A Lean-Congruence Format for EP-Bisimilarity

A Lean-Congruence Format for EP-Bisimilarity

Rob van Glabbeek*
School of Informatics
University of Edinburgh, UK
School of Computer Science and Engineering
University of New South Wales
Sydney, Australia
rvg@cs.stanford.edu

Peter Höfner   Weiyou Wang
School of Computing
Australian National University
Canberra, Australia
peter.hoefner@anu.edu.au
weiyou.wang@anu.edu.au

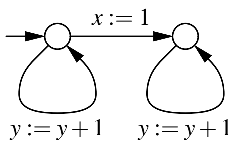
Enabling preserving bisimilarity is a refinement of strong bisimilarity that preserves safety as well as liveness properties. To define it properly, labelled transition systems needed to be upgraded with a successor relation, capturing concurrency between transitions enabled in the same state. We enrich the well-known De Simone format to handle inductive definitions of this successor relation. We then establish that ep-bisimilarity is a congruence for the operators, as well as lean congruence for recursion, for all (enriched) De Simone languages.

1 Introduction

Recently, we introduced a finer alternative to strong bisimilarity, called enabling preserving bisimilarity. The motivation behind this concept was to preserve liveness properties, which are *not* always preserved by classical semantic equivalences, including strong bisimilarity.

Example 1.1 ([13]) Consider the following two programs, and assume that all variables are initialised to 0.

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



```
while(true) do      ||  x := 1;
  y := y+1;         ||
od                  ||
```

The code on the left-hand side presents a non-terminating while-loop offering an internal nondeterministic choice. The conditional **if** x = 0 then x := 1 describes an atomic read-modify-write operation.[1] Since the non-deterministic choice does not guarantee to ever pick the second conditional, this example should not satisfy the liveness property ‘eventually x=1’.

The example on the right-hand side is similar, but here two different components handle the variables x and y separately. The two programs should be considered independent – by default we assume they are executed on different cores. Hence the property ‘eventually x=1’ should hold.

The two programs behave differently with regards to (some) liveness properties. However, it is easy to verify that they are strongly bisimilar, when considering the traditional modelling of such code in terms of transition systems. In fact, their associated transition systems, also displayed above, are identical. Hence, strong bisimilarity does not preserve all liveness properties.

Enabling preserving bisimilarity (ep-bisimilarity) – see next section for a formal definition – distinguishes these examples and preserves liveness. In contrast to classical bisimulations, which are relations of type States × States, this equivalence is based on triples. An ep-bisimulation additionally maintains

*Supported by Royal Society Wolfson Fellowship RSWF\R1\221008
[1] https://en.wikipedia.org/wiki/Read-modify-write

for each pair of related states p and q a relation R between the transitions enabled in p and q, and this relation should be preserved when matching related transitions in the bisimulation game. When formalising this, we need transition systems upgraded with a *successor relation* that matches each transition t enabled in a state p to a transition t' enabled in p', when performing a transition from p to p' that does not affect t. Intuitively, t' describes the same system behaviour as t, but the two transitions could be formally different as they may have different sources. It is this successor relation that distinguishes the transition systems in the example above.

In [13], we showed that ep-bisimilarity is a congruence for all operators of Milner's Calculus of Communication Systems (CCS), enriched with a successor relation. We extended this result to the Algebra of Broadcast Communication with discards and Emissions (ABCdE), an extension of CCS with broadcast communication, discard actions and signal emission. ABCdE subsumes many standard process algebras found in the literature.

In this paper, we introduce a new congruence format for structural operational semantics, which is based on the well-known De Simone Format and respects the successor relation. This format allows us to generalise the results of [13] in two ways: first, we prove that ep-bisimilarity is a congruence for all operators of *any* process algebras that can be formalised in the De Simone format with successors. Applicable languages include CCS and ABCdE. Second, we show that ep-bisimilarity is a lean congruence for recursion [9]. Here, a lean congruence preserves equivalence when replacing closed subexpressions of a process by equivalent alternatives.

2 Enabling Preserving Bisimilarity

To build our abstract theory of De Simone languages and De Simone formats, we briefly recapitulate the definitions of labelled transition systems with successors, and ep-bisimulation. A detailed description can be found in [13].

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 $source(t) = p$. The set of transitions enabled in p is $en(p)$.

Definition 2.1 (LTSS [13]) A *labelled transition system with successors (LTSS)* is a tuple $(S, Tr, source, target, \ell, \leadsto)$ with $(S, Tr, source, target, \ell)$ an LTS and $\leadsto \subseteq Tr \times Tr \times Tr$ the *successor relation* such that if $(t,u,v) \in \leadsto$ (also denoted by $t \leadsto_u v$) then $source(t) = source(u)$ and $source(v) = target(u)$.

Example 2.2 Remember that the 'classical' LTSs of Example 1.1 are identical. 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 $t_1 \leadsto_u t_2$ and $u \leadsto_{t_1} u$. For the other program, the situation is different: as the instructions correspond to a single component (program), all transitions affect each other, i.e. $\leadsto = \emptyset$.

Definition 2.3 (Ep-bisimilarity [13]) Let $(S, Tr, source, target, \ell, \leadsto)$ be an LTSS. 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. $\forall u \in en(q).\ \exists t \in en(p).\ t\, R\, u$, and
   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 $t \leadsto_v t'$ then $\exists u'.\ u \leadsto_w u' \wedge t'\, R'\, u'$, and
   b. if $t\, R\, u$ and $u \leadsto_w u'$ then $\exists t'.\ t \leadsto_v t' \wedge t'\, R'\, u'$.

Table 1: Structural operational semantics of CCS

$\dfrac{}{\alpha.x \xrightarrow{\alpha} x}\ \xrightarrow{\alpha}$	$\dfrac{x \xrightarrow{\alpha} x'}{x+y \xrightarrow{\alpha} x'}\ +_{\mathrm{L}}$	$\dfrac{y \xrightarrow{\alpha} y'}{x+y \xrightarrow{\alpha} y'}\ +_{\mathrm{R}}$
$\dfrac{x \xrightarrow{\eta} x'}{x \mid y \xrightarrow{\eta} x' \mid y}\ \mid_{\mathrm{L}}$	$\dfrac{x \xrightarrow{c} x',\ y \xrightarrow{\bar{c}} y'}{x \mid y \xrightarrow{\tau} x' \mid y'}\ \mid_{\mathrm{C}}$	$\dfrac{y \xrightarrow{\eta} y'}{x \mid y \xrightarrow{\eta} x \mid y'}\ \mid_{\mathrm{R}}$
$\dfrac{x \xrightarrow{\ell} x' \quad (\ell \notin L \cup \overline{L})}{x \backslash L \xrightarrow{\ell} x' \backslash L}\ \backslash L$	$\dfrac{x \xrightarrow{\ell} x'}{x[f] \xrightarrow{f(\ell)} x'[f]}\ [f]$	$\dfrac{\langle S_X \mid S\rangle \xrightarrow{\alpha} y}{\langle X \mid S\rangle \xrightarrow{\alpha} y}\ rec_{Act}$

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

Without Items 2.a and 2.b, the above 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 2.a and 2.b 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 which distinguishes the transition systems for Example 1.1. [13]

Lemma 2.4 [Proposition 10 of [13]] $\leftrightarroweq_{ep}$ is an equivalence relation.

3 An Introductory Example: CCS with Successors

Before starting to introduce the concepts formally, we want to present some motivation in the form of the well-known Calculus of Communicating Systems (CCS) [17]. In this paper we use a proper recursion construct instead of agent identifiers with defining equations. As in [3], we write $\langle X|S\rangle$ for the X-component of a solution of the set of recursive equations S.

CCS is parametrised with set $\mathscr{C}$ of *handshake communication names*. $\bar{\mathscr{C}} := \{\bar{c} \mid c \in \mathscr{C}\}$ is the set of *handshake communication co-names*. $Act_{CCS} := \mathscr{C} \cup \bar{\mathscr{C}} \cup \{\tau\}$ is the set of *actions*, where τ is a special *internal action*. Complementation extends to $\mathscr{C} \cup \bar{\mathscr{C}}$ by $\bar{\bar{c}} := c$.

Below, c ranges over $\mathscr{C} \cup \bar{\mathscr{C}}$ and α, ℓ, η over Act_{CCS}. A *relabelling* is a function $f : \mathscr{C} \to \mathscr{C}$; it extends to Act_{CCS} by $f(\bar{c}) = \overline{f(c)}$, $f(\tau) := \tau$.

The process signature Σ of CCS features binary infix-written operators $+$ and $|$, denoting *choice* and *parallel composition*, a constant $\mathbf{0}$ denoting *inaction*, a unary *action prefixing* operator $\alpha.\_$ for each action $\alpha \in Act_{CCS}$, a unary *restriction* operator $\_\backslash L$ for each set $L \subseteq \mathscr{C}$, and a unary *relabelling* operator $\_[f]$ for each relabelling $f : \mathscr{C} \to \mathscr{C}$.

The semantics of CCS is given by the set $\mathcal{R}$ of *transition rules*, shown in Table 1. Here $\overline{L} := \{\bar{c} \mid c \in L\}$. Each rule has a unique name, displayed in blue.[2] The rules are displayed as templates, following the standard convention of labelling transitions with *label variables* c, α, ℓ, etc. and may be accompanied by side conditions in green, so that each of those templates corresponds to a set of (concrete) transition rules where label variables are "instantiated" to labels in certain ranges and all side conditions are met. The rule names are also schematic and may contain variables. For example, all instances of the transition rule template $+_{\mathrm{L}}$ are named $+_{\mathrm{L}}$, whereas there is one rule name $\xrightarrow{\alpha}$ for each action $\alpha \in Act_{CCS}$.

[2]Our colourings are for readability only.

The transition system specification $(\Sigma, \mathcal{R})$ is in De Simone format [22], a special rule format that guarantees properties of the process algebra (for free), such as strong bisimulation being a congruence for all operators. Following [13], we leave out the infinite sum $\sum_{i\in I} x_i$ of CCS [17], as it is strictly speaking not in De Simone format.

In this paper, we will extend the De Simone format to also guarantee properties for ep-bisimulation. As seen, ep-bisimulation requires that the structural operational semantics is equipped with a successor relation $\leadsto$. The meaning of $\chi \leadsto_\zeta \chi'$ is that transition χ is unaffected by ζ – denoted $\chi \smile\!\!\!\bullet\, \zeta$ – and that when doing ζ instead of χ, afterwards a variant χ' of χ is still enabled. Table 2 shows the *successor rules* for CCS, which allow the relation $\leadsto$ to be derived inductively. It uses the following syntax for transitions χ, which will be formally introduced in Section 6. The expression $t{+_{\rm L}}Q$ refers to the transition that is derived by rule $+_{\rm L}$ of Table 1, with t referring to the transition used in the unique premise of this rule, and Q referring to the process in the inactive argument of the $+$-operator. The syntax for the other transitions is analogous. A small deviation of this scheme occurs for recursion: $rec_{Act}(X,S,t)$ refers to the transition derived by rule rec_{Act} out of the premise t, when deriving a transition of a recursive call $\langle X|S\rangle$.

In Table 2 each rule is named, in orange, after the number of the clause of Definition 20 in [13], were it was introduced.

The primary source of concurrency between transition χ and ζ is when they stem from opposite sides of a parallel composition. This is expressed by Rules 7a and 7b. We require all obtained successor statements $\chi \leadsto_\zeta \chi'$ to satisfy the conditions of Definition 2.1 – this yields $Q' = target(w)$ and $P' = target(v)$; in [13] Q' and P' were written this way.

In all other cases, successors of χ are inherited from successors of their building blocks.

When ζ stems from the left side of a $+$ via rule $+_{\rm L}$ of Table 1, then any transition χ stemming from the right is discarded by ζ, so $\chi \not\smile\!\!\!\bullet\, \zeta$. Thus, if $\chi \smile\!\!\!\bullet\, \zeta$ then these transitions have the form $\chi = t{+_{\rm L}}Q$ and $\zeta = v{+_{\rm L}}Q$, and we must have $t \smile\!\!\!\bullet\, v$. So $t \leadsto_v t'$ for some transition t'. As the execution of ζ discards the summand Q, we also obtain $\chi \leadsto_\zeta t'$. This motivates Rule 3a. Rule 4a follows by symmetry.

In a similar way, Rule 8a covers the case that χ and ζ both stem from the left component of a parallel composition. It can also happen that χ stems form the left component, whereas ζ is a synchronisation, involving both components. Thus $\chi = t|_{\rm L}Q$ and $\zeta = v|_{\rm C}w$. For $\chi \smile\!\!\!\bullet\, \zeta$ to hold, it must be that $t \smile\!\!\!\bullet\, v$, whereas the w-part of ζ cannot interfere with t. This yields the Rule 8b. Rule 8c is explained in a similar vain from the possibility that ζ stems from the left while χ is a synchronisation of both components. Rule 9 follows by symmetry. In case both χ and ζ are synchronisations involving both components, i.e., $\chi = t|_{\rm C}u$ and $\zeta = v|_{\rm C}w$, it must be that $t \smile\!\!\!\bullet\, v$ and $u \smile\!\!\!\bullet\, w$. Now the resulting variant χ' of χ after ζ is simply $t'|u'$, where $t \leadsto_v t'$ and $u \leadsto_w u'$. This underpins Rule 10.

If the common source O of χ and ζ has the form $P[f]$, χ and ζ must have the form $t[f]$ and $v[f]$. Whether t and v are concurrent is not influenced by the renaming. So $t \smile\!\!\!\bullet\, v$. The variant of t that remains after doing v is also not affected by the renaming, so if $t \leadsto_v t'$ then $\chi \leadsto_\zeta t'[f]$. The case that $O = P\backslash L$ is equally trivial. This yields Rules 11a and 11b.

In case $O = \langle X|S\rangle$, χ must have the form $rec_{Act}(X,S,t)$, and ζ has the form $rec_{Act}(X,S,v)$, where t and v are enabled in $\langle S_X|S\rangle$. Now $\chi \smile\!\!\!\bullet\, \zeta$ only if $t \smile\!\!\!\bullet\, v$, so $t \leadsto_v t'$ for some transition t'. The recursive call disappears upon executing ζ, and we obtain $\chi \leadsto_\zeta t'$. This yields Rule 11c.

Example 3.1 The programs from Example 1.1 could be represented in CCS as $P := \langle X|S\rangle$ where $S = \left\{ \begin{array}{l} X = a.X + b.Y \\ Y = a.Y \end{array} \right\}$ and $Q := \langle Z|\{Z = a.Z\}\rangle|b.\mathbf{0}$. Here $a,b \in Act_{CCS}$ are the atomic actions incrementing y and x. The relation matching P with Q and $\langle Y,S\rangle$ with $\langle Z|\{Z = a.Z\}\rangle|\mathbf{0}$ is a strong bisimulation. Yet, P and Q are not ep-bisimilar, as the rules of Table 2 derive $u \leadsto_{t_1} u$ (cf. Example 2.2)

Table 2: Successor rules for CCS

$\dfrac{t \rightsquigarrow_v t'}{t +_{\mathrm{L}} Q \rightsquigarrow_{v +_{\mathrm{L}} Q} t'}\ 3a$		$\dfrac{u \rightsquigarrow_w u'}{P +_{\mathrm{R}} u \rightsquigarrow_{P +_{\mathrm{R}} w} u'}\ 4a$
$\dfrac{}{t\vert_{\mathrm{L}} Q \rightsquigarrow_{P\vert_{\mathrm{R}} w} t\vert_{\mathrm{L}} Q'}\ 7a$	$\dfrac{t \rightsquigarrow_v t' \quad u \rightsquigarrow_w u'}{t\vert_{\mathrm{C}} u \rightsquigarrow_{v\vert_{\mathrm{C}} w} t'\vert_{\mathrm{C}} u'}\ 10$	$\dfrac{}{P\vert_{\mathrm{R}} u \rightsquigarrow_{v\vert_{\mathrm{L}} Q} P'\vert_{\mathrm{R}} u}\ 7b$
$\dfrac{t \rightsquigarrow_v t'}{t\vert_{\mathrm{L}} Q \rightsquigarrow_{v\vert_{\mathrm{L}} Q} t'\vert_{\mathrm{L}} Q}\ 8a$	$\dfrac{t \rightsquigarrow_v t'}{t\vert_{\mathrm{L}} Q \rightsquigarrow_{v\vert_{\mathrm{C}} w} t'\vert_{\mathrm{L}} Q'}\ 8b$	$\dfrac{t \rightsquigarrow_v t'}{t\vert_{\mathrm{C}} u \rightsquigarrow_{v\vert_{\mathrm{L}} Q} t'\vert_{\mathrm{C}} u}\ 8c$
$\dfrac{u \rightsquigarrow_w u'}{P\vert_{\mathrm{R}} u \rightsquigarrow_{P\vert_{\mathrm{R}} w} P\vert_{\mathrm{R}} u'}\ 9a$	$\dfrac{u \rightsquigarrow_w u'}{P\vert_{\mathrm{R}} u \rightsquigarrow_{v\vert_{\mathrm{C}} w} P'\vert_{\mathrm{R}} u'}\ 9b$	$\dfrac{u \rightsquigarrow_w u'}{t\vert_{\mathrm{C}} u \rightsquigarrow_{P\vert_{\mathrm{R}} w} t\vert_{\mathrm{C}} u'}\ 9c$
$\dfrac{t \rightsquigarrow_v t'}{t\backslash L \rightsquigarrow_{v\backslash L} t'\backslash L}\ 11a$	$\dfrac{t \rightsquigarrow_v t'}{t[f] \rightsquigarrow_{v[f]} t'[f]}\ 11b$	$\dfrac{t \rightsquigarrow_v t'}{rec_{Act}(X,S,t) \rightsquigarrow_{rec_{Act}(X,S,v)} t'}\ 11c$

where $u = \langle Z|\{Z = a.Z\}\rangle|_{\mathrm{R}} \xrightarrow{b} \mathbf{0}$ and $t_1 = rec_{Act}(Z, \{Z{=}a.Z\}, \xrightarrow{a} Q)|_{\mathrm{L}} b.\mathbf{0}$. This cannot be matched by P, thus violating condition 2.b. of Definition 2.3.

In this paper we will introduce a new De Simone format for transition systems with successors (TSSS). We will show that $\leftrightarroweq_{ep}$ is a congruence for all operators (as well as a lean congruence for recursion) in any language that fits this format. Since the rules of Table 2 fit this new De Simone format, it follows that $\leftrightarroweq_{ep}$ is a congruence for the operators of CCS.

Informally, the conclusion of a successor rule in this extension of the De Simone format must have the form $\zeta \rightsquigarrow_{\xi} \zeta'$ where ζ, ξ and ζ' are *open transitions*, denoted by *transition expressions* with variables, formally introduced in Section 6. Both ζ and ξ must have a leading operator R and S of the same type, and the same number of arguments. These leading operators must be rule names of the same type. Their arguments are either process variables $P, Q, \ldots$ or transition variables $t, u, \ldots$, as determined by the trigger sets I_{R} and I_{S} of R and S. These are the sets of indices listing the arguments for which rules R and S have a premise. If the i^{th} arguments of R and S are both process variables, they must be the same, but for the rest all these variables are different. For a subset I of $I_{\mathrm{R}} \cap I_{\mathrm{S}}$, the rule has premises $t_i \rightsquigarrow_{u_i} t'_i$ for $i \in I$, where t_i and u_i are the i^{th} arguments of R and S, and t'_i is a fresh variable. Finally, the right-hand side of the conclusion may be an arbitrary univariate transition expression, containing no other variables than:

- the t'_i for $i \in I$,
- a t_i occurring in ζ, with $i \notin I_{\mathrm{S}}$,
- a fresh process variable P'_i that must match the target of the transition u_i for $i \in I_{\mathrm{S}} \backslash I$,
- *or* a fresh transition variable whose source matches the target of u_i for $i \in I_{\mathrm{S}} \backslash I$, and
- any P occurring in both ζ and ξ, *or* any fresh transition variable whose source must be P.

The rules of Table 2 only feature the first three possibilities; the others occur in the successor relation of ABCdE – see Section 8.

4 Structural Operational Semantics

Both the De Simone format and our forthcoming extension are based on the syntactic form of the operational rules. In this section, we recapitulate foundational definitions needed later on. Let $\mathcal{V}_{\mathcal{P}}$ be an infinite set of *process variables*, ranged over by X, Y, x, y, x_i, etc.

Definition 4.1 (Process Expressions [8]) An *operator declaration* is a pair (Op,n) of an *operator symbol* $Op \notin \mathcal{V}_\mathcal{P}$ and an *arity* $n \in \mathbb{N}$. An operator declaration $(c,0)$ is also called a *constant declaration*. A *process signature* is a set of operator declarations. The set $\mathbb{P}^r(\Sigma)$ of *process expressions* over a process signature Σ is defined inductively by:

- $\mathcal{V}_\mathcal{P} \subseteq \mathbb{P}^r(\Sigma)$,
- if $(Op,n) \in \Sigma$ and $p_1,\dots,p_n \in \mathbb{P}^r(\Sigma)$ then $Op(p_1,\dots,p_n) \in \mathbb{P}^r(\Sigma)$, and
- if $V_S \subseteq \mathcal{V}_\mathcal{P}$, $S : V_S \to \mathbb{P}^r(\Sigma)$ and $X \in V_S$, then $\langle X|S\rangle \in \mathbb{P}^r(\Sigma)$.

A process expression $c()$ is abbreviated as c and is also called a *constant*. An expression $\langle X|S\rangle$ as appears in the last clause is called a *recursive call*, and the function S therein is called a *recursive specification*. It is often displayed as $\{X = S_X \mid X \in V_S\}$. Therefore, for a recursive specification S, V_S denotes the domain of S and S_X represents $S(X)$ when $X \in V_S$. Each expression S_Y for $Y \in V_S$ counts as a subexpression of $\langle X|S\rangle$. An occurrence of a process variable y in an expression p is *free* if it does not occur in a subexpression of the form $\langle X|S\rangle$ with $y \in V_S$. For an expression p, $var(p)$ denotes the set of process variables having at least one free occurrence in p. An expression is *closed* if it contains no free occurrences of variables. Let $\mathrm{P}^r(\Sigma)$ be the set of closed process expressions over Σ.

Definition 4.2 (Substitution) A Σ-*substitution* σ is a partial function from $\mathcal{V}_\mathcal{P}$ to $\mathbb{P}^r(\Sigma)$. It is *closed* if it is a total function from $\mathcal{V}_\mathcal{P}$ to $\mathrm{P}^r(\Sigma)$.

If $p \in \mathbb{P}^r(\Sigma)$ and σ a Σ-substitution, then $p[\sigma]$ denotes the expression obtained from p by replacing, for x in the domain of σ, every free occurrence of x in p by $\sigma(x)$, while renaming bound process variables if necessary to prevent name-clashes. In that case $p[\sigma]$ is called a *substitution instance* of p. A substitution instance $p[\sigma]$ where σ is given by $\sigma(x_i) = q_i$ for $i \in I$ is denoted as $p[q_i/x_i]_{i \in I}$, and for S a recursive specification $\langle p|S\rangle$ abbreviates $p[\langle Y|S\rangle/Y]_{Y \in V_S}$.

These notions, including "free" and "closed", extend to syntactic objects containing expressions, with the understanding that such an object is a substitution instance of another one if the same substitution has been applied to each of its constituent expressions.

We assume fixed but arbitrary sets $\mathscr{L}$ and $\mathcal{N}$ of *transition labels* and *rule names*.

Definition 4.3 (Transition System Specification [16]) Let Σ be a process signature. A Σ-*(transition) literal* is an expression $p \xrightarrow{a} q$ with $p,q \in \mathbb{P}^r(\Sigma)$ and $a \in \mathscr{L}$. A *transition rule* over Σ is an expression of the form $\frac{H}{\lambda}$ with H a finite list of Σ-literals (the *premises* of the transition rule) and λ a Σ-literal (the *conclusion*). A *transition system specification (TSS)* is a tuple $(\Sigma, \mathcal{R}, \mathrm{N})$ with $\mathcal{R}$ a set of transition rules over Σ, and $\mathrm{N} : \mathcal{R} \to \mathcal{N}$ a (not necessarily injective) *rule-naming function*, that provides each rule $r \in \mathcal{R}$ with a name $\mathrm{N}(r)$.

Definition 4.4 (Proof) Assume literals, rules, substitution instances and rule-naming. A *proof* of a literal λ from a set $\mathcal{R}$ of rules is a well-founded, upwardly branching, ordered tree where nodes are labelled by pairs (μ, R) of a literal μ and a rule name R, such that

- the root is labelled by a pair (λ, S), and
- if (μ, R) is the label of a node and $(\mu_1, \mathrm{R}_1), \dots, (\mu_n, \mathrm{R}_n)$ is the list of labels of this node's children then $\frac{\mu_1,\dots,\mu_n}{\mu}$ is a substitution instance of a rule in $\mathcal{R}$ with name R.

Definition 4.5 (Associated LTS [12]) The *associated LTS* of a TSS $(\Sigma, \mathcal{R}, \mathrm{N})$ is the LTS $(S, Tr, source, target, \ell)$ with $S := \mathrm{P}^r(\Sigma)$ and Tr the collection of proofs π of closed Σ-literals $p \xrightarrow{a} q$ from $\mathcal{R}$, where $source(\pi) = p$, $\ell(\pi) = a$ and $target(\pi) = q$.

Above we deviate from the standard treatment of structural operational semantics [16, 8] on four counts. Here we employ CCS to motivate those design decisions.

In Definition 4.5, the transitions Tr are taken to be proofs of closed literals $p \xrightarrow{a} q$ rather than such literals themselves. This is because there can be multiple a-transitions from p to q that need to be distinguished when taking the concurrency relation between transitions into account. For example, if $p := \langle X|\{X = a.X + c.X\}\rangle$ and $q := \langle Y|\{Y = a.Y\}\rangle$ then $p|q$ has three outgoing transitions:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a.p \xrightarrow{a} p}\ \overset{a}{\to}}{a.p+c.p \xrightarrow{a} p}\ +_{\mathrm{L}}}{p \xrightarrow{a} p}\ \mathit{rec}_{\mathit{Act}}}{p|q \xrightarrow{a} p|q}\ |_{\mathrm{L}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{c.p \xrightarrow{c} p}\ \overset{c}{\to}}{a.p+c.p \xrightarrow{c} p}\ +_{\mathrm{R}}}{p \xrightarrow{c} p}\ \mathit{rec}_{\mathit{Act}}}{p|q \xrightarrow{c} p|q}\ |_{\mathrm{L}} \qquad \dfrac{\dfrac{\dfrac{}{a.q \xrightarrow{a} q}\ \overset{a}{\to}}{q \xrightarrow{a} q}\ \mathit{rec}_{\mathit{Act}}}{p|q \xrightarrow{a} p|q}\ |_{\mathrm{R}}$$

The rightmost transition is concurrent with the middle one, whereas the leftmost one is not.

A similar example can be used to motivate why in Definition 4.4 the nodes are labelled not only by the inferred literal, but also by the name of the applied rule.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a.p \xrightarrow{a} p}\ \overset{a}{\to}}{a.p+c.p \xrightarrow{a} p}\ +_{\mathrm{L}}}{p \xrightarrow{a} p}\ \mathit{rec}_{\mathit{Act}}}{p|p \xrightarrow{a} p|p}\ |_{\mathrm{L}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{c.p \xrightarrow{c} p}\ \overset{c}{\to}}{a.p+c.p \xrightarrow{c} p}\ +_{\mathrm{R}}}{p \xrightarrow{c} p}\ \mathit{rec}_{\mathit{Act}}}{p|p \xrightarrow{c} p|p}\ |_{\mathrm{L}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{a.p \xrightarrow{a} p}\ \overset{a}{\to}}{a.p+c.p \xrightarrow{a} p}\ +_{\mathrm{L}}}{p \xrightarrow{a} p}\ \mathit{rec}_{\mathit{Act}}}{p|p \xrightarrow{a} p|p}\ |_{\mathrm{R}}$$

The rightmost transition is concurrent with the middle one, but the leftmost one is not. If we were to erase the rule names, the difference between these two transitions would disappear.

In Definition 4.3 we require the premises of rules to be lists rather than sets, and accordingly in Definition 4.4 we require proof trees to be ordered. This is to distinguish transitions/proofs in which a substitution instance of a rule has two identical premises (corresponding to different arguments of the leading operator) with different proofs. This phenomenon does not occur in CCS, but we could have illustrated it with CSP [5] or ABCdE [13].

Finally, suppose that in Definition 4.3 we had chosen the rule-naming function N to be the identity. This is equivalent to not having a rule-naming function at all, instead labelling nodes in proofs with rules rather than names of rules. Then in the transition

$$\dfrac{\dfrac{}{a.\mathbf{0} \xrightarrow{a} \mathbf{0}}\ \overset{a}{\to}}{\langle X|\{X = a.\mathbf{0}\rangle \xrightarrow{a} \mathbf{0}}\ \mathit{rec}_{\mathit{Act}}$$

we should replace the generic name $\mathit{rec}_{\mathit{Act}}$ of a recursion rule with the specific rule employed. This could be the rule $\frac{a.\mathbf{0} \xrightarrow{a} z}{\langle X|\{X = a.\mathbf{0}\}\rangle \xrightarrow{a} z}$, but just as well the rule $\frac{y \xrightarrow{a} z}{\langle X|\{X = y\}\rangle \xrightarrow{a} z}$, when employing a substitution that sends y to $a.\mathbf{0}$. To avoid the resulting unnecessary duplication of transitions, we give both recursion rules the same name.

5 De Simone Languages

The syntax of a *De Simone language* is specified by a process signature, and its semantics is given as a TSS over that process signature of a particular form [22], nowadays known as the *De Simone format*. Here, we extend the De Simone format to support *indicator transitions*, as occur in [11, 10, 13]. These are transitions $p \xrightarrow{\ell} q$ for which it is essential that $p = q$. They are used to convey a property of the state p rather than model an action of p. To accommodate them we need a variant of the recursion rule whose conclusion again is of the form $r \xrightarrow{\ell} r$. This variant will be illustrated in Section 8.

As for $\mathscr{L}$, we fix a set $Act \subseteq \mathscr{L}$ of *actions*.

Definition 5.1 (De Simone Format) A TSS $(\Sigma, \mathcal{R}, \textsc{n})$ is in *De Simone format* if for every recursive call $\langle X|S\rangle$ and every $\alpha \in Act$ and $\ell \in \mathscr{L}\backslash Act$, it has transition rules

$$\frac{\langle S_X|S\rangle \xrightarrow{\alpha} y}{\langle X|S\rangle \xrightarrow{\alpha} y}\ rec_{Act} \qquad \text{and} \qquad \frac{\langle S_X|S\rangle \xrightarrow{\ell} y}{\langle X|S\rangle \xrightarrow{\ell} \langle X|S\rangle}\ rec_{In} \quad \text{for some} \quad y \notin var(\langle S_X|S\rangle),$$

and each of its other transition rules (*De Simone rules*) has the form

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\}}{Op(x_1,\dots,x_n) \xrightarrow{a} q}$$

where $(Op,n) \in \Sigma$, $I \subseteq \{1,\dots,n\}$, $a, a_i \in \mathscr{L}$, x_i (for $1 \leq i \leq n$) and y_i (for $i \in I$) are pairwise distinct process variables, and q is a univariate process expression containing no other free process variables than x_i ($1 \leq i \leq n \wedge i \notin I$) and y_i ($i \in I$), having the properties that

- each subexpression of the form $\langle X|S\rangle$ is closed, and
- if $a \in \mathscr{L}\backslash Act$ then $a_i \in \mathscr{L}\backslash Act$ ($i \in I$) and $q = Op(z_1,\dots,z_n)$, where $z_i := \begin{cases} y_i & \text{if } i \in I \\ x_i & \text{otherwise.} \end{cases}$

Here *univariate* means that each variable has at most one free occurrence in it. The last clause above guarantees that for any indicator transition t, one with $\ell(t) \in \mathscr{L}\backslash Act$, we have $target(t) = source(t)$. For a De Simone rule of the above form, n is the *arity*, (Op,n) is the *type*, a is the *label*, q is the *target*, I is the *trigger set* and the tuple $(\ell_i,\dots,\ell_n)$ with $\ell_i = a_i$ if $i \in I$ and $\ell_i = *$ otherwise, is the *trigger*. Transition rules in the first two clauses are called *recursion rules*.

We also require that if $\textsc{n}(r) = \textsc{n}(r')$ for two different De Simone rules $r, r' \in \mathcal{R}$, then r, r' have the same type, target and trigger set, but different triggers. The names of the recursion rules are as indicated in blue above, and differ from the names of any De Simone rules.

Many process description languages encountered in the literature, including CCS [17] as presented in Section 3, SCCS [18], ACP [3] and MEIJE [2], are De Simone languages.

6 Transition System Specifications with Successors

In Section 4, a *process* is denoted by a closed process expression; an open process expression may contain variables, which stand for as-of-yet unspecified subprocesses. Here we will do the same for transition expressions with variables. However, in this paper a transition is defined as a proof of a literal $p \xrightarrow{a} q$ from the operational rules of a language. Elsewhere, a transition is often defined as a provable literal $p \xrightarrow{a} q$, but here we need to distinguish transitions based on these proofs, as this influences whether two transitions are concurrent.

It turns out to be convenient to introduce an *open proof* of a literal as the semantic interpretation of an open transition expression. It is simply a proof in which certain subproofs are replaced by proof variables.

Definition 6.1 (Open Proof) Given definitions of literals, rules and substitution instances, and a rule-naming function $\textsc{n}$, an *open proof* of a literal λ from a set $\mathcal{R}$ of rules using a set $\mathcal{V}$ of *(proof) variables* is a well-founded, upwardly branching, ordered tree of which the nodes are labelled either by pairs $(\mu, \textsc{r})$ of a literal μ and a rule name $\textsc{r}$, or by pairs (μ, px) of a literal μ and a variable $px \in \mathcal{V}$ such that

- the root is labelled by a pair (λ, χ),
- if (μ, px) is the label of a node then this node has no children,

- if two nodes are labelled by (μ, px) and (μ', px) separately then $\mu = \mu'$, and
- if (μ, R) is the label of a node and $(\mu_1, \chi_1), \ldots, (\mu_n, \chi_n)$ is the list of labels of this node's children then $\frac{\mu_1,\ldots,\mu_n}{\mu}$ is a substitution instance of a rule named R.

Let $\mathcal{V}_\mathcal{T}$ be an infinite set of *transition variables*, disjoint from $\mathcal{V}_\mathcal{P}$. We will use tx, ux, vx, ty, tx_i, etc. to range over $\mathcal{V}_\mathcal{T}$.

Definition 6.2 (Open Transition) Fix a TSS $(\Sigma, \mathcal{R}, \text{N})$. An *open transition* is an open proof of a Σ-literal from $\mathcal{R}$ using $\mathcal{V}_\mathcal{T}$. For an open transition $\mathring{t}$, $var_\mathcal{T}(\mathring{t})$ denotes the set of transition variables occurring in $\mathring{t}$; if its root is labelled by $(p \xrightarrow{a} q, \chi)$ then $src_\circ(\mathring{t}) = p$, $\ell_\circ(\mathring{t}) = a$ and $tar_\circ(\mathring{t}) = q$. The *binding function* $\beta_{\mathring{t}}$ of $\mathring{t}$ from $var_\mathcal{T}(\mathring{t})$ to Σ-literals is defined by $\beta_{\mathring{t}}(tx) = \mu$ if $tx \in var_\mathcal{T}(\mathring{t})$ and (μ, tx) is the label of a node in $\mathring{t}$. Given an open transition, we refer to the subproofs obtained by deleting the root node as its *direct subtransitions*.

All occurrences of transition variables are considered *free*. Let $\mathbb{T}^r(\Sigma, \mathcal{R}, \text{N})$ be the set of open transitions in the TSS $(\Sigma, \mathcal{R}, \text{N})$ and $\text{T}^r(\Sigma, \mathcal{R}, \text{N})$ the set of closed open transitions. We have $\text{T}^r(\Sigma, \mathcal{R}, \text{N}) = Tr$.

Let $en_\circ(p)$ denote $\{\mathring{t} \mid src_\circ(\mathring{t}) = p\}$.

Definition 6.3 (Transition Expression) A *transition declaration* is a tuple (R, n, I) of a *transition constructor* R, an arity $n \in \mathbb{N}$ and a trigger set $I \subseteq \{1, \ldots, n\}$. A *transition signature* is a set of transition declarations. The set $\mathbb{TE}^r(\Sigma_\mathcal{P}, \Sigma_\mathcal{T})$ of *transition expressions* over a process signature $\Sigma_\mathcal{P}$ and a transition signature $\Sigma_\mathcal{T}$ is defined inductively as follows.

- if $tx \in \mathcal{V}_\mathcal{T}$ and μ is a Σ-literal then $(tx :: \mu) \in \mathbb{TE}^r(\Sigma_\mathcal{P}, \Sigma_\mathcal{T})$,
- if $E \in \mathbb{TE}^r(\Sigma_\mathcal{P}, \Sigma_\mathcal{T})$, $S : \mathcal{V}_\mathcal{P} \rightharpoonup \mathbb{P}^r(\Sigma_\mathcal{P})$ and $X \in \text{dom}(S)$ then $rec_{Act}(X, S, E), rec_{In}(X, S, E) \in \mathbb{TE}^r(\Sigma_\mathcal{P}, \Sigma_\mathcal{T})$, and
- if $(\text{R}, n, I) \in \Sigma_\mathcal{T}$, $E_i \in \mathbb{TE}^r(\Sigma_\mathcal{P}, \Sigma_\mathcal{T})$ for each $i \in I$, and $E_i \in \mathbb{P}^r(\Sigma_\mathcal{P})$ for each $i \in \{1, \ldots, n\} \setminus I$, then $\text{R}(E_1, \ldots, E_n) \in \mathbb{TE}^r(\Sigma_\mathcal{P}, \Sigma_\mathcal{T})$.

Given a TSS $(\Sigma, \mathcal{R}, \text{N})$ in De Simone format, each open transition $\mathring{t} \in \mathbb{T}^r(\Sigma, \mathcal{R})$ is named by a unique transition expression in $\mathbb{TE}^r(\Sigma, \Sigma_\mathcal{T})$; here $\Sigma_\mathcal{T} = \{(\text{N}(r), n, I) \mid r \in \mathcal{R}$ is a De Simone rule, n is its arity and I is its trigger set$\}$:

- if the root of $\mathring{t}$ is labelled by (μ, tx) where $tx \in \mathcal{V}_\mathcal{T}$ then $\mathring{t}$ is named $(tx :: \mu)$,
- if the root of $\mathring{t}$ is labelled by $(\langle X|S\rangle \xrightarrow{a} q, \text{R})$ where $a \in Act$ then $\mathring{t}$ is named $rec_{Act}(X, S, E)$ where E is the name of the direct subtransition of $\mathring{t}$,
- if the root of $\mathring{t}$ is labelled by $(\langle X|S\rangle \xrightarrow{\ell} \langle X|S\rangle, \text{R})$ where $\ell \in \mathscr{L} \setminus Act$ then $\mathring{t}$ is named $rec_{In}(X, S, E)$ where E is the name of the direct subtransition of $\mathring{t}$, and
- if the root of $\mathring{t}$ is labelled by $(Op(p_1, \ldots, p_n) \xrightarrow{a} q, \text{R})$ then $\mathring{t}$ is named $\text{R}(E_1, \ldots, E_n)$ where, letting n and I be the arity and the trigger set of the rules named R, E_i for each $i \in I$ is the name of the direct subtransitions of $\mathring{t}$ corresponding to the index i, and $E_i = p_i$ for each $i \in \{1, \ldots, n\} \setminus I$.

We now see that the first requirement for the rule-naming function in Definition 5.1 ensures that every open transition is uniquely identified by its name.

Definition 6.4 (Transition Substitution) Let $(\Sigma, \mathcal{R}, \text{N})$ be a TSS. A $(\Sigma, \mathcal{R})$-*substitution* is a partial function $\sigma_\mathcal{T} : (\mathcal{V}_\mathcal{P} \rightharpoonup \mathbb{P}^r(\Sigma)) \cup (\mathcal{V}_\mathcal{T} \rightharpoonup \mathbb{T}^r(\Sigma, \mathcal{R}))$. It is *closed* if it is a total function $\sigma_\mathcal{T} : (\mathcal{V}_\mathcal{P} \to \text{P}^r(\Sigma)) \cup (\mathcal{V}_\mathcal{T} \to \text{T}^r(\Sigma, \mathcal{R}))$. A $(\Sigma, \mathcal{R})$-substitution $\sigma_\mathcal{T}$ *matches* all process expressions. It matches an open transition $\mathring{t}$ whose binding function is $\beta_{\mathring{t}}$ if for all $(tx, \mu) \in \beta_{\mathring{t}}$, $\sigma_\mathcal{T}(tx)$ being defined and $\mu = (p \xrightarrow{a} q)$ implies $\ell_\circ(\sigma_\mathcal{T}(tx)) = a$ and $src_\circ(\sigma_\mathcal{T}(tx)), tar_\circ(\sigma_\mathcal{T}(tx))$ being the substitution instances of p, q respectively by applying $\sigma_\mathcal{T} \restriction \mathcal{V}_\mathcal{P}$.

If $E \in \mathbb{P}^r(\Sigma) \cup \mathbb{T}^r(\Sigma, \mathcal{R})$ and $\sigma_\mathcal{T}$ is a $(\Sigma, \mathcal{R})$-substitution matching E, then $E[\sigma_\mathcal{T}]$ denotes the expression obtained from E by replacing, for $tx \in \mathcal{V}_\mathcal{T}$ in the domain of $\sigma_\mathcal{T}$, every subexpression of the form

$(tx :: \mu)$ in E by $\sigma_{\mathcal{T}}(tx)$, and for $x \in \mathcal{V}_{\mathcal{P}}$ in the domain of $\sigma_{\mathcal{T}}$, every free occurrence of x in E by $\sigma_{\mathcal{T}}(x)$, while renaming bound process variables if necessary to prevent name-clashes. In that case $E[\sigma_{\mathcal{T}}]$ is called a *substitution instance* of E.

Note that a substitution instance of an open transition can be a transition expression not representing an open transition. For example, applying a $(\Sigma,\mathcal{R})$-substitution $\sigma_{\mathcal{T}}$ given by $\sigma_{\mathcal{T}}(ty) := (tx :: y \xrightarrow{\bar{c}} y')$ to the open transition $(tx :: x \xrightarrow{c} x') \mid (ty :: y \xrightarrow{c} y')$ results in $(tx :: x \xrightarrow{c} x') \mid (tx :: y \xrightarrow{c} y')$ which is not an open transition because the transition variable tx is used for two different Σ-literals. This will not happen if $\sigma_{\mathcal{T}}$ is closed.

Observation 6.5 Given a TSS $(\Sigma,\mathcal{R},\text{N})$, if $\mathring{t} \in en_\circ(p)$ is a open transition and $\sigma_{\mathcal{T}}$ is a closed $(\Sigma,\mathcal{R})$-substitution which matches $\mathring{t}$ then $\mathring{t}[\sigma_{\mathcal{T}}] \in Tr$, $source(\mathring{t}[\sigma_{\mathcal{T}}]) = src_\circ(\mathring{t})[\sigma_{\mathcal{T}}]$, $\ell(\mathring{t}[\sigma_{\mathcal{T}}]) = \ell_\circ(\mathring{t})$ and $target(\mathring{t}[\sigma_{\mathcal{T}}]) = tar_\circ(\mathring{t})[\sigma_{\mathcal{T}}]$.

Definition 6.6 (Transition System Specification with Successors) Let $(\Sigma,\mathcal{R},\text{N})$ be a TSS. A $(\Sigma,\mathcal{R})$-*(successor) literal* is an expression $\mathring{t} \leadsto_{\mathring{u}} \mathring{v}$ with $\mathring{t},\mathring{u},\mathring{v} \in \mathbb{T}^r(\Sigma,\mathcal{R})$, $src_\circ(\mathring{t}) = src_\circ(\mathring{u})$ and $src_\circ(\mathring{v}) = tar_\circ(\mathring{u})$. A *successor rule* over $(\Sigma,\mathcal{R})$ is an expression of the form $\frac{H}{\lambda}$ with H a finite list of $(\Sigma,\mathcal{R})$-literals (the *premises* of the successor rule) and λ a $(\Sigma,\mathcal{R})$-literal (the *conclusion*). A *transition system specification with successors (TSSS)* is a tuple $(\Sigma,\mathcal{R},\text{N},\mathcal{U})$ with $(\Sigma,\mathcal{R},\text{N})$ a TSS and $\mathcal{U}$ a set of successor rules over $(\Sigma,\mathcal{R})$.

Definition 6.7 (Associated LTSS) For a TSSS $(\Sigma,\mathcal{R},\text{N},\mathcal{U})$, the *associated LTSS* is the LTSS $(S, Tr, source, target, \ell, \leadsto)$ with $S := \text{P}^r(\Sigma)$, Tr the collection of proofs π of closed Σ-literals $p \xrightarrow{a} q$ from $\mathcal{R}$, where $source(\pi) = p$, $\ell(\pi) = a$ and $target(\pi) = q$, and

$$\leadsto := \{(t,u,v) \mid \text{a proof of closed } (\Sigma,\mathcal{R})\text{-literal } t \leadsto_u v \text{ from } \mathcal{U} \text{ exists}\}.$$

7 De Simone Languages with Successors

We have enriched standard definitions such as transitions systems and specifications with successors. This allows up to add successors to the De Simone format to define a new congruence format.

Definition 7.1 (De Simone Format) A TSSS $(\Sigma,\mathcal{R},\text{N},\mathcal{U})$ is in *De Simone format* if $(\Sigma,\mathcal{R},\text{N})$ is in De Simone format, for every recursive call $\langle X|S\rangle$ and $xa, ya, za \in \mathscr{L}$ it has a successor rule

$$\frac{(tx :: S_X \xrightarrow{xa} x') \leadsto_{(ty :: S_X \xrightarrow{ya} y')} (tz :: y' \xrightarrow{za} z')}{rec_\chi(X,S,tx :: S_X \xrightarrow{xa} x') \leadsto_{rec_{Act}(X,S,ty :: S_X \xrightarrow{ya} y')} \mathring{u}}$$

where $\mathring{u} = (tz :: y' \xrightarrow{za} z')$ if $ya \in Act$ and $\mathring{u} = rec_\chi(X,S,tx :: S_X \xrightarrow{xa} x')$ otherwise, $rec_\chi = rec_{Act}$ if $xa \in Act$ and $rec_\chi = rec_{In}$ otherwise, x', y', z' are pairwise distinct process variables not occurring in $\langle X|S\rangle$, and tx, ty, tz are pairwise distinct transition variables. Moreover, each of its other successor rules has the form

$$\frac{\{(tx_i :: x_i \xrightarrow{xa_i} x_i') \leadsto_{(ty_i :: x_i \xrightarrow{ya_i} y_i')} (tz_i :: y_i' \xrightarrow{za_i} z_i') \mid i \in I\}}{\text{R}(xe_1,\ldots,xe_n) \leadsto_{\text{S}(ye_1,\ldots,ye_n)} \mathring{v}}$$

such that

- $I \subseteq \{1,\ldots,n\}$,
- x_i, x_i', y_i', z_i' for all relevant i are pairwise distinct process variables,
- tx_i, ty_i, tz_i for all relevant i are pairwise distinct transition variables,

- if $i \in I$ then $xe_i = (tx_i :: x_i \xrightarrow{xa_i} x'_i)$ and $ye_i = (ty_i :: x_i \xrightarrow{ya_i} y'_i)$,
- if $i \notin I$ then xe_i is either x_i or $(tx_i :: x_i \xrightarrow{xa_i} x'_i)$, and ye_i is either x_i or $(ty_i :: x_i \xrightarrow{ya_i} y'_i)$,
- $\mathtt{R}$ and $\mathtt{S}$ are n-ary transition constructors such that the open transitions $\mathtt{R}(xe_1,\dots,xe_n)$, $\mathtt{S}(ye_1,\dots,ye_n)$ and $\mathring{v}$ satisfy
$$src_\circ(\mathtt{R}(xe_1,\dots,xe_n)) = src_\circ(\mathtt{S}(ye_1,\dots,ye_n))$$
and $src_\circ(\mathring{v}) = tar_\circ(\mathtt{S}(ye_1,\dots,ye_n))$,
- $\mathring{v}$ is univariate and contains no other variable expressions than
  - x_i or $(tz_i :: x_i \xrightarrow{za_i} z'_i)$ $(1 \le i \le n \wedge xe_i = ye_i = x_i)$,
  - $(tx_i :: x_i \xrightarrow{xa_i} x'_i)$ $(1 \le i \le n \wedge xe_i \neq x_i \wedge ye_i = x_i)$,
  - y'_i or $(tz_i :: y'_i \xrightarrow{za_i} z'_i)$ $(1 \le i \le n \wedge i \notin I \wedge ye_i \neq x_i)$,
  - $(tz_i :: y'_i \xrightarrow{za_i} z'_i)$ $(i \in I)$, and
- if $\ell_\circ(\mathtt{S}(ye_1,\dots,ye_n)) \in \mathscr{L}\backslash Act$ then for $i \in I$, $ya_i \in \mathscr{L}\backslash Act$; for $i \notin I$, either $xe_i = x_i$ or $ye_i = x_i$; and $\mathring{v} = \mathtt{R}(ze_1,\dots,ze_n)$, where
$$ze_i := \begin{cases} (tz_i :: y'_i \xrightarrow{za_i} z'_i) & \text{if } i \in I \\ xe_i & \text{if } i \notin I \text{ and } ye_i = x_i \\ y'_i & \text{otherwise.} \end{cases}$$

The last clause above is simply to ensure that if $t \rightsquigarrow_u v$ for an indicator transition u, that is, with $\ell(u) \notin Act$, then $v = t$.

The other conditions of Definition 7.1 are illustrated by the Venn diagram of Figure 1. The outer circle depicts the indices $1,\dots,n$ numbering the arguments of the operator Op that is the common type of the De Simone rules named $\mathtt{R}$ and $\mathtt{S}$; $I_{\mathtt{R}}$ and $I_{\mathtt{S}}$ are the trigger sets of $\mathtt{R}$ and $\mathtt{S}$, respectively. In line with Definition 6.3, $xe = x_i$ for $i \in I_{\mathtt{R}}$, and $xe = (tx_i :: x_i \xrightarrow{xa_i} x'_i)$ for $i \notin I_{\mathtt{R}}$. Likewise, $ye = x_i$ for $i \in I_{\mathtt{S}}$, and $ye = (ty_i :: x_i \xrightarrow{xa_i} y'_i)$ for $i \notin I_{\mathtt{S}}$. So the premises of any rule named $\mathtt{S}$ are $\{x_i \xrightarrow{xa_i} y'_i \mid i \in I_{\mathtt{S}}\}$. By Definition 5.1 the target of such a rule is a univariate process expression q with no other variables than $z_1,\dots,z_n$, where $z_i := x_i$ for $i \in I_{\mathtt{S}}$ and $z_i := y'_i$ for $i \notin I_{\mathtt{S}}$. Since $src_\circ(\mathring{v}) = q$, the transition expression $\mathring{v}$ must be univariate, and have no variables other than ze_i for $i = 1,\dots,n$, where ze_i is either the process variable z_i or a transition variable expression $(tz_i :: z_i \xrightarrow{xa_i} z'_i)$.

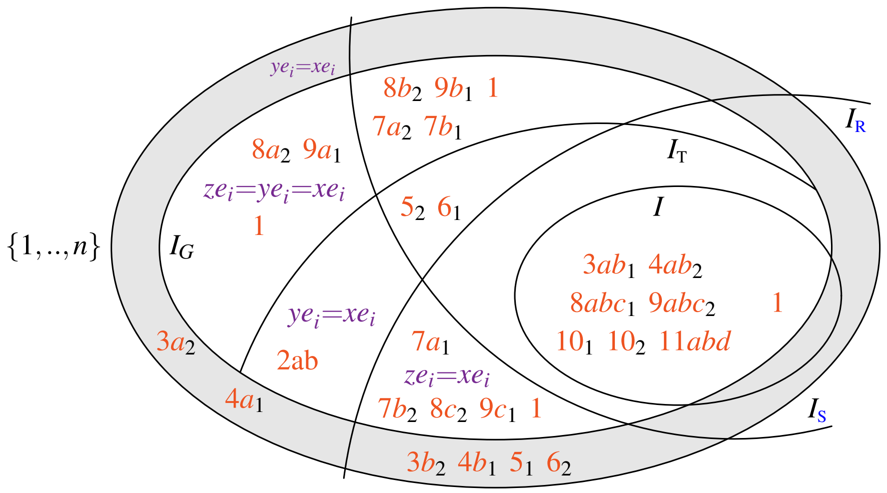


Figure 1: Inclusion between index sets $I, I_{\mathtt{R}}, I_{\mathtt{S}}, I_{\mathtt{T}}, I_G \subseteq \{1,..,n\}$. One has $(I_{\mathtt{R}} \cap I_G)\backslash I_{\mathtt{S}} \subseteq I_{\mathtt{T}}$. The annotations n_i show the location of index i (suppressed for unary operators) of rule n.

I is the set of indices i for which the above successor rule has a premise. Since this premise involves the transition variables tx_i and ty_i, necessarily $I \subseteq I_{\mathrm{R}} \cap I_{\mathrm{S}}$. Let I_G be the set of indices for which ze_i occurs in $\mathring{v}$, and $I_{\mathrm{T}} \subseteq I_G$ be the subset where ze_i is a transition variable. The conditions on $\mathring{v}$ in Definition 7.1 say that $I \cap I_G \subseteq I_{\mathrm{T}}$ and $(I_{\mathrm{R}} \cap I_G) \backslash I_{\mathrm{S}} \subseteq I_{\mathrm{T}}$. For $i \in I \cap I_G$, the transition variable tz_i is inherited from the premises of the rule, and for $i \in (I_{\mathrm{R}} \cap I_G) \backslash I_{\mathrm{S}}$ the transition variable tz_i is inherited from its source.

In order to show that most classes of indices allowed by our format are indeed populated, we indicated the positions of the indices of the rules of CCS and (the forthcoming) ABCdE from Tables 2 and 5.

Any De Simone language, including CCS, SCCS, ACP and MEIJE, can trivially be extended to a language with successors, e.g. by setting $\mathcal{U} = \emptyset$. This would formalise the assumption that the parallel composition operator of these languages is governed by a *scheduler*, scheduling actions from different components in a nondeterministic way. The choice of $\mathcal{U}$ from Table 2 instead formalises the assumption that parallel components act independently, up to synchronisations between them.

We now present the main theorem of this paper, namely that ep-bisimulation is a lean congruence for all languages that can be presented in De Simone format with successors. A lean congruence preserves equivalence when replacing closed subexpressions of a process by equivalent alternatives. Being a lean congruence implies being a congruence for all operators of the language, but also covers the recursion construct.

Theorem 7.2 (Lean Congruence) Ep-bisimulation is a lean congruence for all De Simone languages with successors. Formally, fix a TSSS $(\Sigma, \mathcal{R}, \mathrm{N}, \mathcal{U})$ in De Simone format. If $p \in \mathbb{P}^r(\Sigma)$ and ρ, ν are two closed Σ-substitutions with $\forall x \in \mathcal{V}_{\mathcal{P}}.\ \rho(x) \leftrightarroweq_{ep} \nu(x)$ then $p[\rho] \leftrightarroweq_{ep} p[\nu]$.

The proof can be found in Appendix A of the full version of this paper [15].

In contrast to a lean congruence, a full congruence would also allow replacement within a recursive specification of subexpressions that may contain recursion variables bound outside of these subexpressions. As our proof is already sophisticated, we consider the proof of full congruence to be beyond the scope of the paper. In fact we are only aware of two papers that provide a proof of full congruence via a rule format [21, 9].

We carefully designed our De Simone format with successors and can state the following conjecture.

Conjecture 7.3 Ep-bisimulation is a full congruence for all De Simone languages with successors.

8 A Larger Case Study: The Process Algebra ABCdE

The *Algebra of Broadcast Communication with discards and Emissions* (ABCdE) stems from [13]. It combines CCS [17], its extension with broadcast communication [20, 11, 10], and its extension with signals [4, 6, 7, 10]. Here, we extend CCS as presented in Section 3.

ABCdE is parametrised with sets $\mathscr{C}$ of *handshake communication names* as used in CCS, $\mathscr{B}$ of *broadcast communication names* and $\mathscr{S}$ of *signals*. $\bar{\mathscr{S}} := \{\bar{s} \mid s \in \mathscr{S}\}$ 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 $\sharp \in \{!, ?, :\}$. $Act := \mathscr{C} \uplus \bar{\mathscr{C}} \uplus \{\tau\} \uplus \mathscr{B}! \uplus \mathscr{B}? \uplus \mathscr{S}$ is the set of *actions*, with τ the *internal action*, and $\mathscr{L} := Act \uplus \mathscr{B}{:} \uplus \bar{\mathscr{S}}$ is the set of *transition labels*. Complementation extends to $\mathscr{C} \uplus \bar{\mathscr{C}} \uplus \mathscr{S} \uplus \bar{\mathscr{S}}$ by $\bar{\bar{c}} := c$.

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

Next to the constant and operators of CCS, the process signature Σ of ABCdE features a unary *signalling* operator $\_\hat{}s$ for each signal $s \in \mathscr{S}$.

Table 3: Structural operational semantics of ABCdE

$$\frac{}{\mathbf{0} \xrightarrow{b:} \mathbf{0}}\ b{:}\mathbf{0} \qquad \frac{\alpha \neq b?}{\alpha.x \xrightarrow{b:} \alpha.x}\ b{:}\alpha. \qquad \frac{x \xrightarrow{b:} x',\ y \xrightarrow{b:} y'}{x+y \xrightarrow{b:} x'+y'}\ +_{\mathrm{C}}$$

$$\frac{x \xrightarrow{b\sharp_1} x',\ y \xrightarrow{b\sharp_2} y' \quad (\sharp_1 \circ \sharp_2 = \sharp \neq \_)}{x|y \xrightarrow{b\sharp} x'|y'}\ |_{\mathrm{C}} \qquad \text{with}$$

$\circ$	!	?	:
!	_	!	!
?	!	?	?
:	!	?	:

$$\frac{}{x\,\hat{}\,s \xrightarrow{\bar{s}} x\,\hat{}\,s}\ ({\to}^{s}) \qquad \frac{x \xrightarrow{\bar{s}} x'}{x+y \xrightarrow{\bar{s}} x'+y}\ +^{e}_{L} \qquad \frac{y \xrightarrow{\bar{s}} y'}{x+y \xrightarrow{\bar{s}} x+y'}\ +^{e}_{\mathrm{R}}$$

$$\frac{x \xrightarrow{\alpha} x'}{x\,\hat{}\,s \xrightarrow{\alpha} x'}\ \hat{}\,s_{Act} \qquad \frac{x \xrightarrow{\gamma} x'}{x\,\hat{}\,s \xrightarrow{\gamma} x'\,\hat{}\,s}\ \hat{}\,s_{In} \qquad \frac{\langle S_X|S\rangle \xrightarrow{\gamma} y}{\langle X|S\rangle \xrightarrow{\gamma} \langle X|S\rangle}\ rec_{In}$$

The semantics of ABCdE is given by the transition rule templates displayed in Tables 1 and 3. The latter augments CCS with mechanisms for broadcast communication and signalling. The rule $|_{\mathrm{C}}$ presents the core of broadcast communication [20], where any broadcast-action $b!$ performed by a component in a parallel composition needs to synchronise with either a receive action $b?$ or a discard action $b{:}$ of any other component. In order to ensure associativity of the parallel composition, rule $|_{\mathrm{C}}$ also allows receipt actions of both components ($\sharp_1 = \sharp_2 = ?$), or a receipt and a discard, to be combined into a receipt action.

A transition $p \xrightarrow{b:} q$ is derivable only if $q = p$. It indicates that the process p is unable to receive a broadcast communication $b!$ on channel b. The Rule $b{:}\mathbf{0}$ allows the nil process (inaction) to discard any incoming message; in the same spirit $b{:}\alpha.$ allows a message to be discarded by a process that cannot receive it. A process offering a choice can only perform a discard-action if both choice-options can discard the corresponding broadcast (Rule $+_{\mathrm{C}}$). Finally, by rule rec_{In}, a recursively defined process $\langle X|S\rangle$ can discard a broadcast iff $\langle S_X|S\rangle$ can discard it. The variant rec_{In} of rec_{Act} is introduced to maintain the property that $target(\theta) = source(\theta)$ for any indicator transition θ.

A signalling process $p\,\hat{}\,s$ emits the signal s to be read by another process. A typically example is a traffic light being red. Signal emission is modelled as an indicator transition, which does not change the state of the emitting process. The first rule $({\to}^{s})$ 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 rule $|_{\mathrm{C}}$ of Table 1. Reading a signal does not change the state of the emitter.

Rules $+^{e}_{L}$ and $+^{e}_{\mathrm{R}}$ describe the interaction between signal emission and choice. Relabelling and restriction are handled by rules $\backslash L$ and $[f]$ of Table 1, respectively. These operators do not prevent the emission of a signal, and emitting signals never changes the state of the emitting process. Signal emission $p\,\hat{}\,s$ does not block other transitions of p.

It is trivial to check that the TSS of ABCdE is in De Simone format.

The transition signature of ABCdE (Table 4) is completely determined by the set of transition rule templates in Tables 1 and 3. We have united the rules for handshaking and broadcast communication by assigning the same name $|_{\mathrm{C}}$ to all their instances. When expressing transitions in ABCdE as expressions, we use infix notation for the binary transition constructors, and prefix or postfix notation for unary ones. For example, the transition $b{:}\mathbf{0}()$ is shortened to $b{:}\mathbf{0}$, $\xrightarrow{\alpha}(p)$ to $\xrightarrow{\alpha}p$, $\backslash L(t)$ to $t\backslash L$, and $|_{\mathrm{L}}(t,p)$ to $t|_{\mathrm{L}}p$.

Table 4: Transition signature of ABCdE

Constructor	$\xrightarrow{\alpha}$	$(\rightarrow^{s})$	$b{:}\mathbf{0}$	$b{:}\alpha.$	$+_{\mathrm{L}}$	$+_{\mathrm{R}}$	$+_{\mathrm{C}}$	$+^{\mathrm{e}}_{L}$	$+^{e}_{\mathrm{R}}$	$\vert_{\mathrm{L}}$	$\vert_{\mathrm{C}}$	$\vert_{\mathrm{R}}$	$\backslash L$	$[f]$	$\hat{}s_{Act}$	$\hat{}s_{In}$
Arity	1	1	0	1	2	2	2	2	2	2	2	2	1	1	1	1
Trigger Set	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	{1}	{2}	{1,2}	{1}	{2}	{1}	{1,2}	{2}	{1}	{1}	{1}	{1}

Table 5: Successor rules for ABCdE

$$\frac{\ell(t) \in \{b?, b{:}\}}{\xrightarrow{b?}P \rightsquigarrow_{\xrightarrow{b?}P} t}\ 2a^{b} \qquad \frac{\ell(\zeta) \in \mathscr{B}{:} \cup \bar{\mathscr{S}}}{\chi \rightsquigarrow_{\zeta} \chi}\ 1 \qquad \frac{\ell(t) \in \{b?, b{:}\}}{b{:}\alpha.P \rightsquigarrow_{\xrightarrow{\alpha}P} t}\ 2b^{b}$$

$$\frac{t \rightsquigarrow_{v} t'}{t +^{\mathrm{e}}_{L} Q \rightsquigarrow_{v +_{\mathrm{L}} Q} t'}\ 3a \qquad \frac{t \rightsquigarrow_{v} t'}{t +_{\mathrm{C}} u \rightsquigarrow_{v +_{\mathrm{L}} Q} t'}\ 3b \qquad \frac{u \rightsquigarrow_{w} u'}{P +^{e}_{\mathrm{R}} u \rightsquigarrow_{P +_{\mathrm{R}} w} u'}\ 4a \qquad \frac{u \rightsquigarrow_{w} u'}{t +_{\mathrm{C}} u \rightsquigarrow_{P +_{\mathrm{R}} w} u'}\ 4b$$

$$\frac{\ell(t) = b? \quad \ell(u') \in \{b?, b{:}\}}{\begin{array}{c} t +_{\mathrm{L}} Q \rightsquigarrow_{P +_{\mathrm{R}} w} u' \\ t +^{\mathrm{e}}_{L} Q \rightsquigarrow_{P +_{\mathrm{R}} w} u' \end{array}}\ 5^{b} \qquad \frac{\ell(u) = b? \quad \ell(t') \in \{b?, b{:}\}}{\begin{array}{c} P +_{\mathrm{R}} u \rightsquigarrow_{v +_{\mathrm{L}} Q} t' \\ P +^{e}_{\mathrm{R}} u \rightsquigarrow_{v +_{\mathrm{L}} Q} t' \end{array}}\ 6^{b}$$

$$\frac{t \rightsquigarrow_{v} t'}{rec_{In}(X, S, t) \rightsquigarrow_{rec_{Act}(X,S,v)} t'}\ 11c \qquad \frac{t \rightsquigarrow_{v} t'}{\begin{array}{c} t\hat{}s_{Act} \rightsquigarrow_{v\hat{s}_{Act}} t' \\ t\hat{}s_{In} \rightsquigarrow_{v\hat{s}_{Act}} t' \end{array}}\ 11d$$

Meta	Variable Expression
P	x_1
Q	x_2
P'	y'_1
Q'	y'_2
t	$(tx_1 :: x_1 \xrightarrow{xa_1} x'_1)$
u	$(tx_2 :: x_2 \xrightarrow{xa_2} x'_2)$
v	$(ty_1 :: x_1 \xrightarrow{ya_1} y'_1)$
w	$(ty_2 :: x_2 \xrightarrow{ya_2} y'_2)$
t'	$(tz_1 :: y'_1 \xrightarrow{za_1} z'_1)$
u'	$(tz_2 :: y'_2 \xrightarrow{za_2} z'_2)$

Table 5 extends the successor relation of CCS (Table 2) to ABCdE. P, Q are process variables, t, v transition variables enabled at P, u, w transition variables enabled at Q, P', Q' the targets of v, w, respectively and t', u' transitions enabled at P', Q', respectively. To express those rules in the same way as Definition 7.1, we replace the metavariables P, Q, t, u, etc. with variable expressions as indicated on the right. Here xa_i, ya_i, za_i are label variables that should be instantiated to match the trigger of the rules and side conditions. As ABCdE does not feature operators of arity >2, the index i from Definition 7.1 can be 1 or 2 only.

To save duplication of rules 8b, 8c, 9b, 9c and 10 we have assigned the same name $|_{\mathrm{C}}$ to the rules for handshaking and broadcast communication. The intuition of the rules of Table 5 is explained in detail in [13].

In the naming convention for transitions from [13] the sub- and superscripts of the transition constructors $+$, $|$ and $\hat{}s$, and of the recursion construct, were suppressed. In most cases that yields no ambiguity, as the difference between $|_{\mathrm{L}}$ and $|_{\mathrm{R}}$, for instance, can be detected by checking which of its two arguments are of type transition versus process. Moreover, it avoids the duplication in rules 3a, 4a, 5, 6, 11c and 11d. The ambiguity between $+_{\mathrm{L}}$ and $+^{\mathrm{e}}_{L}$ (or $+_{\mathrm{R}}$ and $+^{e}_{\mathrm{R}}$) was in [13] resolved by adorning rules 3–6 with a side condition $\ell(v) \notin \bar{\mathscr{S}}$ or $\ell(w) \notin \bar{\mathscr{S}}$, and the ambiguity between rec_{Act} and rec_{In} (or $\hat{}s_{Act}$ and $\hat{}s_{In}$) by adorning rules 11c and 11d with a side condition $\ell(v) \in Act$; this is not needed here.

It is easy to check that all rules are in the newly introduced De Simone format, except Rule 1. However, this rule can be converted in to a collection of De Simone rules by substituting $\mathtt{R}(xe_1, \ldots, xe_n)$ for χ and $\mathtt{S}(ye_1, \ldots, ye_n)$ for ζ, adding a premise in the form of $xe_i \rightsquigarrow_{ye_i} (tz_i :: y'_i \xrightarrow{za_i} z'_i))$ if $i \in I_{\mathrm{R}} \cap I_{\mathrm{S}}$, for each pair of rules of the same type named R and S.[3] The various occurrences of 1 in Figure 1 refer to these substitution instances. It follows that $\leftrightarrow_{ep}$ is a congruence for the operators of ABCdE, as well as a lean congruence for recursion.

9 Related Work & Conclusion

In this paper we have added a successor relation to the well-known De Simone format. This has allowed us to prove the general result that enabling preserving bisimilarity – a finer semantic equivalence relation than strong bisimulation – is a lean congruence for all languages with a structural operational semantics within this format. We do not cover full congruence yet, as proofs for general recursions are incredible hard and usually excluded from work justifying semantic equivalences.

There is ample work on congruence formats in the literature. Good overview papers are [1, 19]. For system description languages that do not capture time, probability or other useful extensions to standard process algebras, all congruence formats target strong bisimilarity, or some semantic equivalence or preorder that is strictly coarser than strong bisimilarity. As far as we know, the present paper is the first to define a congruence format for a semantic equivalence that is finer than strong bisimilarity.

Our congruence format also ensures a lean congruence for recursion. So far, the only papers that provide a rule format yielding a congruence property for recursion are [21] and [9], and both of them target strong bisimilarity.

In Sections 3 and 8, we have applied our format to show lean congruence of ep-bisimilarity for the process algebra CCS and ABCdE, respectively. This latter process algebra features broadcast communication and signalling. These two features are representative for issues that may arise elsewhere, and help to ensure that our results are as general as possible. Our congruence format can effortlessly be applied to other calculi like CSP [5] or ACP [3].

In order to evaluate ep-bisimilarity on process algebras like CCS, CSP, ACP or ABCdE, their semantics needs to be given in terms of labelled transition systems extended with a successor relation $\leadsto$. This relation models concurrency between transitions enabled in the same state, and also tells what happens to a transition if a concurrent transition is executed first. Without this extra component, labelled transition systems lack the necessary information to capture liveness properties in the sense explained in the introduction.

In a previous paper [13] we already gave such a semantics to ABCdE. The rules for the successor relation presented in [13], displayed in Tables 2 and 5, are now seen to fit our congruence format. We can now also conclude that ep-bisimulation is a lean congruence for ABCdE. In [14, Appendix B] we contemplate a very different approach for defining the relation $\leadsto$. Following [10], 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 $\leadsto$-relation is defined. It is shown that this leads to the same $\leadsto$-relation as the operational approach from [13] and Tables 2 and 5.

[3]This yields $1^2+2\cdot 1+5\cdot 3+3\cdot 2+2\cdot 1=26$ rules of types $(\mathbf{0},0)$, $(\alpha.\_,1)$, $(+,2)$, $(\hat{~}s,1)$ and $\langle X|S\rangle$ not included in Tables 2 and 5.

References

[1] L. Aceto, W. Fokkink & C. Verhoef (2000): *Structural Operational Semantics*. In J. Bergstra, A. Ponse & S. Smolka, editors: *Handbook of Process Algebra*, chapter 3, Springer, pp. 197–292.

[2] D. Austry & G. Boudol (1984): *Algèbre de Processus et Synchronisation*. *Theoretical Computer Science* 30, pp. 91–131, doi:10.1016/0304-3975(84)90067-7.

[3] J.C.M. Baeten & W.P. Weijland (1990): *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, doi:10.1017/CBO9780511624193.

[4] J.A. Bergstra (1988): *ACP with signals*. In J. Grabowski, P. Lescanne & W. Wechler, editors: Proc. International Workshop on *Algebraic and Logic Programming*, LNCS 343, Springer, pp. 11–20, doi:10.1007/3-540-50667-5_53.

[5] S.D. Brookes, C.A.R. Hoare & A.W. Roscoe (1984): *A theory of communicating sequential processes*. *Journal of the ACM* 31(3), pp. 560–599, doi:10.1145/828.833.

[6] F. Corradini, M.R. Di Berardini & W. Vogler (2009): *Time and Fairness in a Process Algebra with Non-blocking Reading*. In M. Nielsen, A. Kucera, P.B. Miltersen, C. Palamidessi, P. Tuma & F.D. Valencia, editors: *Theory and Practice of Computer Science*, SOFSEM'09, LNCS 5404, Springer, pp. 193–204, doi:10.1007/978-3-540-95891-8_20.

[7] V. Dyseryn, R.J. van Glabbeek & P. Höfner (2017): *Analysing Mutual Exclusion using Process Algebra with Signals*. In K. Peters & S. Tini, editors: Proceedings Combined 24th International Workshop on *Expressiveness in Concurrency* and 14th Workshop on *Structural Operational Semantics*, Berlin, Germany, 4th September 2017, *Electronic Proceedings in Theoretical Computer Science* 255, Open Publishing Association, pp. 18–34, doi:10.4204/EPTCS.255.2.

[8] R.J. van Glabbeek (1994): *On the expressiveness of ACP (extended abstract)*. In A. Ponse, C. Verhoef & S.F.M. van Vlijmen, editors: Proceedings First Workshop on the *Algebra of Communicating Processes*, ACP'94, Utrecht, The Netherlands, May 1994, Workshops in Computing, Springer, pp. 188–217, doi:10.1007/978-1-4471-2120-6_8.

[9] R.J. van Glabbeek (2017): *Lean and Full Congruence Formats for Recursion*. In: Proceedings 32nd Annual ACM/IEEE Symposium on *Logic in Computer Science*, LICS 2017, IEEE Computer Society Press, doi:10.1109/LICS.2017.8005142.

[10] R.J. van Glabbeek (2019): *Justness: A Completeness Criterion for Capturing Liveness Properties (extended abstract)*. In M. Bojańczyk & A. Simpson, editors: Proceedings 22st International Conference on *Foundations of Software Science and Computation Structures* (FoSSaCS'19); held as part of the *European Joint Conferences on Theory and Practice of Software* (ETAPS'19), Prague, Czech Republic, April 2019, LNCS 11425, Springer, pp. 505–522, doi:10.1007/978-3-030-17127-8_29.

[11] R.J. van Glabbeek & P. Höfner (2015): *Progress, Fairness and Justness in Process Algebra*. Technical Report 8501, NICTA, Sydney, Australia. Available at `http://arxiv.org/abs/1501.03268`.

[12] R.J. van Glabbeek & P. Höfner (2019): *Progress, Justness and Fairness*. *ACM Computing Surveys* 52(4):69, doi:10.1145/3329125.

[13] R.J. van Glabbeek, P. Höfner & W. Wang (2021): *Enabling Preserving Bisimulation Equivalence*. In S. Haddad & D. Varacca, editors: Proceedings 32nd International Conference on *Concurrency Theory*, CONCUR'21, *Leibniz International Proceedings in Informatics (LIPIcs)* 203, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, doi:10.4230/LIPIcs.CONCUR.2021.33.

[14] R.J. van Glabbeek, P. Höfner & W. Wang (2021): *Enabling Preserving Bisimulation Equivalence*. Available at `https://arxiv.org/abs/2108.00142`. Full version of [13].

[15] R.J. van Glabbeek, P. Höfner & W. Wang (2023): *A Lean-Congruence Format for EP-Bisimilarity*. arXiv:2308.16350. Full version of this paper.

[16] J.F. Groote & F.W. Vaandrager (1992): *Structured Operational Semantics and Bisimulation as a Congruence*. *Information and Computation* 100(2), pp. 202–260, doi:10.1016/0890-5401(92)90013-6.

[17] R. Milner (1990): *Operational and algebraic semantics of concurrent processes*. In J. van Leeuwen, editor: *Handbook of Theoretical Computer Science*, chapter 19, Elsevier Science Publishers B.V. (North-Holland), pp. 1201–1242. 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.

[18] R. Milner (1983): *Calculi for Synchrony and Asynchrony*. *Theoretical Computer Science* 25, pp. 267–310, doi:10.1016/0304-3975(83)90114-7.

[19] M.R. Mousavi, M.A. Reniers & J.F. Groote (2007): *SOS formats and meta-theory: 20 years after*. *Theoretical Computer Science* 373(3), pp. 238–272, doi:10.1016/j.tcs.2006.12.019.

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

[21] A. Rensink (2000): *Bisimilarity of Open Terms*. *Information and Computation* 156(1-2), pp. 345–385, doi:10.1006/inco.1999.2818.

[22] R. de Simone (1985): *Higher-level synchronising devices in* MEIJE-*SCCS*. *Theoretical Computer Science* 37, pp. 245–267, doi:10.1016/0304-3975(85)90093-3.