ◀

▶ **Observation 11.** The union of any collection of ep-bisimulations is itself an ep-bisimulation.

Consequently there exists a largest ep-bisimulation.

Before proving that ep-bisimilarity preserves justness, we lift this relation to paths.

▶ **Definition 12** (Ep-bisimilarity of Paths). Given an ep-bisimulation $\mathscr{R}$ and two paths $\pi = p_0\, u_0\, p_1\, u_1\, p_2 \ldots$ and $\pi' = p'_0\, u'_0\, p'_1\, u'_1\, p'_2 \ldots$, we write $\pi\ \mathscr{R}\ \pi'$ iff $l(\pi) = l(\pi')$, and there exists $R_i \subseteq Tr \times Tr$ for all $i \in \mathbb{N}$ with $i \leq l(\pi)$, such that

1. $(p_i, p'_i, R_i) \in \mathscr{R}$ for each $i \in \mathbb{N}$ with $i \leq l(\pi)$,
2. $u_i\ R_i\ u'_i$ for each $i < l(\pi)$,
3. if $t\ R_i\ t'$ and $t \rightsquigarrow_{u_i} v$ with $i < l(\pi)$, then $\exists\, v'.\ t' \rightsquigarrow_{u'_i} v' \wedge v\ R_{i+1}\ v'$, and
4. if $t\ R_i\ t'$ and $t' \rightsquigarrow_{u'_i} v'$ with $i < l(\pi)$, then $\exists\, v.\ t \rightsquigarrow_{u_i} v \wedge v\ R_{i+1}\ v'$.

Paths π and π' are *enabling preserving bisimilar*, notation $\pi \leftrightarrows_{ep} \pi'$, if there exists an ep-bisimulation $\mathscr{R}$ with $\pi\ \mathscr{R}\ \pi'$. If $\pi \leftrightarrows_{ep} \pi'$, we also write $\pi\ \vec{R}\ \pi'$ if $\vec{R} := (R_0, R_1, \ldots)$ are the R_i required above.

Note that if $p \leftrightarrows_{ep} q$ and π is any path starting from p, then, by Definition 8, there is a path π' starting from q with $\pi \leftrightarrows_{ep} \pi'$. The following lemma lifts Observation 9.

▶ **Lemma 13.** If $\pi\ \vec{R}\ \rho$ with π finite and $t\ R_0\ t'$ then $t \smile\!\!\bullet\ \hat{\pi}$ iff $t' \smile\!\!\bullet\ \hat{\rho}$.

Proof. We have to show that $\exists v.\ t \rightsquigarrow_{\hat{\pi}} v$ iff $\exists v'.\ t' \rightsquigarrow_{\hat{\rho}} v'$. Using symmetry, we may restrict attention to the "only if" direction. We prove a slightly stronger statement, namely that for every transition v with $t \rightsquigarrow_{\hat{\pi}} v$ there exists a v' such that $t' \rightsquigarrow_{\hat{\rho}} v'$ and $v\ R_{l(\pi)}\ v'$.

We proceed by induction on the length of π.

The base case, where $l(\pi) = 0$, $\hat{\pi} = \varepsilon$ and thus $v = t$, holds trivially, taking $v' := t'$.

So assume $\pi u p\ \vec{R}\ \rho u' p'$ and $t \rightsquigarrow_{\hat{\pi}u} w$. Then there is a v with $t \rightsquigarrow_{\hat{\pi}} v$ and $v \rightsquigarrow_u w$. By induction there is a transition v' such that $t' \rightsquigarrow_{\hat{\rho}} v'$ and $v\ R_{l(\pi)}\ v'$. By Definition 12(3), there is a w' with $v' \rightsquigarrow_{u'} w'$ and $w\ R_{l(\pi)+1}\ w'$. Thus $t' \rightsquigarrow_{\hat{\rho}u'} w'$ by Definition 6. ◀

▶ **Theorem 14.** Ep-bisimilarity preserves justness: Given two paths π and π' in an LTSS with $\pi \leftrightarrows_{ep} \pi'$, and a set B of blocking actions, then π is B-just iff π' is B-just.

Proof. Let $\pi = p_0\, u_0\, p_1\, u_1\, p_2 \ldots$ and $\pi' = p'_0\, u'_0\, p'_1\, u'_1\, p'_2 \ldots$. Suppose π is B-unjust, so there exist an $i \in \mathbb{N}$ with $i \leq l(\pi)$ and a transition $t \in Tr^\bullet$ with $\ell(t) \notin B$ and $source(t) = p_i$ such that $t \smile\!\!\bullet\ \hat{\rho}$ for each finite prefix ρ of the suffix $p_i\, u_i\, p_{i+1}\, u_{i+1}\, p_{i+2} \ldots$ of π. It suffices to show that also π' is B-unjust.

Take an ep-bisimulation $\mathscr{R}$ such that $\pi\ \mathscr{R}\ \pi'$. Choose $R_i \subseteq Tr \times Tr$ for all $i \in \mathbb{N}$ with $i \leq l(\pi)$, satisfying the four conditions of Definition 12. Pick any $t' \in Tr$ with $t\ R_i\ t'$ – such a t' must exist by Definition 8. Now $source(t') = p'_i$. By Definition 8, $t' \in Tr^\bullet$ and $\ell(t') = \ell(t) \notin B$. It remains to show that $t' \smile\!\!\bullet\ \hat{\rho}'$ for each finite prefix ρ' of the suffix $p'_i\, u'_i\, p'_{i+1}\, u'_{i+1}\, p'_{i+2} \ldots$ of π'. This follows by Lemma 13. ◀

4 Stating and Verifying Liveness Properties

The main purpose of ep-bisimilarity is as a vehicle for proving liveness properties. A *liveness property* is any property saying that eventually something good will happen [17]. Liveness properties are *linear-time properties*, in the sense that they are interpreted primarily on the (complete) *runs* of a system. When a distributed system is formalised as a state in an (extended) LTS, a run of the distributed system is modelled as a path in the transition system, starting from that state. However, not every such path models a realistic system run. A *completeness criterion* [9] selects some of the paths of a system as *complete paths*, namely those that model runs of the represented system.

A state s in an (extended) LTS is said to satisfy a linear time property φ when employing the completeness criterion CC, notation $s \models^{CC} \varphi$, if each complete run of s satisfies φ [10]. Writing $\pi \models \varphi$ when property φ holds for path π, we thus have $s \models^{CC} \varphi$ iff $\pi \models \varphi$ for all

complete paths π starting from s. When simplifying a system s into an equivalent system s', so that $s \sim s'$ for some equivalence relation $\sim$, it is important that judgements $\models^{CC} \varphi$ are preserved:

$$s \sim s' \quad \Rightarrow \quad (s \models^{CC} \varphi \Leftrightarrow s' \models^{CC} \varphi).$$

This is guaranteed when for each path π of s there exists a path π' of s', such that (a) $\pi \models \varphi$ iff $\pi' \models \varphi$, and (b) π' is complete iff π is complete. Here (a) is already guaranteed when π and π' are related by strong bisimilarity. Taking CC to be B-justness for any classification of a set of actions B as blocking, and $\sim$ to be $\leftrightarrows_{ep}$, Theorem 14 now ensures (b) as well.

5 Interpreting Justness in Process Algebras

Rather than using LTSs directly to model distributed systems, one usually employs other formalisms such as process algebras or Petri nets, for they are often easier to use for system modelling. Their formal semantics maps their syntax into states of LTSs. In this section we introduce the *Algebra of Broadcast Communication with discards and Emissions* (ABCdE), an extension of Milner's *Calculus of Communication Systems* (CCS) [18] with broadcast communication and signal emissions. In particular, we give a structural operational semantics [19] that interprets process expressions as states in an LTS. Subsequently, we define the successor relation $\rightsquigarrow$ for ABCdE, thereby enriching the LTS into an LTSS.

We use ABCdE here, as for many realistic applications CCS is not expressive enough [7, 11]. The presented approach can be applied to a wide range of process algebras. ABCdE is largely designed to be a starting point for transferring the presented theory to algebras used for "real" applications. For example, broadcast communication is needed for verifying routing protocols (e.g. [14]); signals are employed to correctly render and verify protocols for mutual exclusion [11, 3]. Another reason is that broadcasts as well as signals, in different ways, give rise to asymmetric concurrency relations, and we want to show that our approach is flexible enough to handle this.

5.1 Algebra of Broadcast Communication with Discards and Emissions

ABCdE is parametrised with sets $\mathscr{A}$ of *agent identifiers*, $\mathscr{C}$ of *handshake communication names*, $\mathscr{B}$ of *broadcast communication names*, and $\mathscr{S}$ of *signals*; each $A \in \mathscr{A}$ comes with a defining equation $A \stackrel{def}{=} P$ with P being a guarded ABCdE expression as defined below. $\bar{\mathscr{C}} := \{\bar{c} \mid c \in \mathscr{C}\}$ is the set of *handshake communication co-names*, and $\bar{\mathscr{S}} := \{\bar{s} \mid s \in \mathscr{S}\}$ is the set of signal emissions. The collections $\mathscr{B}!$, $\mathscr{B}?$, and $\mathscr{B}:$ of *broadcast*, *receive*, and *discard* actions are given by $\mathscr{B}\sharp := \{b\sharp \mid b \in \mathscr{B}\}$ for $\sharp \in \{!, ?, :\}$. $Act := \mathscr{C} \cup \bar{\mathscr{C}} \cup \{\tau\} \cup \mathscr{B}! \cup \mathscr{B}? \cup \mathscr{S}$ is the set of *actions*, where τ is a special *internal action*. $\mathscr{L} := Act \cup \mathscr{B}: \cup \bar{\mathscr{S}}$ is the set of *transition labels*. Complementation extends to $\mathscr{C} \cup \bar{\mathscr{C}} \cup \mathscr{S} \cup \bar{\mathscr{S}}$ by $\bar{\bar{c}} := c$.

Below, c ranges over $\mathscr{C} \cup \bar{\mathscr{C}} \cup \mathscr{S} \cup \bar{\mathscr{S}}$, η over $\mathscr{C} \cup \bar{\mathscr{C}} \cup \{\tau\} \cup \mathscr{S} \cup \bar{\mathscr{S}}$, α over Act, ℓ over $\mathscr{L}$, b over $\mathscr{B}$, $\sharp, \sharp_1, \sharp_2$ over $\{!, ?, :\}$ and s, r over $\mathscr{S}$. A *relabelling* is a function $f : (\mathscr{C} \to \mathscr{C}) \cup (\mathscr{B} \to \mathscr{B}) \cup (\mathscr{S} \to \mathscr{S})$; it extends to $\mathscr{L}$ by $f(\bar{c}) = \overline{f(c)}$, $f(\tau) := \tau$, and $f(b\sharp) = f(b)\sharp$.

The set $\mathbb{P}$ of ABCdE expressions or *processes* is the smallest set including:

$\mathbf{0}$		*inaction*	$\alpha.P$	for $\alpha \in Act$ and $P \in \mathbb{P}$	*action prefixing*
$P + Q$	for $P, Q \in \mathbb{P}$	*choice*	$P \mid Q$	for $P, Q \in \mathbb{P}$	*parallel composition*
$P \backslash L$	for $L \subseteq \mathscr{C} \cup \mathscr{S}$, $P \in \mathbb{P}$	*restriction*	$P[f]$	for f a relabelling, $P \in \mathbb{P}$	*relabelling*
A	for $A \in \mathscr{A}$	*agent identifier*	$P\hat{}s$	for $s \in \mathscr{S}$	*signalling*

Table 1 Structural operational semantics of ABCdE – Basic.

$\alpha.P \xrightarrow{\alpha} P$ (Act)	$\dfrac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$ (Sum-l)	$\dfrac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'}$ (Sum-r)
$\dfrac{P \xrightarrow{\eta} P'}{P\|Q \xrightarrow{\eta} P'\|Q}$ (Par-l)	$\dfrac{P \xrightarrow{c} P',\ Q \xrightarrow{\bar{c}} Q'}{P\|Q \xrightarrow{\tau} P'\|Q'}$ (Comm)	$\dfrac{Q \xrightarrow{\eta} Q'}{P\|Q \xrightarrow{\eta} P\|Q'}$ (Par-r)
$\dfrac{P \xrightarrow{\ell} P'}{P\backslash L \xrightarrow{\ell} P'\backslash L}\ (\ell \notin L \cup \overline{L})$ (Res)	$\dfrac{P \xrightarrow{\ell} P'}{P[f] \xrightarrow{f(\ell)} P'[f]}$ (Rel)	$\dfrac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'}\ (A \stackrel{def}{=} P)$ (Rec)

We abbreviate $\alpha.\mathbf{0}$ by α, and $P\backslash\{c\}$ by $P\backslash c$. An expression is guarded if each agent identifier occurs within the scope of a prefixing operator.

The semantics of ABCdE is given by the labelled transition relation $\rightarrow \subseteq \mathbb{P} \times \mathscr{L} \times \mathbb{P}$, where transitions $P \xrightarrow{\ell} Q$ are derived from the rules of Tables 1–3. Here $\overline{L} := \{\bar{c} \mid c \in L\}$.

Table 1 shows the basic operational semantics rules, identical to the ones of CCS [18]. The process $\alpha.P$ performs the action α first and subsequently acts as P. The choice operator $P+Q$ may act as either P or Q, depending on which of the processes is able to act at all. The parallel composition $P|Q$ executes an action η from P, an action η from Q, or in the case where P and Q can perform complementary actions c and $\bar{c}$, the process can perform a synchronisation, resulting in an internal action τ. The restriction operator $P\backslash L$ inhibits execution of the actions from L and their complements. The relabelling $P[f]$ acts like process P with all labels ℓ replaced by $f(\ell)$. Finally, an agent A can do the same actions as the body P of its defining equation. When we take $\mathscr{B} = \mathscr{S} := \emptyset$, only the rules of Table 1 matter, and ABCdE simplifies to CCS.

Table 2 augments CCS with a mechanism for broadcast communication. The rules are similar to the ones for the Calculus of Broadcasting Systems (CBS) [20]; they also appear in the process algebra ABC [12], a strict subalgebra of ABCdE. The Rule (Bro) presents the core of broadcast communication, where any broadcast-action $b!$ performed by a component in a parallel composition is guaranteed to be received by any other component that is ready to do so, i.e., in a state that admits a $b?$-transition. Since it is vital that the sender of a broadcast can always proceed with it, regardless of the state of other processes running in parallel, the process algebra features discard actions $b:$, in such a way that each process in any state can either receive a particular broadcast b, by performing the action $b?$, or discard it, by means of a $b:$, but not both. A broadcast transmission $b!$ can synchronise with either $b?$ or $b:$, and thus is never blocked by lack of a listening party. In order to ensure associativity of the parallel composition, one requires rule (Bro) to consider receipt at the same time ($\sharp_1 = \sharp_2 = ?$). The remaining four rules of Table 2 generate the discard-actions. The Rule (Dis-nil) allows the nil process (inaction) to discard any incoming message; in the same spirit (Dis-act) allows a

Table 2 Structural operational semantics of ABCdE – Broadcast.

$\mathbf{0} \xrightarrow{b:} \mathbf{0}$ (Dis-nil) $\qquad \alpha.P \xrightarrow{b:} \alpha.P\ (\alpha \neq b?)$ (Dis-act) $\qquad \dfrac{P \xrightarrow{b:} P',\ Q \xrightarrow{b:} Q'}{P+Q \xrightarrow{b:} P'+Q'}$ (Dis-sum)

$\dfrac{P \xrightarrow{b\sharp_1} P',\ Q \xrightarrow{b\sharp_2} Q'}{P|Q \xrightarrow{b\sharp} P'|Q'}(\sharp_1 \circ \sharp_2 = \sharp \neq \_)$ with

$\circ$	!	?	:
!	–	!	!
?	!	?	?
:	!	?	:

(Bro) $\qquad \dfrac{P \xrightarrow{b:} P'}{A \xrightarrow{b:} A}\ (A \stackrel{def}{=} P)$ (Dis-rec)

Table 3 Structural operational semantics of ABCdE – Signals.

$$P\hat{}s \xrightarrow{\bar{s}} P\hat{}s \text{ (Sig)} \qquad \frac{P \xrightarrow{\bar{s}} P'}{P+Q \xrightarrow{\bar{s}} P'+Q} \text{ (Sig-sum-l)} \qquad \frac{Q \xrightarrow{\bar{s}} Q'}{P+Q \xrightarrow{\bar{s}} P+Q'} \text{ (Sig-sum-r)}$$

$$\frac{P \xrightarrow{\bar{s}} P'}{P\hat{}r \xrightarrow{\bar{s}} P'\hat{}r} \text{ (Sig-sig)} \qquad \frac{P \xrightarrow{\bar{s}} P'}{A \xrightarrow{\bar{s}} A} \ (A \stackrel{def}{=} P) \text{ (Sig-rec)} \qquad \frac{P \xrightarrow{\alpha} P'}{P\hat{}r \xrightarrow{\alpha} P'} \text{ (Act-sig)} \qquad \frac{P \xrightarrow{b:} P'}{P\hat{}r \xrightarrow{b:} P'\hat{}r} \text{ (Dis-sig)}$$

message to be discarded by a process that cannot receive it. A process offering a choice can only perform a discard-action if neither choice-option can handle it (Rule (Dis-sum)). Finally, an agent A can discard a broadcast iff the body P of its defining equation can discard it. Note that in all these cases a process does not change state by discarding a broadcast.

There exists a variant of CBS, ABC and ABCdE without discard actions, see [12, 9]. This approach, however, features negative premises in the operational rules. As a consequence, the semantics are not in De Simone format [23]. Making use of discard actions and staying within the De Simone format allows us to use meta-theory about this particular format. For example we know, without producing our own proof, that the operators + and | of ABC and ABCdE are associative and commutative, up to strong bisimilarity [2]. Moreover, strong bisimilarity [18] is a congruence for all operators of ABCdE.

Next to the standard operators of CCS and a broadcast mechanism, ABCdE features also signal emission. Informally, the signalling operator $P\hat{}s$ emits the signal s to be read by another process. Signal emissions cannot block other actions of P. Classical examples are the modelling of read-write processes or traffic lights (see Section 2).

Formally, our process algebra features a set $\mathscr{S}$ of signals. The semantics of signals is presented in Table 3. The first rule (Sig) models the emission $\bar{s}$ of signal s to the environment. The environment (processes running in parallel) can read the signal by performing a read action s. This action synchronises with the emission $\bar{s}$, via the rules of Table 1. Reading does not change the state of the emitter. The next four rules describe the interaction between signal emission and other operators, namely choice, signal emission and recursion. In short, these operators do not prevent the emission of a signal, and emitting signals never changes the state of the emitting process. Other operators, such as relabelling and restriction do not need special attention as they are already handled by the corresponding rules in Table 1. This is achieved by carefully selecting the types of the labels: while (Sum-l) features a label α of type Act, the rules for restriction and relabelling use a label $\ell \in \mathscr{L}$. In case a process performs a "proper" action, the signal emission ceases (Rule (Act-sig)), but if the process performs a broadcast discard transition, it does not (Rule (Dis-sig)).

The presented semantics stays within the realm of the De Simone format [23], which brings many advantages. However, there exists an alternative, equivalent semantics, which is based on predicates. Rather than explicitly modelling P emitting s by the transition $P \xrightarrow{\bar{s}} P$, one can introduce the predicate $P^{\curvearrowright s}$. The full semantics can be found in [3]. Some readers might find this notation more intuitive as signal emitting processes do not perform an actual action when a component reads the emitted signal.

5.2 Naming Transitions

The operational semantics of ABCdE presented in Section 5.1 interprets the language as an LTS. In Section 5.3, we aim to extend this LTS into an LTSS by defining a successor relation $\rightsquigarrow$ on the transitions, and thereby also a concurrency relation $\frown\!\!\bullet$. However, the standard interpretation of CCS-like languages, which takes as transitions the triples $P \xrightarrow{\alpha} Q$ that are derivable from the operational rules, does not work for our purpose.

▶ **Example 15.** Let $P = A|B$ with $A \stackrel{def}{=} \tau.A + a.A$ and $B \stackrel{def}{=} \bar{a}.B$. Now the transition $P \xrightarrow{\tau} P$ arises in two ways; either as a transition solely of the left component, or as a synchronisation between both components. The first form of that transition is concurrent with the transition $P \xrightarrow{\bar{a}} P$, but the second is not. In fact, an infinite path that would only perform the τ-transition stemming from the left component would not be just, whereas a path that schedules both τ-transitions infinitely often is. This shows that we want to distinguish these two τ-transitions, and hence not see a transition as a triple $P \xrightarrow{\alpha} Q$. ⌟

Instead, we take as the set Tr of transitions in our LTSS the *derivations* of the *transition triples* $P \xrightarrow{\alpha} Q$ from our operational rules. This is our reason to start with a definition of an LTS that features transitions as a primitive rather than a derived concept.

▶ **Definition 16** (Derivation)**.** A *derivation* of a transition triple φ is a *well-founded* (without infinite paths that keep going up), ordered, upwardly branching tree with the nodes labelled by transition triples, such that

1. the root is labelled by φ, and
2. if μ is the label of a node and K is the sequence of labels of this node's children then $\frac{K}{\mu}$ is a substitution instance of a rule from Tables 1–3.

Given a derivation, we refer to the subtrees obtained by deleting the root node as its *direct subderivations*. Furthermore, by definition, $\frac{K_\varphi}{\varphi}$ is a substitution instance of a rule, where φ is the label of the derivation's root and K_φ is the sequence of labels of the root's children; the derivation is said to be *obtained* by this rule.

We interpret ABCdE as an LTS $(S, Tr, source, target, \ell)$ by taking as states S the ABCdE expressions $\mathbb{P}$ and as transitions Tr the derivations of transition triples $P \xrightarrow{\alpha} Q$. Given a derivation t of a triple $P \xrightarrow{\alpha} Q$, we define its label $\ell(t) \coloneqq \alpha$, its source $source(t) \coloneqq P$, and its target $target(t) \coloneqq Q$.

▶ **Definition 17** (Name of Derivation)**.** The derivation obtained by application of (Act) is called $\stackrel{\alpha}{\rightarrow}P$. The derivation obtained by application of (Comm) or (Bro) is called $t|u$, where t, u are the names of its direct subderivations.[4] The derivation obtained by application of (Par-L) is called $t|Q$ where t is the direct subderivation's name and Q is the process on the right hand side of $|$ in the derivation's source. In the same way, the derivation obtained by application of (Par-R) is called $P|t$, while (Sum-L), (Sum-R), (Res), (Rel), and (Rec) yield $t{+}Q$, $P{+}t$, $t\backslash L$, $t[f]$ and $A{:}t$, where t is the direct subderivation's name. The remaining four rules of Table 2 yield $b{:}\mathbf{0}$, $b{:}\alpha.P$, $t{+}u$ and $A{:}t$, where t, u are direct subderivations' names. The derivation of $P\hat{}s \xrightarrow{\bar{s}} P\hat{}s$ obtained by (Sig) is called $P^{\rightarrow s}$. Rules (Act-sig), (Dis-sig) and (Sig-sig) yield $t\hat{}r$, and rules (Sig-sum-L), (Sig-sum-R) and (Sig-rec) yield $t{+}Q$, $P{+}t$ and $A{:}t$, where t is the direct subderivation's name.

A derivation's name reflects the syntactic structure of that derivation. The derivations' names not only provide a convenient way to identify derivations but also highlight the compositionality of derivations. For example, given a derivation t of $P \xrightarrow{c} P'$ and a derivation u of $Q \xrightarrow{\bar{c}} Q'$ with $c \in \mathscr{C} \cup \bar{\mathscr{C}} \cup \mathscr{S} \cup \bar{\mathscr{S}}$, $t|u$ will be a derivation of $P|Q \xrightarrow{\tau} P'|Q'$.

Hereafter, we refer to a derivation of a transition triple as a "transition".

[4] The order of a rule's premises should be maintained in the names of derivations obtained by it. Here t should be the derivation corresponding to the first premise and u to the second. As a result, $t \neq u \implies t|u \neq u|t$.

5.3 Successors

In this section we extend the LTS of ABCdE into an LTSS, by defining the successor relation $\rightsquigarrow$. For didactic reasons, we do so first for CCS, and then extend our work to ABCdE.

Note that $\chi \rightsquigarrow_\zeta \chi'$ can hold only when $\mathit{source}(\chi) = \mathit{source}(\zeta)$, i.e., transitions χ and ζ are both enabled in the state $O := \mathit{source}(\chi) = \mathit{source}(\zeta)$. It can thus be defined by structural induction on O. The meaning of $\chi \rightsquigarrow_\zeta \chi'$ is (a) that χ is unaffected by ζ – denoted $\chi \smile\!\!\!\bullet\, \zeta$ – and (b) that when doing ζ instead of χ, afterwards a variant χ' of χ is still enabled. Restricted to CCS, the relation $\smile\!\!\!\bullet$ is moreover symmetric, and we can write $\chi \smile \zeta$.

In the special case that $O = \mathbf{0}$ or $O = \alpha.Q$, there are no two concurrent transitions enabled in O, so this yields no triples $\chi \rightsquigarrow_\zeta \chi'$. When $O = P + Q$, any two concurrent transitions $\chi \smile \zeta$ enabled in O must either stem both from P or both from Q. In the former case, these transitions have the form $\chi = t + Q$ and $\zeta = v + Q$, and we must have $t \smile v$, in the sense that t and v stem from different parallel components within P. So $t \rightsquigarrow_v t'$ for some transition t'. As the execution of ζ discards the summand Q, we also obtain $\chi \rightsquigarrow_\zeta t'$. This motivates Item 1 in Definition 18 below. Item 2 follows by symmetry.

Let $O = P|Q$. One possibility for $\chi \rightsquigarrow_\zeta \chi'$ is that χ comes from the left component and ζ from the right. So χ has the form $t|Q$ and $\zeta = P|w$. In that case χ and ζ must be concurrent: we always have $\chi \smile \zeta$. When doing w on the right, the left component does not change, and afterwards t is still possible. Hence $\chi \rightsquigarrow_\zeta t|\mathit{target}(w)$. This explains Item 3 in Definition 18.

Another possibility is that χ and ζ both stem from the left component. In that case $\chi = t|Q$ and $\zeta = v|Q$, and it must be that $t \smile u$ within the left component. Thus $t \rightsquigarrow_v t'$ for some transition t', and we obtain $\chi \rightsquigarrow_\zeta t'|Q$. This motivates the first part of Item 4.

It can also happen that χ stems form the left component, whereas ζ is a synchronisation, involving both components. Thus $\chi = t|Q$ and $\zeta = v|w$. For $\chi \smile \zeta$ to hold, it must be that $t \smile v$, whereas the w-part of ζ cannot interfere with t. This yields the second part of Item 4.

The last part of Item 4 is explained in a similar vain from the possibility that ζ stems from the left while χ is a synchronisation of both components. Item 5 follows by symmetry.

In case both χ and ζ are synchronisations involving both components, i.e., $\chi = t|u$ and $\zeta = v|w$, it must be that $t \smile v$ and $u \smile w$. Now the resulting variant χ' of χ after ζ is simply $t'|v'$, where $t \rightsquigarrow_v t'$ and $u \rightsquigarrow_v u'$. This underpins Item 6.

If O has the form $P[f]$, χ and ζ must have the form $t[f]$ and $v[f]$, respectively. Whether t and v are concurrent is not influenced by the renaming operator. So $t \smile v$. The variant of t that remains after doing v is also not affected by the renaming, so if $t \rightsquigarrow_v t'$ then $\chi \rightsquigarrow_\zeta t'[f]$. The case that O has the form $P\backslash L$ is equally trivial. This yields the first two parts of Item 7.

In case $O = A$ with $A \stackrel{def}{=} P$, then χ and ζ must have the forms $A{:}t$ and $A{:}v$, respectively, where t and v are enabled in P. Now $\chi \smile \zeta$ only if $t \smile v$, so $t \rightsquigarrow_v t'$ for some transition t'. As the recursion around P disappears upon executing ζ, we obtain $\chi \rightsquigarrow_\zeta t'$. This yields the last part of Item 7. Together, this motivates the following definition.

▶ **Definition 18** (Successor Relation for CCS). The relation $\rightsquigarrow \subseteq \mathit{Tr} \times \mathit{Tr} \times \mathit{Tr}$ is the smallest relation satisfying

1. $t \rightsquigarrow_v t'$ implies $t + Q \rightsquigarrow_{v+Q} t'$,
2. $u \rightsquigarrow_w u'$ implies $P + u \rightsquigarrow_{P+w} u'$,
3. $t|Q \rightsquigarrow_{P|w} (t|\mathit{target}(w))$ and $P|u \rightsquigarrow_{v|Q} (\mathit{target}(v)|u)$,
4. $t \rightsquigarrow_v t'$ implies $t|Q \rightsquigarrow_{v|Q} t'|Q$, $t|Q \rightsquigarrow_{v|w} (t'|\mathit{target}(w))$, and $t|u \rightsquigarrow_{v|Q} t'|u$,
5. $u \rightsquigarrow_w u'$ implies $P|u \rightsquigarrow_{P|w} P|u'$, $P|u \rightsquigarrow_{v|w} (\mathit{target}(v)|u')$, and $t|u \rightsquigarrow_{P|w} t|u'$,
6. $t \rightsquigarrow_v t' \wedge u \rightsquigarrow_w u'$ implies $t|u \rightsquigarrow_{v|w} t'|u'$,
7. $t \rightsquigarrow_v t'$ implies $t\backslash L \rightsquigarrow_{v\backslash L} t'\backslash L$, $t[f] \rightsquigarrow_{v[f]} t'[f]$ and $A{:}t \rightsquigarrow_{A:v} t'$.

for all $t, t', u, u', v, w \in Tr$, $P, Q \in \mathbb{P}$ and L, f, A with $source(t) = source(v) = P$, $source(u) = source(w) = Q$, $source(t') = target(v)$, $source(u') = target(w)$, $L \subseteq \mathscr{C}$, f a relabelling and $A \in \mathscr{A}$ – provided that the composed transitions exist.

By projecting the ternary relation $\rightsquigarrow$ on its first two components, we obtain a characterisation of the concurrency relation $\smile$ between CCS transitions:

▶ **Observation 19** (Concurrency Relation for CCS). The relation $\smile \subseteq Tr \times Tr$ is the smallest relation satisfying

1. $t \smile v$ implies $t + Q \smile v + Q$,
2. $u \smile w$ implies $P + u \smile P + w$,
3. $t|Q \smile P|w$ and $P|u \smile v|Q$,
4. $t \smile v$ implies $t|Q \smile v|Q$, $t|Q \smile v|w$, and $t|u \smile v|Q$,
5. $u \smile w$ implies $P|u \smile P|w$, $P|u \smile v|w$, and $t|u \smile P|w$,
6. $t \smile v \wedge u \smile w$ implies $t|u \smile v|w$,
7. $t \smile v$ implies $t\backslash L \smile v\backslash L$, $t[f] \smile v[f]$ and $A{:}t \smile A{:}v$,

for all $t, u, v, w \in Tr$, $P, Q \in \mathbb{P}$ and L, f, A with $source(t) = source(v) = P$, $source(u) = source(w) = Q$, $L \subseteq \mathscr{C}$, f a relabelling and $A \in \mathscr{A}$ – provided that the composed transitions exist.

The same concurrency relation appeared earlier in [12]. Definition 18 and Observation 19 implicitly provide SOS rules for $\rightsquigarrow$ and $\smile$, such as $\frac{t \smile v'}{t+Q \smile v+Q}$. It is part of future work to investigate a rule format for ep-bisimilarity.

Definition 20 below generalises Definition 18 to all of ABCdE. In the special case that ζ is a broadcast discard or signal emission, i.e., $\ell(\zeta) \in \mathscr{B}{:} \cup \bar{\mathscr{S}}$, the transition ζ does not change state – we have $source(\zeta) = target(\zeta) = O$ – and is supposed not to interfere with any other transition χ enabled in O. Hence $\chi \smile\!\!\bullet\, \zeta$ and $\chi \rightsquigarrow_\zeta \chi$. This is Item 1 from Definition 20.

Consequently, in Item 11, which corresponds to Item 7 from Definition 18, we can now safely restrict attention to the case $\ell(\zeta) \in Act$. The last part of that item says that if within the scope of a signalling operator an action v occurs, one escapes from this signalling context, similarly to the cases of choice and recursion. That would not apply if v is a broadcast discard or signal emission, however.

An interesting case is when χ is a broadcast receive or discard transition, i.e., $\ell(\chi) = b?$ or $b{:}$. We postulate that one can never interfere with such an activity, as each process is always able to synchronise with a broadcast action, either by receiving or by discarding it. So we have $\chi \smile\!\!\bullet\, \zeta$ for all ζ with $O = source(\zeta) = source(\chi)$. It could be, however, that in $\chi \rightsquigarrow_\zeta \chi'$, one has $\ell(\chi) = b?$ and $\ell(\chi') = b{:}$, or vice versa. Item 2 says that if $O = \alpha.P$, with ζ the α-transition to P, then χ' can be any transition labelled $b?$ or $b{:}$ that is enabled in P. The second parts of Items 3 and 4 generalise this idea to discard actions enabled in a state of the form $P + Q$. Finally, Items 5 and 6 state that when χ is a broadcast receive stemming from the left side of $O = P + Q$ and ζ an action from the right, or vice versa, then χ' may be any transition labelled $b?$ or $b{:}$ that is enabled in $target(\zeta)$. In all other cases, successors of χ are inherited from successors of their building block, similar to the cases of other transitions.

▶ **Definition 20** (Successor Relation for ABCdE). The relation $\rightsquigarrow \subseteq Tr \times Tr \times Tr$ is the smallest relation satisfying

1. $\ell(\zeta) \in \mathscr{B}{:} \cup \bar{\mathscr{S}}$ and $source(\zeta) = source(\chi)$ implies $\chi \rightsquigarrow_\zeta \chi$,
2. $\ell(t) \in \{b?, b{:}\}$ implies $\xrightarrow{b?}P \rightsquigarrow_{\xrightarrow{b?}P} t$ and $b{:}\alpha.P \rightsquigarrow_{\xrightarrow{\alpha}P} t$,
3. $\ell(v) \notin \bar{\mathscr{S}} \wedge t \rightsquigarrow_v t'$ implies $t + Q \rightsquigarrow_{v+Q} t'$ and $t + u \rightsquigarrow_{v+Q} t'$,
4. $\ell(w) \notin \bar{\mathscr{S}} \wedge u \rightsquigarrow_w u'$ implies $P + u \rightsquigarrow_{P+w} u'$ and $t + u \rightsquigarrow_{P+w} u'$,
5. $\ell(w) \notin \bar{\mathscr{S}} \wedge \ell(t) = b? \wedge \ell(u') \in \{b?, b{:}\}$ implies $t + Q \rightsquigarrow_{P+w} u'$,

6. $\ell(v) \notin \bar{\mathscr{S}} \wedge \ell(u) = b? \wedge \ell(t') \in \{b?, b:\}$ implies $P + u \rightsquigarrow_{v+Q} t'$,
7. $t|Q \rightsquigarrow_{P|w} (t|target(w))$ and $P|u \rightsquigarrow_{v|Q} (target(v)|u)$,
8. $t \rightsquigarrow_v t'$ implies $t|Q \rightsquigarrow_{v|Q} t'|Q$, $t|Q \rightsquigarrow_{v|w} (t'|target(w))$, and $t|u \rightsquigarrow_{v|Q} t'|u$,
9. $u \rightsquigarrow_w u'$ implies $P|u \rightsquigarrow_{P|w} P|u'$, $P|u \rightsquigarrow_{v|w} (target(v)|u')$, and $t|u \rightsquigarrow_{P|w} t|u'$,
10. $t \rightsquigarrow_v t' \wedge u \rightsquigarrow_w u'$ implies $t|u \rightsquigarrow_{v|w} t'|u'$,
11. $\ell(v) \in Act \wedge t \rightsquigarrow_v t'$ implies $t\backslash L \rightsquigarrow_{v\backslash L} t'\backslash L$, $t[f] \rightsquigarrow_{v[f]} t'[f]$, $A{:}t \rightsquigarrow_{A:v} t'$ and $t\hat{}r \rightsquigarrow_{v\hat{}r} t'$,

for all $t, t', u, u', v, w \in Tr$, $P, Q \in \mathbb{P}$ and α, L, f, A, b, r with $source(t) = source(v) = P$, $source(u)=source(w)=Q$, $source(t')=target(v)$ and $source(u')=target(w)$, $\alpha \in Act$, $L \subseteq \mathscr{C} \cup \mathscr{S}$, f a relabelling, $A \in \mathscr{A}$, $b \in \mathscr{B}$ and $r \in \mathscr{S}$ – provided that the composed transitions exist.

Although we have chosen to inductively define the $\rightsquigarrow$ relations, in [15, Appendix B] we follow a different approach in which Definition 20 appears as a theorem rather than a definition. Following [9], we understand each transition as the synchronisation of a number of elementary particles called *synchrons*. Then relations on synchrons are proposed in terms of which the $\rightsquigarrow$ relation is defined. That this leads to the same result indicates that the above definition is more principled than arbitrary.

5.4 Congruence and Other Basic Properties of Ep-bisimilarity

As mentioned before, the operators $+$ and $|$ are associative and commutative up to strong bisimilarity. We can strengthen this result.

▶ **Theorem 21.** *The operators $+$ and $|$ are associative and commutative up to $\leftrightarrows_{ep}$.*

Proof. Remember that $\mathbb{P}$ denotes the set of ABCdE expressions or processes. *Commutativity of $+$, i.e., $P + Q \leftrightarrows_{ep} Q + P$*: The relation

$$\{(I, I, Id_I) \mid I \in \mathbb{P}\} \cup \{(P + Q, Q + P, R_{P,Q}) \mid P, Q \in \mathbb{P}\}$$

is an ep-bisimulation. Here $Id_I := \{(t, t) \mid t \in en(I)\}$ and

$$\begin{aligned} R_{P,Q} := &\{(t + Q, Q + t) \mid t \in en(P) \wedge \ell(t) \notin \mathscr{B}{:}\} \cup \\ &\{(P + u, u + P) \mid u \in en(Q) \wedge \ell(u) \notin \mathscr{B}{:}\} \cup \\ &\{(t + u, u + t) \mid t \in en(P) \wedge u \in en(Q) \wedge \ell(t) = \ell(u) \in \mathscr{B}{:}\}\,. \end{aligned}$$

$R_{P,Q}$ relates transitions, i.e., derivations of transition triples, that are composed of the same sets of direct subderivations, even though their order is reversed.

Associativity of $+$, i.e., $(O + P) + Q \leftrightarrows_{ep} O + (P + Q)$: The relation

$$\{(I, I, Id_I) \mid I \in \mathbb{P}\} \cup \{((O + P) + Q, O + (P + Q), R_{O,P,Q}) \mid O, P, Q \in \mathbb{P}\}$$

is an ep-bisimulation. Here Id_I and $R_{O,P,Q}$ are defined similarly to the previous case.

Commutativity of $|$, i.e., $P|Q \leftrightarrows_{ep} Q|P$: The relation $\{(P|Q, Q|P, R_{P,Q}) \mid P, Q \in \mathbb{P}\}$ is an ep-bisimulation. Here

$$\begin{aligned} R_{P,Q} = &\{(t|Q, Q|t) \mid t \in en(P) \wedge \ell(t) \notin \mathscr{B}! \cup \mathscr{B}? \cup \mathscr{B}{:}\} \cup \\ &\{(P|u, u|P) \mid u \in en(Q) \wedge \ell(u) \notin \mathscr{B}! \cup \mathscr{B}? \cup \mathscr{B}{:}\} \cup \\ &\{(t|u, u|t) \mid t \in en(P) \wedge u \in en(Q) \wedge \ell(t) = \overline{\ell(u)} \in \mathscr{C} \cup \bar{\mathscr{C}} \cup \mathscr{S} \cup \bar{\mathscr{S}}\} \cup \\ &\{(t|u, u|t) \mid t \in en(P) \wedge u \in en(Q) \wedge \\ &\qquad \exists\, b \in \mathscr{B}.\ \{\ell(t), \ell(u)\} \in \{\{b!, b?\}, \{b!, b:\}, \{b?\}, \{b?, b:\}, \{b:\}\}\}\,. \end{aligned}$$

Associativity of $|$, i.e., $(O|P)|Q \leftrightarrows_{ep} O|(P|Q)$: The relation

$$\{(O|P)|Q, O|(P|Q), R_{O,P,Q}) \mid O, P, Q \in \mathbb{P}\}$$

is an ep-bisimulation, where $R_{O,P,Q}$ is defined similarly to the previous case. ◀

Additionally, not only strong bisimilarity should be a congruence for all operators of ABCdE – which follows immediately from the De Simone format – but also our new ep-bisimilarity. This means that if two process terms are ep-bisimilar, then they are also ep-bisimilar in any context.

▶ **Theorem 22.** *Ep-bisimilarity is a congruence for all operators of ABCdE.*

We cannot get it directly from the existing meta-theory on structural operational semantics, as nobody has studied ep-bisimilarity before. As is standard, the proof is a case distinction on the type of the operator. For example, the case for action prefixing requires

$$P \leftrightarrows_{ep} Q \Rightarrow \alpha.P \leftrightarrows_{ep} \alpha.Q \text{ for } \alpha \in Act\,.$$

Such properties can be checked by inspecting the syntactic form of the transition rules, using structural induction. While proofs for some statements, such as the one for action prefixing, are merely a simple exercise, others require more care, including long and tedious case distinctions. A detailed proof of Theorem 22 can be found in Appendix A.

6 Failed Alternatives for Ep-Bisimulation

On an LTSS $(S, Tr, source, target, \ell, \rightsquigarrow)$ an ep-bisimulation has the type $S \times S \times \mathscr{P}(Tr \times Tr)$. This is different from that of other classical bisimulations, which have the type $S \times S$. While developing ep-bisimulation we have also explored dozens of other candidates, many of them being of type $(S \times S) \cup (Tr \times Tr)$. The inclusion of a relation between transitions is necessary to reflect the concept of components or concurrency in one way or the other. One such candidate definition declares a relation $\mathscr{R} \subseteq (S \times S) \cup (Tr \times Tr)$ a valid bisimulation iff the set of triples

$$\{(p, q, R) \mid (p, q) \in \mathscr{R} \cap (S \times S) \wedge R = \mathscr{R} \cap (en(p) \times en(q))\}$$

is an ep-bisimulation. However, neither this candidate nor any of the others leads to a transitive notion of bisimilarity. The problem stems from systems, not hard to model in ABCdE, with multiple paths π_i from states p to p', such that a triple (p, q, R) in an ep-bisimulation $\mathscr{R}$ forces triples (p', q', R'_i) to be in $\mathscr{R}$ for multiple relations $R_i \subseteq en(p') \times en(q')$, depending on the chosen path π_i.

7 Related Work

Our LTSSs generalise the concurrent transition systems of [24]. There $t \rightsquigarrow_v u$ is written as $t{\uparrow}v = u$, and $\uparrow$ is a partial function rather than a relation, in the sense that for given t and v there can be at most one u with $t{\uparrow}v = u$. This condition is not satisfied by broadcast communication, which is one of the reasons we switched to the notation $t \rightsquigarrow_v u$. As an example, consider $b!|a.(b? + b?)$. The $b!$-transition after the a-transition has two variants, namely $\xrightarrow{b!}\mathbf{0}|(\xrightarrow{b?}\mathbf{0}+b?)$ and $\xrightarrow{b!}\mathbf{0}|(b?+\xrightarrow{b?}\mathbf{0})$. Another property of concurrent transition systems that is not maintained in our framework is the symmetry of the induced concurrency relation. Finally, [24] requires that $(v{\uparrow}t){\uparrow}(u{\uparrow}t) = (v{\uparrow}u){\uparrow}(t{\uparrow}u)$, the *cube axiom*, whereas we have so far not found reasons to restrict attention to processes satisfying this axiom. We are, however, open to the possibility that for future applications of LTSSs, some closure properties may be imposed on them.

In [1] a location-based bisimulation is proposed. It also keeps track of the components participating in transitions. The underlying model is quite different from ours, which makes it harder to formally argue that this notion of bisimilarity is incomparable to ours. We do not know yet whether it could be used to reason about justness.

8 Conclusion and Future Work

In related work, it has been argued that fairness assumptions used for verifying liveness properties of distributed systems are too strong or unrealistic [13, 4, 12]. As a consequence, justness, a minimal fairness assumption required for the verification of liveness properties, has been proposed. Unfortunately, all classical semantic equivalences, such as strong bisimilarity, fail to preserve justness.

In this paper, we have introduced labelled transition systems augmented by a successor relation, and, based on that, the concept of enabling preserving bisimilarity, a finer variant of strong bisimilarity. We have proven that this semantic equivalence is a congruence for all classical operators. As it also preserves justness, it is our belief that enabling preserving bisimilarity in combination with justness can and should be used for verifying liveness properties of large-scale distributed systems.

Casually speaking, ep-bisimilarity is strong bisimilarity augmented with the requirement that the relation between enabled transitions is inherited by successor transitions. A straightforward question is whether this feature can be combined with other semantic equivalences, such as weak bisimilarity or trace equivalence.

We have further shown how process algebras can be mapped into LTSSs. Of course, process algebra is only one of many formal frameworks for modelling concurrent systems. For accurately capturing causalities between event occurrences, models like Petri nets [22], event structures [25] or higher dimensional automata [21, 8] are frequently preferable. Part of future work is therefore to develop a formal semantics with respect to LTSSs for these frameworks.

In order to understand the scope of justness in real-world applications, we plan to study systems that depend heavily on liveness. As a starting point we plan to verify locks, such as ticket lock.

References

1 Gérard Boudol, Ilaria Castellani, Matthew Hennessy, and Astrid Kiehn. A theory of processes with localities. *Formal Aspects Comput.*, 6(2):165–200, 1994. doi:10.1007/BF01221098.

2 Sjoerd Cranen, Mohammad Reza Mousavi, and Michel A. Reniers. A rule format for associativity. In F. van Breugel and M. Chechik, editors, *Concurrency Theory (CONCUR '08)*, volume 5201 of LNCS, pages 447–461. Springer, 2008. doi:10.1007/978-3-540-85361-9_35.

3 Victor Dyseryn, Robert J. van Glabbeek, and Peter Höfner. Analysing mutual exclusion using process algebra with signals. In Kirstin Peters and Simone Tini, editors, Proc. Combined 24th International Workshop on *Expressiveness in Concurrency* and 14th Workshop on *Structural Operational Semantics*, volume 255 of *Electronic Proceedings in Theoretical Computer Science*, pages 18–34. Open Publishing Association, 2017. doi:10.4204/EPTCS.255.2.

4 Ansgar Fehnker, Robert J. van Glabbeek, Peter Höfner, Annabelle K. McIver, Marius Portmann, and Wee Lum Tan. A process algebra for wireless mesh networks used for modelling, verifying and analysing AODV. Technical Report 5513, NICTA, 2013. URL: http://arxiv.org/abs/1312.7645.

5 Nissim Francez. *Fairness*. Springer, 1986. doi:10.1007/978-1-4612-4886-6.

6 Robert J. van Glabbeek. Bisimulations for higher dimensional automata. Email message, July 7, 1991, 1991. URL: http://theory.stanford.edu/~rvg/hda.

7 Robert J. van Glabbeek. On specifying timeouts. In L. Aceto and A.D. Gordon, editors, Short Contributions from the Workshop on *Algebraic Process Calculi: The First Twenty Five Years and Beyond,* PA '05, Bertinoro, Italy, 2005, volume 162 of *Electronic Notes in Theoretical Computer Science*, pages 112–113. Elsevier, 2005. doi:10.1016/j.entcs.2005.12.083.

8 Robert J. van Glabbeek. On the expressiveness of higher dimensional automata. *Theoretical Computer Science*, 368(1-2):169–194, 2006. doi:10.1016/j.tcs.2006.06.024.

9 Robert J. van Glabbeek. Justness: A completeness criterion for capturing liveness properties. Technical report, Data61, CSIRO, 2019. Extended abstract in M. Bojańczyk & A. Simpson, editors: Proc. 22st International Conference on *Foundations of Software Science and Computation Structures* (FoSSaCS 2019); held as part of the European Joint Conferences on *Theory and Practice of Software* (ETAPS 2019), Prague, Czech Republic, 2019, LNCS 11425, Springer, pp. 505-522, doi:10.1007/978-3-030-17127-8_29. arXiv:1909.00286.

10 Robert J. van Glabbeek. Reactive temporal logic. In O. Dardha and J. Rot, editors, Proceedings Combined 27th International Workshop on *Expressiveness in Concurrency* and 17th Workshop on *Structural Operational Semantics,* Online, 31 August 2020, volume 322 of *Electronic Proceedings in Theoretical Computer Science*, pages 51–68. Open Publishing Association, 2020. doi:10.4204/EPTCS.322.6.

11 Robert J. van Glabbeek and Peter Höfner. CCS: it's not fair! *Acta Informatica*, 52(2-3):175–205, 2015. doi:10.1007/s00236-015-0221-6.

12 Robert J. van Glabbeek and Peter Höfner. Progress, fairness and justness in process algebra. Technical Report 8501, NICTA, 2015. arXiv:1501.03268.

13 Robert J. van Glabbeek and Peter Höfner. Progress, justness, and fairness. *ACM Computing Surveys*, 52(4), 2019. doi:10.1145/3329125.

14 Robert J. van Glabbeek, Peter Höfner, Marius Portmann, and Wee Lum Tan. Modelling and verifying the AODV routing protocol. *Distributed Computing*, 29(4):279–315, 2016. doi:10.1007/s00446-015-0262-7.

15 Robert J. van Glabbeek, Peter Höfner, and Weiyou Wang. Enabling preserving bisimulation equivalence, full version of the present paper, 2021. arXiv:2108.00142.

16 Eric Goubault and Thomas P. Jensen. Homology of higher dimensional automata. In W.R. Cleaveland, editor, *Proceedings CONCUR 92,* Stony Brook, NY, USA, volume 630 of LNCS, pages 254–268. Springer, 1992. doi:10.1007/BFb0084796.

17 Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, 1977. doi:10.1109/TSE.1977.229904.

18 Robin Milner. Operational and algebraic semantics of concurrent processes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 19, pages 1201–1242. Elsevier Science Publishers B.V. (North-Holland), 1990. Alternatively see *Communication and Concurrency*, Prentice-Hall, Englewood Cliffs, 1989, of which an earlier version appeared as *A Calculus of Communicating Systems*, LNCS 92, Springer, 1980, doi:10.1007/3-540-10235-3.

19 Gordon D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60–61:17–139, 2004. Originally appeared in 1981. doi:10.1016/j.jlap.2004.05.001.

20 K. V. S. Prasad. A calculus of broadcasting systems. In Samson Abramsky and T. S. E. Maibaum, editors, TAPSOFT'91: Proceedings of the International Joint Conference on *Theory and Practice of Software Development,* Volume 1: *Colloquium on Trees in Algebra and Programming* (CAAP'91), volume 493 of LNCS, pages 338–358. Springer, 1991. doi:10.1007/3-540-53982-4_19.

21 Vaughan R. Pratt. Modeling concurrency with geometry. In *Principles of Programming Languages (PoPL'91)*, pages 311–322, 1991. doi:10.1145/99583.99625.

22 Wolfgang Reisig. *Understanding Petri Nets — Modeling Techniques, Analysis Methods, Case Studies.* Springer, 2013. doi:10.1007/978-3-642-33278-4.

23 Robert de Simone. Higher-level synchronising devices in Meije-SCCS. *Theoretical Comp. Science*, 37:245–267, 1985. doi:10.1016/0304-3975(85)90093-3.

24 Eugine W. Stark. Concurrent transition systems. *Theoretical Computer Science*, 64:221–269, 1989. doi:10.1016/0304-3975(89)90050-9.

25 Glynn Winskel. Event structures. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II*, volume 255 of LNCS, pages 325–392. Springer, 1987. doi:10.1007/3-540-17906-2_31.

A Congruence Proofs

Ep-bisimilarity is a congruence for all operators of ABCdE iff Propositions 23–28 below hold. We prove them one by one.

▶ **Proposition 23.** If $P \leftrightarrows_{ep} Q$ and $\alpha \in Act$ then $\alpha.P \leftrightarrows_{ep} \alpha.Q$.

Proof. A transition enabled in $\alpha.P$ is either $\xrightarrow{\alpha} P$ or $b{:}\alpha.P$ for some $b \in \mathscr{B}$ with $\alpha \neq b?$.

Let $\mathscr{R} \subseteq \mathbb{P} \times \mathbb{P} \times \mathscr{P}(Tr \times Tr)$ be the smallest relation satisfying
1. if $(P, Q, R) \in \mathscr{R}'$ for some ep-bisimulation $\mathscr{R}'$ then $(P, Q, R) \in \mathscr{R}$,
2. if $(P, Q, R) \in \mathscr{R}$ and $\alpha \in Act$ then $(\alpha.P, \alpha.Q, \alpha.R) \in \mathscr{R}$, where

$$\alpha.R := \{(\xrightarrow{\alpha} P, \xrightarrow{\alpha} Q)\} \cup \{(b{:}\alpha.P, b{:}\alpha.Q) \mid b \in \mathscr{B} \wedge \alpha \neq b?\}.$$

It suffices to show that $\mathscr{R}$ is an ep-bisimulation. I.e., all entries in $\mathscr{R}$ satisfy the requirements of Definition 8. We proceed by structural induction.

Induction base: Suppose $(P, Q, R) \in \mathscr{R}'$ for some ep-bisimulation $\mathscr{R}'$. Since $\mathscr{R}' \subseteq \mathscr{R}$, all requirements of Definition 8 are satisfied.

Induction step: Suppose $(P, Q, R) \in \mathscr{R}$ satisfies all requirements of Definition 8, we prove that (P', Q', R'), where $P' = \alpha.P$, $Q' = \alpha.Q$, and $R' = \alpha.R$, also satisfies those requirements. This follows directly with Definitions 8 and 20. ◀

▶ **Proposition 24.** If $P_{\text{L}} \leftrightarrows_{ep} Q_{\text{L}}$ and $P_{\text{R}} \leftrightarrows_{ep} Q_{\text{R}}$ then $P_{\text{L}} + P_{\text{R}} \leftrightarrows_{ep} Q_{\text{L}} + Q_{\text{R}}$.

Proof. A transition enabled in $P + Q$ is either
- $t + Q$ for some $t \in en(P)$ with $\ell(t) \notin \mathscr{B}{:}$,
- $P + u$ for some $u \in en(Q)$ with $\ell(u) \notin \mathscr{B}{:}$, or
- $t + u$ for some $t \in en(P)$ and $u \in en(Q)$ with $\ell(t) = \ell(u) \in \mathscr{B}{:}$.

Let $\mathscr{R} \subseteq \mathbb{P} \times \mathbb{P} \times \mathscr{P}(Tr \times Tr)$ be the smallest relation satisfying
1. if $(P, Q, R) \in \mathscr{R}'$ for some ep-bisimulation $\mathscr{R}'$ then $(P, Q, R) \in \mathscr{R}$,
2. if $(P_{\text{L}}, Q_{\text{L}}, R_{\text{L}}), (P_{\text{R}}, Q_{\text{R}}, R_{\text{R}}) \in \mathscr{R}$ then $(P_{\text{L}} + P_{\text{R}}, Q_{\text{L}} + Q_{\text{R}}, R_{\text{L}} + R_{\text{R}}) \in \mathscr{R}$, where

$$\begin{aligned} R_{\text{L}} + R_{\text{R}} := &\{(t + P_{\text{R}}, v + Q_{\text{R}}) \mid t\ R_{\text{L}}\ v \wedge \ell(t) \notin \mathscr{B}{:}\} \cup \\ &\{(P_{\text{L}} + u, Q_{\text{L}} + w) \mid u\ R_{\text{R}}\ w \wedge \ell(u) \notin \mathscr{B}{:}\} \cup \\ &\{(t + u, v + w) \mid t\ R_{\text{L}}\ v \wedge u\ R_{\text{R}}\ w \wedge \ell(t) = \ell(u) \in \mathscr{B}{:}\}. \end{aligned}$$

It suffices to show that $\mathscr{R}$ is an ep-bisimulation. I.e., all entries in $\mathscr{R}$ satisfy the requirements of Definition 8. We proceed by structural induction.

Induction base: Suppose $(P, Q, R) \in \mathscr{R}'$ for some ep-bisimulation $\mathscr{R}'$. Since $\mathscr{R}' \subseteq \mathscr{R}$, all requirements of Definition 8 are satisfied.

Induction step: Suppose $(P_{\text{L}}, Q_{\text{L}}, R_{\text{L}}), (P_{\text{R}}, Q_{\text{R}}, R_{\text{R}}) \in \mathscr{R}$ satisfy all requirements of Definition 8, we prove that (P, Q, R), where $P = P_{\text{L}} + P_{\text{R}}$, $Q = Q_{\text{L}} + Q_{\text{R}}$ and $R = R_{\text{L}} + R_{\text{R}}$, also satisfies those requirements.

$R \subseteq en(P) \times en(Q)$ follows from $R_{\text{L}} \subseteq en(P_{\text{L}}) \times en(Q_{\text{L}})$ and $R_{\text{R}} \subseteq en(P_{\text{R}}) \times en(Q_{\text{R}})$.

Requirement 1.a: It suffices to find, for each $\chi \in en(P)$, a $\zeta \in en(Q)$ with $\chi\ R\ \zeta$.
1. Suppose $\chi = t + P_{\text{R}}$ for some $t \in en(P_{\text{L}})$ with $\ell(t) \notin \mathscr{B}{:}$.
   We obtain $v \in en(Q_{\text{L}})$ with $t\ R_{\text{L}}\ v$, and pick $\zeta = v + Q_{\text{R}}$.
2. Suppose $\chi = P_{\text{L}} + u$ for some $u \in en(P_{\text{R}})$ with $\ell(u) \notin \mathscr{B}{:}$.
   We obtain $w \in en(Q_{\text{R}})$ with $u\ R_{\text{R}}\ w$, and pick $\zeta = Q_{\text{L}} + w$.

3. Suppose $\chi = t + u$ for some $t \in en(P_{\mathrm{L}})$ and $u \in en(P_{\mathrm{R}})$ with $\ell(t) = \ell(u) \in \mathscr{B}\!:$.
   We obtain $v \in en(Q_{\mathrm{L}})$ and $w \in en(Q_{\mathrm{R}})$ with $t\ R_{\mathrm{L}}\ v$ and $u\ R_{\mathrm{R}}\ w$, and pick $\zeta = v + w$.

In all cases, $\zeta \in en(Q)$ and $\chi\ R\ \zeta$ hold trivially.

Requirement 1.b: The proof is similar to that of Requirement 1.(a) and is omitted.

Requirement 1.c: This follows directly from the observations that

- $\ell(t) = \ell(v) \implies \ell(t + P_{\mathrm{R}}) = \ell(v + Q_{\mathrm{R}})$,
- $\ell(u) = \ell(w) \implies \ell(P_{\mathrm{L}} + u) = \ell(Q_{\mathrm{L}} + w)$, and
- $\ell(t) = \ell(v) \wedge \ell(u) = \ell(w) \implies \ell(t + u) = \ell(v + w)$;

provided that the composed transitions exist.

Requirement 2: It suffices to find, for arbitrary Υ, Υ' with $\Upsilon\ R\ \Upsilon'$, an R' with $(target(\Upsilon), target(\Upsilon'), R') \in \mathscr{R}$, such that

(a) for arbitrary χ, χ' with $\chi\ R\ \chi'$ and $\chi \rightsquigarrow_{\Upsilon} \zeta$, we can find a ζ' with $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ and $\zeta\ R'\ \zeta'$,

(b) for arbitrary χ, χ' with $\chi\ R\ \chi'$ and $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$, we can find a ζ with $\chi \rightsquigarrow_{\Upsilon} \zeta$ and $\zeta\ R'\ \zeta'$.

Below we focus merely on (a), as (b) will follow by symmetry.

Suppose $\ell(\Upsilon) \in \mathscr{B}\!: \cup \bar{\mathscr{S}}$. Pick $R' = R$. Then $(target(\Upsilon), target(\Upsilon'), R') = (P, Q, R) \in \mathscr{R}$. From $\chi \rightsquigarrow_{\Upsilon} \zeta$ we have $\zeta = \chi$. Pick $\zeta' = \chi'$. Then $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$. $\zeta\ R'\ \zeta'$ is given by $\chi\ R\ \chi'$.

We further split the cases when $\ell(\Upsilon) \in Act$.

1. Suppose $\Upsilon = v + P_{\mathrm{R}}$ and $\Upsilon' = v' + Q_{\mathrm{R}}$ with $v\ R_{\mathrm{L}}\ v'$. We obtain R'_{L} that satisfies Requirement 2 with respect to v and v'. Pick $R' = R'_{\mathrm{L}}$. Then $(target(\Upsilon), target(\Upsilon'), R') = (target(v), target(v'), R'_{\mathrm{L}}) \in \mathscr{R}$.
   a. Suppose $\chi = t + P_{\mathrm{R}}$ and $\chi' = t' + Q_{\mathrm{R}}$ with $t\ R_{\mathrm{L}}\ t'$. From $\chi \rightsquigarrow_{\Upsilon} \zeta$ we have $t \rightsquigarrow_{v} \zeta$. Then we obtain x' with $t' \rightsquigarrow_{v'} x'$ and $\zeta\ R'_{\mathrm{L}}\ x'$. Pick $\zeta' = x'$. Then $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ follows from $t' \rightsquigarrow_{v'} x'$.
   b. Suppose $\chi = P_{\mathrm{L}} + u$ and $\chi' = Q_{\mathrm{L}} + u'$ with $u\ R_{\mathrm{R}}\ u'$. We obtain x' with $\zeta\ R'_{\mathrm{L}}\ x'$ and pick $\zeta' = x'$. From $\chi \rightsquigarrow_{\Upsilon} \zeta$ we have $\ell(\chi) = b?$ and $\ell(\zeta) \in \{b?, b\!:\}$ for some $b \in \mathscr{B}$. Then $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ follows from $\ell(\chi') = b?$ and $\ell(x') \in \{b?, b\!:\}$.
   c. Suppose $\chi = t + u$ and $\chi' = t' + u'$ with $t\ R_{\mathrm{L}}\ t'$ and $u\ R_{\mathrm{R}}\ u'$. From $\chi \rightsquigarrow_{\Upsilon} \zeta$ we have $t \rightsquigarrow_{v} \zeta$. Then we obtain x' with $t' \rightsquigarrow_{v'} x'$ and $\zeta\ R'_{\mathrm{L}}\ x'$. Pick $\zeta' = x'$. Then $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ follows from $t' \rightsquigarrow_{v'} x'$.

   In all cases, $\zeta\ R'\ \zeta'$ is given by $\zeta\ R'_{\mathrm{L}}\ x'$.
2. Suppose $\Upsilon = P_{\mathrm{L}} + w$ and $\Upsilon' = Q_{\mathrm{L}} + w'$ with $w\ R_{\mathrm{R}}\ w'$. The proof is similar to that of the previous case. ◀

▶ **Proposition 25.** If $P_{\mathrm{L}} \leftrightarrows_{ep} Q_{\mathrm{L}}$ and $P_{\mathrm{R}} \leftrightarrows_{ep} Q_{\mathrm{R}}$ then $P_{\mathrm{L}}|P_{\mathrm{R}} \leftrightarrows_{ep} Q_{\mathrm{L}}|Q_{\mathrm{R}}$.

Proof. A transition enabled in $P|Q$ is either

- $t|Q$ for some $t \in en(P)$ with $\ell(t) \notin \mathscr{B}\!! \cup \mathscr{B}\!? \cup \mathscr{B}\!:$,
- $P|u$ for some $u \in en(Q)$ with $\ell(u) \notin \mathscr{B}\!! \cup \mathscr{B}\!? \cup \mathscr{B}\!:$,
- $t|u$ for some $t \in en(P)$ and $u \in en(Q)$ with $\ell(t) = \overline{\ell(u)} \in \mathscr{C} \cup \bar{\mathscr{C}} \cup \mathscr{S} \cup \bar{\mathscr{S}}$, or
- $t|u$ for some $t \in en(P)$ and $u \in en(Q)$ with $\{\ell(t), \ell(u)\} \in \{\{b!, b?\}, \{b!, b\!:\}, \{b?\}, \{b?, b\!:\}, \{b\!:\}\}$ for some $b \in \mathscr{B}$.

Let $\mathscr{R} \subseteq \mathbb{P} \times \mathbb{P} \times \mathscr{P}(Tr \times Tr)$ be the smallest relation satisfying

1. if $(P, Q, R) \in \mathscr{R}'$ for some ep-bisimulation $\mathscr{R}'$ then $(P, Q, R) \in \mathscr{R}$,

2. if $(P_{\mathrm{L}}, Q_{\mathrm{L}}, R_{\mathrm{L}}), (P_{\mathrm{R}}, Q_{\mathrm{R}}, R_{\mathrm{R}}) \in \mathscr{R}$ then $(P_{\mathrm{L}}|P_{\mathrm{R}}, Q_{\mathrm{L}}|Q_{\mathrm{R}}, R_{\mathrm{L}}|R_{\mathrm{R}}) \in \mathscr{R}$, where

$$\begin{aligned} R_{\mathrm{L}}|R_{\mathrm{R}} \coloneqq & \{(t|P_{\mathrm{R}}, v|Q_{\mathrm{R}}) \mid t\ R_{\mathrm{L}}\ v \wedge \ell(t) \notin \mathscr{B}! \cup \mathscr{B}? \cup \mathscr{B}:\} \cup \\ & \{(P_{\mathrm{L}}|u, Q_{\mathrm{L}}|w) \mid u\ R_{\mathrm{R}}\ w \wedge \ell(u) \notin \mathscr{B}! \cup \mathscr{B}? \cup \mathscr{B}:\} \cup \\ & \{(t|u, v|w) \mid t\ R_{\mathrm{L}}\ v \wedge u\ R_{\mathrm{R}}\ w \wedge \ell(t) = \overline{\ell(u)} \in \mathscr{C} \cup \bar{\mathscr{C}} \cup \mathscr{S} \cup \bar{\mathscr{S}}\} \cup \\ & \{(t|u, v|w) \mid t\ R_{\mathrm{L}}\ v \wedge u\ R_{\mathrm{R}}\ w \wedge \\ & \qquad \exists b \in \mathscr{B}.\ \{\ell(t), \ell(u)\} \in \{\{b!, b?\}, \{b!, b:\}, \{b?\}, \{b?, b:\}, \{b:\}\}\}\,. \end{aligned}$$

It suffices to show that $\mathscr{R}$ is an ep-bisimulation. I.e., all entries in $\mathscr{R}$ satisfy the requirements of Definition 8. We proceed by structural induction.

Induction base: Suppose $(P, Q, R) \in \mathscr{R}'$ for some ep-bisimulation $\mathscr{R}'$. Since $\mathscr{R}' \subseteq \mathscr{R}$, all requirements of Definition 8 are satisfied.

Induction step: Suppose $(P_{\mathrm{L}}, Q_{\mathrm{L}}, R_{\mathrm{L}}), (P_{\mathrm{R}}, Q_{\mathrm{R}}, R_{\mathrm{R}}) \in \mathscr{R}$ satisfy all requirements of Definition 8, we prove that (P, Q, R), where $P = P_{\mathrm{L}}|P_{\mathrm{R}}$, $Q = Q_{\mathrm{L}}|Q_{\mathrm{R}}$ and $R = R_{\mathrm{L}}|R_{\mathrm{R}}$, also satisfies those requirements.

$R \subseteq \mathit{en}(P) \times \mathit{en}(Q)$ follows from $R_{\mathrm{L}} \subseteq \mathit{en}(P_{\mathrm{L}}) \times \mathit{en}(Q_{\mathrm{L}})$ and $R_{\mathrm{R}} \subseteq \mathit{en}(P_{\mathrm{R}}) \times \mathit{en}(Q_{\mathrm{R}})$.

Requirement 1.a: It suffices to find, for each $\chi \in \mathit{en}(P)$, a $\zeta \in \mathit{en}(Q)$ with $\chi\ R\ \zeta$.

1. Suppose $\chi = t|P_{\mathrm{R}}$ for some $t \in \mathit{en}(P_{\mathrm{L}})$ with $\ell(t) \notin \mathscr{B}! \cup \mathscr{B}? \cup \mathscr{B}:$.
   We obtain $v \in \mathit{en}(Q_{\mathrm{L}})$ with $t\ R_{\mathrm{L}}\ v$ and pick $\zeta = v|Q_{\mathrm{R}}$.
2. Suppose $\chi = P_{\mathrm{L}}|u$ for some $u \in \mathit{en}(P_{\mathrm{R}})$ with $\ell(u) \notin \mathscr{B}! \cup \mathscr{B}? \cup \mathscr{B}:$.
   We obtain $w \in \mathit{en}(Q_{\mathrm{R}})$ with $u\ R_{\mathrm{R}}\ w$ and pick $\zeta = Q_{\mathrm{L}}|w$.
3. Suppose $\chi = t|u$ for some $t \in \mathit{en}(P_{\mathrm{L}})$, $u \in \mathit{en}(P_{\mathrm{R}})$ with $\ell(t) = \overline{\ell(u)} \in \mathscr{C} \cup \bar{\mathscr{C}} \cup \mathscr{S} \cup \bar{\mathscr{S}}$.
   We obtain $v \in \mathit{en}(Q_{\mathrm{L}})$ and $w \in \mathit{en}(Q_{\mathrm{R}})$ with $t\ R_{\mathrm{L}}\ v$ and $u\ R_{\mathrm{R}}\ w$, and pick $\zeta = v|w$.
4. Suppose $\chi = t|u$ for some $t \in \mathit{en}(P_{\mathrm{L}})$ and $u \in \mathit{en}(P_{\mathrm{R}})$ with $\{\ell(t), \ell(u)\} \in \{\{b!, b?\}, \{b!, b:\}, \{b?\}, \{b?, b:\}, \{b:\}\}$ for some $b \in \mathscr{B}$.
   We obtain $v \in \mathit{en}(Q_{\mathrm{L}})$ and $w \in \mathit{en}(Q_{\mathrm{R}})$ with $t\ R_{\mathrm{L}}\ v$ and $u\ R_{\mathrm{R}}\ w$, and pick $\zeta = v|w$.

In all cases, $\zeta \in \mathit{en}(Q)$ and $\chi\ R\ \zeta$ hold trivially.

Requirement 1.b: The proof is similar to that of Requirement 1.(a) and is omitted.

Requirement 1.c: This follows directly from the observation that

- $\ell(t) = \ell(v) \implies \ell(t|P_{\mathrm{R}}) = \ell(v|Q_{\mathrm{R}})$,
- $\ell(u) = \ell(w) \implies \ell(P_{\mathrm{L}}|u) = \ell(Q_{\mathrm{L}}|w)$, and
- $\ell(t) = \ell(v) \wedge \ell(u) = \ell(w) \implies \ell(t|u) = \ell(v|w)$;

provided that the composed transitions exist.

Requirement 2: It suffices to find, for arbitrary Υ, Υ' with $\Upsilon\ R\ \Upsilon'$, an R' with $(\mathit{target}(\Upsilon), \mathit{target}(\Upsilon'), R') \in \mathscr{R}$, such that

(a) for arbitrary χ, χ' with $\chi\ R\ \chi'$ and $\chi \rightsquigarrow_{\Upsilon} \zeta$, we can find a ζ' with $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ and $\zeta\ R'\ \zeta'$,

(b) for arbitrary χ, χ' with $\chi\ R\ \chi'$ and $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$, we can find a ζ with $\chi \rightsquigarrow_{\Upsilon} \zeta$ and $\zeta\ R'\ \zeta'$.

Below we focus merely on (a), as (b) will follow by symmetry.

1. Suppose $\Upsilon = v|P_{\mathrm{R}}$ and $\Upsilon' = v'|Q_{\mathrm{R}}$ with $v\ R_{\mathrm{L}}\ v'$. We obtain R'_{L} that satisfies Requirement 2 with respect to v and v'. Pick $R' = R'_{\mathrm{L}}|R_{\mathrm{R}}$. Then $(\mathit{target}(\Upsilon), \mathit{target}(\Upsilon'), R') = (\mathit{target}(v)|P_{\mathrm{R}}, \mathit{target}(v')|Q_{\mathrm{R}}, R'_{\mathrm{L}}|R_{\mathrm{R}}) \in \mathscr{R}$.
   a. Suppose $\chi = t|P_{\mathrm{R}}$ and $\chi' = t'|Q_{\mathrm{R}}$ with $t\ R_{\mathrm{L}}\ t'$. From $\chi \rightsquigarrow_{\Upsilon} \zeta$ we have $\zeta = x|P_{\mathrm{R}}$ for some x with $t \rightsquigarrow_{v} x$. Then we obtain x' with $t' \rightsquigarrow_{v'} x'$ and $x\ R'_{\mathrm{L}}\ x'$. Pick $\zeta' = x'|Q_{\mathrm{R}}$. Then $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ follows from $t' \rightsquigarrow_{v'} x'$; $\zeta\ R'\ \zeta'$ is given by $x\ R'_{\mathrm{L}}\ x'$.
   b. Suppose $\chi = P_{\mathrm{L}}|u$ and $\chi' = Q_{\mathrm{L}}|u'$ with $u\ R_{\mathrm{R}}\ u'$. From $\chi \rightsquigarrow_{\Upsilon} \zeta$ we have $\zeta = \mathit{target}(v)|u$. Pick $\zeta' = \mathit{target}(v')|u'$. Then $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ follows directly; $\zeta\ R'\ \zeta'$ is given by $u\ R_{\mathrm{R}}\ u'$.

   c. Suppose $\chi = t|u$ and $\chi' = t'|u'$ with $t\ R_{\text{L}}\ t'$ and $u\ R_{\text{R}}\ u'$. From $\chi \rightsquigarrow_{\Upsilon} \zeta$ we have $\zeta = x|u$ for some x with $t \rightsquigarrow_{v} x$. Then we obtain x' with $t' \rightsquigarrow_{v'} x'$ and $x\ R'_{\text{L}}\ x'$. Pick $\zeta' = x'|u'$. Then $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ follows from $t' \rightsquigarrow_{v'} x'$; $\zeta\ R'\ \zeta'$ is given by $x\ R'_{\text{L}}\ x'$ and $u\ R_{\text{R}}\ u'$.

2. Suppose $\Upsilon = P_{\text{L}}|w$ and $\Upsilon' = Q_{\text{L}}|w'$ with $w\ R_{\text{R}}\ w'$. The proof is similar to that of the previous case.
3. Suppose $\Upsilon = v|w$ and $\Upsilon' = v'|w'$ with $v\ R_{\text{L}}\ v'$ and $w\ R_{\text{R}}\ w'$. We obtain R'_{L} that satisfies Requirement 2 with respect to v and v', and R'_{R} that satisfies Requirement 2 with respect to w and w'. Pick $R' = R'_{\text{L}}|R'_{\text{R}}$. Then $(\mathit{target}(\Upsilon), \mathit{target}(\Upsilon'), R') = (\mathit{target}(v)|\mathit{target}(w), \mathit{target}(v')|\mathit{target}(w'), R'_{\text{L}}|R'_{\text{R}}) \in \mathscr{R}$.
   a. Suppose $\chi = t|P_{\text{R}}$ and $\chi' = t'|Q_{\text{R}}$ with $t\ R_{\text{L}}\ t'$. From $\chi \rightsquigarrow_{\Upsilon} \zeta$ we have $\zeta = x|\mathit{target}(w)$ for some x with $t \rightsquigarrow_{v} x$. Then we obtain x' with $t' \rightsquigarrow_{v'} x'$ and $x\ R'_{\text{L}}\ x'$. Pick $\zeta' = x'|\mathit{target}(w')$. Then $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ follows from $t' \rightsquigarrow_{v'} x'$; $\zeta\ R'\ \zeta'$ is given by $x\ R'_{\text{L}}\ x'$.
   b. Suppose $\chi = P_{\text{L}}|u$ and $\chi' = Q_{\text{L}}|u'$ with $u\ R_{\text{R}}\ u'$. From $\chi \rightsquigarrow_{\Upsilon} \zeta$ we have $\zeta = \mathit{target}(v)|y$ for some y with $u \rightsquigarrow_{w} y$. Then we obtain y' with $u' \rightsquigarrow_{w'} y'$ and $y\ R'_{\text{R}}\ y'$. Pick $\zeta' = \mathit{target}(v')|y'$. Then $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ follows from $u' \rightsquigarrow_{w'} y'$; $\zeta\ R'\ \zeta'$ is given by $y\ R'_{\text{R}}\ y'$.
   c. Suppose $\chi = t|u$ and $\chi' = t'|u'$ with $t\ R_{\text{L}}\ t'$ and $u\ R_{\text{R}}\ u'$. From $\chi \rightsquigarrow_{\Upsilon} \zeta$ we have $\zeta = x|y$ for some x, y with $t \rightsquigarrow_{v} x$ and $u \rightsquigarrow_{w} y$. Then we obtain x', y' with $t' \rightsquigarrow_{v'} x'$, $u' \rightsquigarrow_{w'} y'$, $x\ R'_{\text{L}}\ x'$, and $y\ R'_{\text{R}}\ y'$. Pick $\zeta' = x'|y'$. Then $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ follows from $t' \rightsquigarrow_{v'} x'$ and $u' \rightsquigarrow_{w'} y'$; $\zeta\ R'\ \zeta'$ is given by $x\ R'_{\text{L}}\ x'$ and $y\ R'_{\text{R}}\ y'$. ◀

▶ **Proposition 26.** If $P \leftrightarrows_{ep} Q$ and $L \subseteq \mathscr{C} \cup \mathscr{S}$ then $P\backslash L \leftrightarrows_{ep} Q\backslash L$.

Proof. A transition enabled in $P\backslash L$ is $t\backslash L$ for some $t \in \mathit{en}(P)$ with $\ell(t) \notin L \cup \overline{L}$.

Let $\mathscr{R} \subseteq \mathbb{P} \times \mathbb{P} \times \mathscr{P}(\mathit{Tr} \times \mathit{Tr})$ be the smallest relation satisfying
1. if $(P, Q, R) \in \mathscr{R}'$ for some ep-bisimulation $\mathscr{R}'$ then $(P, Q, R) \in \mathscr{R}$,
2. if $(P, Q, R) \in \mathscr{R}$ and $L \subseteq \mathscr{C} \cup \mathscr{S}$ then $(P\backslash L, Q\backslash L, R\backslash L) \in \mathscr{R}$, where

$$R\backslash L \coloneqq \{(t\backslash L, v\backslash L) \mid t\ R\ v \wedge \ell(t) \notin L \cup \overline{L}\}\,.$$

It suffices to show that $\mathscr{R}$ is an ep-bisimulation. I.e., all entries in $\mathscr{R}$ satisfy the requirements of Definition 8. We proceed by structural induction.

Induction base: Suppose $(P, Q, R) \in \mathscr{R}'$ for some ep-bisimulation $\mathscr{R}'$. Since $\mathscr{R}' \subseteq \mathscr{R}$, all requirements of Definition 8 are satisfied.

Induction step: Suppose $(P, Q, R) \in \mathscr{R}$ satisfies all requirements of Definition 8, we prove that (P', Q', R'), where $P' = P\backslash L$, $Q' = Q\backslash L$, and $R' = R\backslash L$, also satisfies those requirements. This follows directly with Definitions 8 and 20. ◀

▶ **Proposition 27.** If $P \leftrightarrows_{ep} Q$ and f is a relabelling then $P[f] \leftrightarrows_{ep} Q[f]$.

Proof. An easy structural induction on the structure of P; similar to the proof for restriction (Proposition 26). ◀

▶ **Proposition 28.** If $P \leftrightarrows_{ep} Q$ and $s \in \mathscr{S}^*$ $P\hat{}s \leftrightarrows_{ep} Q\hat{}s$.

Proof. An easy structural induction on the structure of P; similar to the proof for restriction (Proposition 26). ◀