Justness
A Completeness Criterion for Capturing Liveness Properties (extended abstract)

Rob van Glabbeek[1,2]

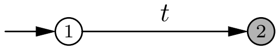
[1] Data61, CSIRO, Australia
[2] Computer Science and Engineering, University of New South Wales, Australia

Abstract. This paper poses that transition systems constitute a good model of distributed systems only in combination with a criterion telling which paths model complete runs of the represented systems. Among such criteria, progress is too weak to capture relevant liveness properties, and fairness is often too strong; for typical applications we advocate the intermediate criterion of justness. Previously, we proposed a definition of justness in terms of an asymmetric concurrency relation between transitions. Here we define such a concurrency relation for the transition systems associated to the process algebra CCS as well as its extensions with broadcast communication and signals, thereby making these process algebras suitable for capturing liveness properties requiring justness.

1 Introduction

Transition systems are a common model for distributed systems. They consist of sets of states, also called *processes*, and transitions—each transition going from a source state to a target state. A given distributed system $\mathcal{D}$ corresponds to a state P in a transition system $\mathbb{T}$—the initial state of $\mathcal{D}$. The other states of $\mathcal{D}$ are the processes in $\mathbb{T}$ that are reachable from P by following the transitions. A run of $\mathcal{D}$ corresponds with a *path* in $\mathbb{T}$: a finite or infinite alternating sequence of states and transitions, starting with P, such that each transition goes from the state before to the state after it. Whereas each finite path in $\mathbb{T}$ starting from P models a *partial run* of $\mathcal{D}$, i.e., an initial segment of a (complete) run, typically not each path models a run. Therefore a transition system constitutes a good model of distributed systems only in combination with what we here call a *completeness criterion*: a selection of a subset of all paths as *complete paths*, modelling runs of the represented system.

A *liveness property* says that "something [good] must happen" eventually [18]. Such a property holds for a distributed system if the [good] thing happens in each of its possible runs. One of the ways to formalise this in terms of transition systems is to postulate a set of good states $\mathscr{G}$, and say that the liveness property $\mathscr{G}$ holds for the process P if all complete paths starting in P pass through a state of $\mathscr{G}$ [16]. Without a completeness criterion the concept of a liveness property appears to be meaningless.



Example 1 The transition system on the right

models Cataline eating a croissant in Paris. It abstracts from all activity in the world except the eating of that croissant, and thus has two states only—the states of the world before and after this event—and one transition t. We depict states by circles and transitions by arrows between them. An initial state is indicated by a short arrow without a source state. A possible liveness property says that the croissant will be eaten. It corresponds with the set of states $\mathscr{G}$ consisting of state 2 only. The states of $\mathscr{G}$ are indicated by shading.

The depicted transition system has three paths starting with state 1: 1, $1\,t$ and $1\,t\,2$. The path $1\,t\,2$ models the run in which Cataline finishes the croissant. The path 1 models a run in which Cataline never starts eating the croissant, and the path $1\,t$ models a run in which Cataline starts eating it, but never finishes. The liveness property $\mathscr{G}$ holds only when using a completeness criterion that rules out the paths 1 and $1\,t$ as modelling actual runs of the system, leaving $1\,t\,2$ as the sole complete path. ¶

The transitions of transition systems can be understood to model atomic actions that can be performed by the represented systems. Although we allow these actions to be instantaneous or durational, in the remainder of this paper we adopt the assumption that "atomic actions always terminate" [23]. This is a partial completeness criterion. It rules out the path $1\,t$ in Example 1. We build in this assumption in the definition of a path by henceforth requiring that finite paths should end with a state.

Progress. The most widely employed completeness criterion is *progress.*[1] In the context of *closed systems*, having no run-time interactions with the environment, it is the assumption that a run will never get stuck in a state with outgoing transitions. This rules out the path 1 in Example 1, as t is outgoing. When adopting progress as completeness criterion, the liveness property $\mathscr{G}$ holds for the system modelled in Example 1.

Progress is assumed in almost all work on process algebra that deals with liveness properties, mostly implicitly. Milner makes an explicit progress assumption for the process algebra CCS in [20]. A progress assumption is built into the temporal logics LTL [24], CTL [7] and CTL* [8], namely by disallowing states without outgoing transitions and evaluating temporal formulas by quantifying over infinite paths only.[2] In [17] the 'multiprogramming axiom' is a progress assumption, whereas in [1] progress is assumed as a 'fundamental liveness property'.

As we argued in [10,15,16], a progress assumption as above is too strong in the context of reactive systems, meaning that it rules out as incomplete too many paths. There, a transition typically represents an interaction between the

[1] Misra [21,22] calls this the 'minimal progress assumption'. In [22] he uses 'progress' as a synonym for 'liveness'. In session types, 'progress' and 'global progress' are used as names of particular liveness properties [4]; this use has no relation with ours.

[2] Exceptionally, states without outgoing transitions are allowed, and then quantification is over all *maximal* paths, i.e. paths that are infinite or end in a state without outgoing transitions [5].

distributed system being modelled and its environment. In many cases a transition can occur only if both the modelled system *and* the environment are ready to engage in it. We therefore distinguish *blocking* and *non-blocking* transitions. A transition is non-blocking if the environment cannot or will not block it, so that its execution is entirely under the control of the system under consideration. A blocking transition on the other hand may fail to occur because the environment is not ready for it. The same was done earlier in the setting of Petri nets [26], where blocking and non-blocking transitions are called *cold* and *hot*, respectively.

In [10,15,16] we worked with transition systems that are equipped with a partitioning of the transitions into blocking and non-blocking ones, and reformulated the progress assumption as follows:

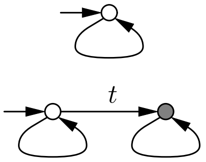
> *a (transition) system in a state that admits a non-blocking transition will eventually progress, i.e., perform a transition.*

In other words, a run will never get stuck in a state with outgoing non-blocking transitions. In Example 1, when adopting progress as our completeness criterion, we assume that Cataline actually wants to eat the croissant, and does not willingly remain in State 1 forever. When that assumption is unwarranted, one would model her behaviour by a transition system different from that of Example 1. However, she may still be stuck in State 1 by lack of any croissant to eat. If we want to model the capability of the environment to withhold a croissant, we classify t as a blocking transition, and the liveness property $\mathscr{G}$ does not hold. If we abstract from a possible shortage of croissants, t is deemed a non-blocking transition, and, when assuming progress, $\mathscr{G}$ holds.

As an alternative approach to a dogmatic division of transitions in a transition system, we could shift the status of transitions to the progress property, and speak of B-progress when B is the set of blocking transitions. In that approach, $\mathscr{G}$ holds for State 1 of Example 1 under the assumption of B-progress when $t \notin B$, but not when $t \in B$.

Justness. Justness is a completeness criterion proposed in [10,15,16]. It strengthens progress. It can be argued that once one adopts progress it makes sense to go a step further and adopt even justness.

Example 2 The transition system on the top right models Alice making an unending sequence of phone calls in London. There is no interaction of any kind between Alice and Cataline. Yet, we may chose to abstracts from all activity in the world except the eating of the croissant by Cataline, and the making of calls by Alice. This yields the combined transition system on the bottom right. Even when taking the transition t to be non-blocking, progress is not a strong enough completeness criterion to ensure that Cataline will ever eat the croissant. For the infinite path that loops in the first state is complete. Nevertheless, as nothing stops Cataline from making progress, in reality t will occur. [16]



This example is not a contrived corner case, but a rather typical illustration of an issue that is central to the study of distributed systems. Other illustrations of this phenomena occur in [10, Section 9.1], [14, Section 10], [11, Section 1.4], [12] and [6, Section 4]. The criterion of justness aims to ensure the liveness property occurring in these examples. In [16] it is formulated as follows:

> *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.*

In Example 2, t is a non-blocking transition enabled in the initial state. It stems from the single parallel component Cataline of the distributed system under consideration. Justness therefore requires that Cataline must partake in a transition. This can only be t, as all other transitions involve component Alice only. Hence justness says that t must occur. The infinite path starting in the initial state and not containing t is ruled out as unjust, and thereby incomplete.

In [16,13] we explain how justness is fundamentally different from fairness, and why fairness is too strong a completeness criterion for many applications.

Unlike progress, the concept of justness as formulated above is in need of some formalisation, i.e., to formally define a component, to make precise for concrete transition systems what it means for a transition to stem from a set of components, and to define when a component partakes in a transition.

A formalisation of justness for the transition system generated by the process algebra AWN, the *Algebra for Wireless Networks* [9], was provided in [10]. In the same vain, [15] offered a formalisation for the transition systems generated by CCS [20], and its extension ABC, the *Algebra of Broadcast Communication* [15], a variant of CBS, the *Calculus of Broadcasting Systems* [25]. The same was done for CCS extended with *signals* in [6]. These formalisations coinductively define *B-justness*, where B ranges over sets of transitions deemed to be blocking, as a family of predicates on paths, and proceed by a case distinction on the operators in the language. Although these definitions *do* capture the concept of justness formulated above, it is not easy to see why.

A more syntax-independent formalisation of justness occurs in [16]. There it is defined directly on transition systems equipped with a, possibly asymmetric, concurrency relation between transitions. However, the concurrency relation itself is defined only for the transition system generated by a fragment of CCS, and the generalisation to full CCS, and other process algebras, is non-trivial.

It is the purpose of this paper to make the definition of justness from [16] available to a large range of process algebras by defining the concurrency relation for CCS, for ABC, and for the extension of CCS with signals used in [6]. We do this in a precise as well as in an approximate way, and show that both approaches lead to the same concept of justness. Moreover, in all cases we establish a closure property on the concurrency relation ensuring that justness is a meaningful notion. We show that for all these algebras justness is *feasible*. Here feasibility is a requirement on completeness criteria advocated in [1,19,16]. Finally, we establish agreement between the formalisation of justness from [16] and the present paper, and the original coinductive ones from [15] and [6].

2 Labelled transition systems with concurrency

We start with the formal definitions of a labelled transition system, a path, and the completeness criterion *progress*, which is parametrised by the choice of a collection B of blocking actions. Then we define the completeness criterion *justness* on labelled transition system upgraded with a concurrency relation.

Definition 1 A *labelled transition system* (LTS) is a tuple $(S, \mathit{Tr}, \mathit{src}, \mathit{target}, \ell)$ with S and Tr sets (of *states* and *transitions*), $\mathit{src}, \mathit{target}\colon \mathit{Tr} \rightarrow S$ and $\ell : \mathit{Tr} \rightarrow \mathscr{L}$, for some set of transition labels $\mathscr{L}$.

Here we work with LTSs labelled over a structured set of labels $(\mathscr{L}, \mathit{Act}, \mathit{Rec})$, where $\mathit{Rec} \subseteq \mathit{Act} \subseteq \mathscr{L}$. Labels in Act are *actions*; the ones in $\mathscr{L} \setminus \mathit{Act}$ are *signals.* Transitions labelled with actions model a state chance in the represented system; signal transitions do not—they satisfy $\mathit{src}(t) = \mathit{target}(t)$ and merely convey a property of a state. $\mathit{Rec} \subseteq \mathit{Act}$ is the set of *receptive* actions; sets $B \subseteq \mathit{Act}$ of blocking actions must always contain Rec. In CCS and most other process algebras $\mathit{Rec} = \emptyset$ and $\mathit{Act} = \mathscr{L}$. Let $\mathit{Tr}^{\bullet} = \{t \in \mathit{Tr} \mid \ell(t) \in \mathit{Act} \setminus \mathit{Rec}\}$ be the set of transitions that are neither signals nor receptive.

Definition 2 A *path* in a transition system $(S, \mathit{Tr}, \mathit{src}, \mathit{target})$ is an alternating sequence $s_0\, t_1\, s_1\, t_2\, s_2 \cdots$ of states and non-signal transitions, starting with a state and either being infinite or ending with a state, such that $\mathit{src}(t_i) = s_{i-1}$ and $\mathit{target}(t_i) = s_i$ for all relevant i.

A *completeness criterion* is a unary predicate on the paths in a transition system.

Definition 3 Let $B \subseteq \mathit{Act}$ be a set of actions with $\mathit{Rec} \subseteq B$—the *blocking* ones. Then $\mathit{Tr}^{\bullet}_{\neg B} := \{t \in \mathit{Tr}^{\bullet} \mid \ell(t) \notin B\}$ is the set of *non-blocking* transitions. A path in $\mathbb{T}$ is *B-progressing* if either it is infinite or its last state is the source of no non-blocking transition $t \in \mathit{Tr}^{\bullet}_{\neg B}$.

B-progress is a completeness criterion for any choice of $B \subseteq \mathit{Act}$ with $\mathit{Rec} \subseteq B$.

Definition 4 A *labelled transition system with concurrency* (LTSC) is a tuple $(S, \mathit{Tr}, \mathit{src}, \mathit{target}, \ell, \smile^{\bullet})$ consisting of a LTS $(S, \mathit{Tr}, \mathit{src}, \mathit{target}, \ell)$ and a *concurrency relation* $\smile^{\bullet} \subseteq \mathit{Tr}^{\bullet} \times \mathit{Tr}$, such that:

$$t \not\smile^{\bullet} t \text{ for all } t \in \mathit{Tr}^{\bullet}, \tag{1}$$

if $t \in \mathit{Tr}^{\bullet}$ and π is a path from $\mathit{src}(t)$ to $s \in S$ such that $t \smile^{\bullet} v$ for all transitions v occurring in π, then there is a $u \in \mathit{Tr}^{\bullet}$ such that $\mathit{src}(u) = s$, $\ell(u) = \ell(t)$ and $t \not\smile^{\bullet} u$. (2)

Informally, $t \smile^{\bullet} v$ means that the transition v does not interfere with t, in the sense that it does not affect any resources that are needed by t, so that in a state where t and v are both possible, after doing v one can still do (a future variant u of) t. In many transition systems $\smile^{\bullet}$ is a symmetric relation, denoted $\smile$.

The transition relation in a labelled transition system is often defined as a relation $Tr \subseteq S \times \mathscr{L} \times S$. This approach is not suitable here, as we will encounter multiple transitions with the same source, target and label that ought to be distinguished based on their concurrency relations with other transitions.

Definition 5 A path π in an LTSC is *B-just*, for $Rec \subseteq B \subseteq Act$, if for each transition $t \in Tr^{\bullet}_{\neg B}$ with $s := src(t) \in \pi$, a transition u occurs in the suffix of π starting at s, such that $t \not\smile^{\bullet} u$.

Informally, justness requires that once a non-blocking non-signal transition t is enabled, sooner or later a transition u will occur that interferes with it, possibly t itself. Note that, for any $Rec \subseteq B \subseteq Act$, B-justness is a completeness criterion stronger than B-progress.

Components. Instead of introducing $\smile^{\bullet}$ as a primitive, it is possible to obtain it as a notion derived from two functions $npc, afc : Tr \rightarrow \mathscr{P}(\mathscr{C})$, for a given set of *components* $\mathscr{C}$. These functions could then be added as primitives to the definition of an LTS. They are based on the idea that a process represents a system built from parallel components. Each transition is obtained as a synchronisation of activities from some of these components. Now $npc(t)$ describes the (nonempty) set of components that are *necessary participants* in the execution of t, whereas $afc(t)$ describes the components that are *affected* by the execution of t. The concurrency relation is then defined by

$$t \smile^{\bullet} u \quad \Leftrightarrow \quad npc(t) \cap afc(u) = \emptyset$$

saying that u interferes with t iff a necessary participant in t is affected by u.

Most material above stems from [16]. However, there $Tr^{\bullet} = Tr$, so that $\smile^{\bullet}$ is irreflexive, i.e., $npc(t) \cap afc(t) \neq \emptyset$ for all $t \in Tr$. Moreover, a fixed set B is postulated, so that the notions of progress and justness are not explicitly parametrised with the choice of B. Furthermore, property (2) is new here; it is the weakest closure property that supports Theorem 1 below. In [16] only the model in which $\smile^{\bullet}$ is derived from npc and afc comes with a closure property:

$$\begin{array}{l}\text{If } t, v \in Tr^{\bullet} \text{ with } src(t) = src(v) \text{ and } npc(t) \cap afc(v) = \emptyset, \text{ then} \\ \exists u \in Tr^{\bullet} \text{ with } src(u) = target(v),\ \ell(u) = \ell(t) \text{ and } npc(u) = npc(t).\end{array} \tag{3}$$

Trivially (3) implies (2).

An important requirement on completeness criteria is that any finite path can be extended into a complete path. This requirement was proposed by Apt, Francez & Katz in [1] and called *feasibility.* It also appears in Lamport [19] under the name *machine closure.* The theorem below list conditions under which B-justness is feasible. Its proof is a variant of a similar theorem from [16] showing conditions under which notions of strong and weak fairness are feasible.

Theorem 1 If, in an LTSC with set of blocking actions B, only countably many transitions from $Tr^{\bullet}_{\neg B}$ are enabled in each state, then B-justness is feasible.

All proofs can found in the full version of this paper [13].

Table 1. Structural operational semantics of CCS

$\alpha.P \xrightarrow{\alpha} P$ (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)
$\dfrac{P \xrightarrow{\ell} P'}{P\backslash L \xrightarrow{\ell} P'\backslash L}$ $(\ell,\bar{\ell} \notin L)$ (Res)	$\dfrac{P \xrightarrow{\ell} P'}{P[f] \xrightarrow{f(\ell)} P'[f]}$ (Rel)	$\dfrac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'}$ $(A \overset{def}{=} P)$ (Rec)

3 CCS and its extensions with broadcast and signals

This section presents four process algebras: Milner's *Calculus of Communicating Systems* (CCS) [20], its extensions with broadcast communication ABC [15] and signals CCSS [6], and an alternative presentation of ABC that avoids negative premises in favour of *discard* transitions.

3.1 CCS

CCS [20] is parametrised with sets $\mathscr{A}$ of *agent identifiers* and $\mathscr{C}_h$ of *(handshake communication) names*; each $A \in \mathscr{A}$ comes with a defining equation $A \overset{def}{=} P$ with P being a CCS expression as defined below. $\bar{\mathscr{C}}_h := \{\bar{c} \mid c \in \mathscr{C}_h\}$ is the set of *co-names*. Complementation is extended to $\bar{\mathscr{C}}_h$ by setting $\bar{\bar{c}} = c$. $Act := \mathscr{C}_h \mathbin{\dot\cup} \bar{\mathscr{C}}_h \mathbin{\dot\cup} \{\tau\}$ is the set of *actions*, where τ is a special *internal action*. Below, c ranges over $\mathscr{C}_h \cup \bar{\mathscr{C}}_h$, η, α, ℓ over Act, and A, B over $\mathscr{A}$. A *relabelling* is a function $f\colon \mathscr{C}_h \to \mathscr{C}_h$; it extends to Act by $f(\bar{c}) = \overline{f(c)}$ and $f(\tau) := \tau$. The set $\mathbb{P}_{\rm CCS}$ of CCS expressions or *processes* is the smallest set including:

$\mathbf{0}$		*inaction*
$\alpha.P$	for $\alpha \in Act$ and $P \in \mathbb{P}_{\rm CCS}$	*action prefixing*
$P+Q$	for $P,Q \in \mathbb{P}_{\rm CCS}$	*choice*
$P\|Q$	for $P,Q \in \mathbb{P}_{\rm CCS}$	*parallel composition*
$P\backslash L$	for $L \subseteq \mathscr{C}_h$ and $P \in \mathbb{P}_{\rm CCS}$	*restriction*
$P[f]$	for f a relabelling and $P \in \mathbb{P}_{\rm CCS}$	*relabelling*
A	for $A \in \mathscr{A}$	*agent identifier*

One often abbreviates $\alpha.\mathbf{0}$ by α, and $P\backslash\{c\}$ by $P\backslash c$. The traditional semantics of CCS is given by the labelled transition relation $\to\ \subseteq \mathbb{P}_{\rm CCS} \times Act \times \mathbb{P}_{\rm CCS}$, where transitions $P \xrightarrow{\ell} Q$ are derived from the rules of Table 1.

3.2 ABC—The Algebra of Broadcast Communication

The Algebra of Broadcast Communication (ABC) [15] is parametrised with sets $\mathscr{A}$ of *agent identifiers*, $\mathscr{B}$ of *broadcast names* and $\mathscr{C}_h$ of *handshake communica-*

Table 2. Structural operational semantics of ABC broadcast communication

(Bro-l)	(Bro-c)	(Bro-r)
$\dfrac{P \xrightarrow{b\sharp_1} P',\ Q \stackrel{b?}{\not\longrightarrow}}{P\|Q \xrightarrow{b\sharp_1} P'\|Q}$	$\dfrac{P \xrightarrow{b\sharp_1} P',\ Q \xrightarrow{b\sharp_2} Q'}{P\|Q \xrightarrow{b\sharp} P'\|Q'}{\scriptstyle \sharp_1\circ\sharp_2=\sharp\neq\_}$ with $\begin{array}{c\|cc} \circ & ! & ? \\ \hline ! & \_ & ! \\ ? & ! & ? \end{array}$	$\dfrac{P \stackrel{b?}{\not\longrightarrow},\ Q \xrightarrow{b\sharp_2} Q'}{P\|Q \xrightarrow{b\sharp_2} P\|Q'}$

tion names; each $A \in \mathscr{A}$ comes with a defining equation $A \stackrel{def}{=} P$ with P being a guarded ABC expression as defined below.

The collections $\mathscr{B}!$ and $\mathscr{B}?$ of *broadcast* and *receive* actions are given by $\mathscr{B}\sharp := \{b\sharp \mid b \in \mathscr{B}\}$ for $\sharp \in \{!, ?\}$. $Act := \mathscr{B}! \mathbin{\dot\cup} \mathscr{B}? \mathbin{\dot\cup} \mathscr{C}_h \mathbin{\dot\cup} \bar{\mathscr{C}}_h \mathbin{\dot\cup} \{\tau\}$ is the set of *actions*. Below, A ranges over $\mathscr{A}$, b over $\mathscr{B}$, c over $\mathscr{C}_h \cup \bar{\mathscr{C}}_h$, η over $\mathscr{C}_h \cup \bar{\mathscr{C}}_h \cup \{\tau\}$ and α, ℓ over Act. A *relabelling* is a function $f : (\mathscr{B} \to \mathscr{B}) \cup (\mathscr{C}_h \to \mathscr{C}_h)$. It extends to Act by $f(\bar{c}) = \overline{f(c)}$, $f(b\sharp) = f(b)\sharp$ and $f(\tau) := \tau$. The set $\mathbb{P}_{\rm ABC}$ of ABC expressions is defined exactly as $\mathbb{P}_{\rm CCS}$. An expression is guarded if each agent identifier occurs within the scope of a prefixing operator. The structural operational semantics of ABC is the same as the one for CCS (see Table 1) but augmented with the rules for broadcast communication in Table 2.

ABC is CCS augmented with a formalism for broadcast communication taken from the Calculus of Broadcasting Systems (CBS) [25]. The syntax without the broadcast and receive actions and all rules except (Bro-l), (Bro-c) and (Bro-r) are taken verbatim from CCS. However, the rules now cover the different name spaces; (Act) for example allows labels of broadcast and receive actions. The rule (Bro-c)—without rules like (Par-l) and (Par-r) with label $b!$—implements a form of broadcast communication where any broadcast $b!$ 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. In order to ensure associativity of the parallel composition, one also needs this rule for components receiving at the same time ($\sharp_1{=}\sharp_2{=}?$). The rules (Bro-l) and (Bro-r) are added to make broadcast communication *non-blocking*: without them a component could be delayed in performing a broadcast simply because one of the other components is not ready to receive it.

3.3 CCS with signals

CCS with signals (CCSS) [6] is CCS extended with a signalling operator $P\hat{\ }s$. Informally, $P\hat{\ }s$ emits the signal s to be read by another process. $P\hat{\ }s$ could for instance be a traffic light emitting the signal *red*. The reading of the signal emitted by $P\hat{\ }s$ does not interfere with any transition of P, such as jumping to *green*. Formally, CCS is extended with a set $\mathscr{S}$ of *signals*, ranged over by s and r. In CCSS the set of actions is defined as $Act := \mathscr{S} \mathbin{\dot\cup} \mathscr{C}_h \cup \bar{\mathscr{C}}_h \mathbin{\dot\cup} \{\tau\}$, and the set of labels by $\mathscr{L} := Act \mathbin{\dot\cup} \bar{\mathscr{S}}$, where $\bar{\mathscr{S}} := \{\bar{s} \mid s \in \mathscr{S}\}$. A relabelling is a function $f : (\mathscr{S} \to \mathscr{S}) \cup (\mathscr{C}_h \to \mathscr{C}_h)$. It extends to $\mathscr{L}$ by $f(\bar{c}) = \overline{f(c)}$ for $c \in \mathscr{C}_h \cup \mathscr{S}$ and $f(\tau) := \tau$. The set $\mathbb{P}_{\rm CCSS}$ of CCSS expressions is defined just as $\mathbb{P}_{\rm CCS}$, but now also $P\hat{\ }s$ is a process for $P \in \mathbb{P}_{\rm CCSS}$ and $s \in \mathscr{S}$, and restriction also covers signals.

Table 3. Structural operational semantics for signals of CCSS

$$
P\hat{}s \xrightarrow{\bar{s}} P\hat{}s \qquad \frac{P \xrightarrow{\bar{s}} P'}{P+Q \xrightarrow{\bar{s}} P'+Q} \qquad \frac{Q \xrightarrow{\bar{s}} Q'}{P+Q \xrightarrow{\bar{s}} P+Q'}
$$

$$
\frac{P \xrightarrow{\alpha} P'}{P\hat{}r \xrightarrow{\alpha} P'} \qquad \frac{P \xrightarrow{\bar{s}} P'}{P\hat{}r \xrightarrow{\bar{s}} P'\hat{}r} \qquad \frac{P \xrightarrow{\bar{s}} P'}{A \xrightarrow{\bar{s}} A}\ (A \overset{def}{=} P)
$$

The semantics of CCSS is given by the labelled transition relation $\rightarrow\ \subseteq \mathbb{P}_{\mathrm{CCSS}} \times \mathscr{L} \times \mathbb{P}_{\mathrm{CCSS}}$ derived from the rules of CCS (Table 1), where now η, ℓ range over $\mathscr{L}$, α over Act, c over $\mathscr{C}_h \cup \mathscr{S}$ and $L \subseteq \mathscr{C}_h \cup \mathscr{S}$, augmented with the rules of Table 3. The first rule is the base case showing that a process $P\hat{}s$ emits the signal s. The rule below models the fact that signalling cannot prevent a process from making progress.

The original semantics of CCSS [6] featured unary predicates $P^{\frown s}$ on processes to model that P emits the signal s; here, inspired by [3], these predicates are represented as transitions $P \xrightarrow{\bar{s}} P$. Whereas this leads to a simpler operational semantics, the price paid is that these new *signal transitions* need special treatment in the definition of justness—cf. Definitions 2 and 5.

3.4 Using signals to avoid negative premises in ABC

Finally, we present an alternative operational semantics ABCd of ABC that avoids negative premises. The price to be paid is the introduction of signals that indicate when a state does not admit a receive action.[3] To this end, let $\mathscr{B}{:} := \{b{:} \mid b \in \mathscr{B}\}$ be the set of *broadcast discards*, and $\mathscr{L} := \mathscr{B}{:} \mathbin{\dot{\cup}} Act$ the set of *transition labels*, with Act as in Section 3.2. The semantics is given by the labelled transition relation $\rightarrow\ \subseteq \mathbb{P}_{\mathrm{ABC}} \times \mathscr{L} \times \mathbb{P}_{\mathrm{ABC}}$ derived from the rules of CCS (Table 1), where now c ranges over $\mathscr{C}_h \cup \bar{\mathscr{C}}_h$, η over $\mathscr{C}_h \cup \bar{\mathscr{C}}_h \cup \{\tau\}$, α over Act and ℓ over $\mathscr{L}$, augmented with the rules of Table 4.

Lemma 1 [25] $P \xrightarrow{b:} Q$ iff $Q = P \wedge P \overset{b?}{\not\longrightarrow}$, for $P, Q \in \mathbb{P}_{\mathrm{ABC}}$ and $b \in \mathscr{B}$.

[3] A state P admits an action $\alpha \in Act$ if there exists a transition $P \xrightarrow{\alpha} Q$.

Table 4. SOS of ABC broadcast communication with discard transitions

$$
\mathbf{0} \xrightarrow{b:} \mathbf{0} \qquad \alpha.P \xrightarrow{b:} \alpha.P\ \ (\alpha{\neq}b?) \qquad \frac{P \xrightarrow{b:} P',\ Q \xrightarrow{b:} Q'}{P+Q \xrightarrow{b:} P'+Q'}
$$

$$
\frac{P \xrightarrow{b\sharp_1} P',\ Q \xrightarrow{b\sharp_2} Q'}{P|Q \xrightarrow{b\sharp} P'|Q'}\ \ {\scriptstyle \sharp_1\circ\sharp_2=\sharp\neq\_}\ \text{with}\ \begin{array}{c|ccc} \circ & ! & ? & : \\ \hline ! & \_ & ! & ! \\ ? & ! & ? & ? \\ : & ! & ? & : \end{array} \qquad \frac{P \xrightarrow{b:} P'}{A \xrightarrow{b:} A}\ (A \overset{def}{=} P)
$$

So the structural operational semantics of ABC from Sections 3.2 and 3.4 yield the same labelled transition relation $\longrightarrow$ when transitions labelled b: are ignored. This approach stems from the Calculus of Broadcasting Systems (CBS) [25].

4 An LTS with concurrency for CCS and its extensions

The forthcoming material applies to each of the process algebras from Section 3, or combinations thereof. Let $\mathbb{P}$ be the set of processes in the language.

We allocate an LTS as in Definition 1 to these languages by taking S to be the set $\mathbb{P}$ of processes, and Tr the set of *derivations* t of transitions $P \xrightarrow{\ell} Q$ with $P, Q \in \mathbb{P}$. Of course $src(t){=}P$, $target(t){=}Q$ and $\ell(t){=}\ell$. Here a *derivation* of a transition $P \xrightarrow{\ell} Q$ is a well-founded tree with the nodes labelled by transitions, such that the root has label $P \xrightarrow{\ell} Q$, and if μ is the label of a node and K is the set of labels of the children of this node then $\frac{K}{\mu}$ is an instance of a rule of Tables 1–4.

We take $Rec := \mathscr{B}?$ in ABC and ABCd: broadcast receipts can always be blocked by the environment, namely by not broadcasting the requested message. For CCS and CCSS we take $Rec := \emptyset$, thus allowing environments that can always participate in certain handshakes, and/or always emit certain signals.

Following [15], we give a name to any derivation of a transition: The unique derivation of the transition $\alpha.P \xrightarrow{\alpha} P$ using the rule (Act) is called $\overset{\alpha}{\rightarrow}P$. The unique derivation of the transition $P\hat{}s \xrightarrow{\bar{s}} P\hat{}s$ is called $P^{\rightarrow s}$. The derivation obtained by application of (Comm) or (Bro-c) on the derivations t and u of the premises of that rule is called $t|u$. The derivation obtained by application of (Par-l) or (Bro-l) on the derivation t of the (positive) premise of that rule, and using process Q at the right of $|$, is $t|Q$. In the same way, (Par-r) and (Bro-r) yield $P|u$, whereas (Sum-l), (Sum-r), (Res), (Rel) and (Rec) yield $t{+}Q$, $P{+}t$, $t\backslash L$, $t[f]$ and $A{:}t$. These names reflect syntactic structure: $t|P \neq P|t$ and $(t|u)|v \neq t|(u|v)$.

Table 3, moreover, contributes derivations $t\hat{}r$. The derivations obtained by application of the rules of Table 4 are called $b{:}\mathbf{0}$, $b{:}\alpha.P$, $t+u$, $t|u$ and $A{:}t$, where t and u are the derivations of the premises.

Synchrons. Let $Arg := \{+_{\text{L}}, +_{\text{R}}, |_{\text{L}}, |_{\text{R}}, \backslash L, [f], A{:}, \hat{}r \mid L \subseteq \mathscr{C}_h \wedge f \text{ a relabelling} \wedge A \in \mathscr{A} \wedge r \in \mathscr{S}\}$. A *synchron* is an expression $\sigma(\overset{\alpha}{\rightarrow}P)$ or $\sigma(P^{\rightarrow s})$ or $\sigma(b{:})$ with $\sigma \in Arg^*$, $\alpha \in Act$, $s \in \mathscr{S}$, $P \in \mathbb{P}$ and $b \in \mathscr{B}$. An *argument* $\iota \in Arg$ is applied componentwise to a set Σ of synchrons: $\iota(\Sigma) := \{\iota\varsigma \mid \varsigma \in \Sigma\}$.

The set of synchrons $\varsigma(t)$ of a derivation t of a transition is defined by

$$\begin{array}{lcl lcl lcl}
\varsigma(\overset{\alpha}{\rightarrow}P) &=& \{(\overset{\alpha}{\rightarrow}P)\} & \varsigma(t+Q) &=& +_L\varsigma(t) & \varsigma(P+t) &=& +_R\varsigma(t) \\
\varsigma(t|Q) &=& |_L\varsigma(t) & \varsigma(t|u) &=& |_L\varsigma(t) \cup |_R\varsigma(u) & \varsigma(P|u) &=& |_R\varsigma(u) \\
\varsigma(t\backslash L) &=& \backslash L\,\varsigma(t) & \varsigma(t[f]) &=& [f]\varsigma(t) & \varsigma(A{:}t) &=& A{:}\varsigma(t) \\
\varsigma(P^{\rightarrow s}) &=& \{(P^{\rightarrow s})\} & \varsigma(t\hat{}r) &=& \hat{}r\,\varsigma(t) & & & \\
\varsigma(b{:}\mathbf{0}) &=& \{(b{:})\} & \varsigma(b{:}\alpha.P) &=& \{(b{:})\} & \multicolumn{3}{r}{\varsigma(t+v) = +_L\varsigma(t) \cup +_R\varsigma(v)}
\end{array}$$

Thus, a synchron of t represents a path in the proof-tree t from its root to a leaf. Each transition derivation can be seen as the synchronisation of one or more synchrons. Note that we use the symbol ς as a variable ranging over synchrons, and as the name of a function—context disambiguates.

Example 3 The CCS process $P = \big((c.Q + (d.R|e.S))|\bar{c}.T\big)\backslash c$ has 3 outgoing transitions: $P \xrightarrow{\tau} (Q|T)\backslash c$, $P \xrightarrow{d} ((R|e.S)|\bar{c}.T)\backslash c$ and $P \xrightarrow{e} ((d.R|S)|\bar{c}.T)\backslash c$. Let t_τ, t_d and $t_e \in \mathit{Tr}$ be the unique derivations of these transitions. Then t_τ is a synchronisation of two synchrons, whereas t_d and $t_e \in \mathit{Tr}$ have only one each: $\varsigma(t_\tau) = \{\backslash c\,|_L +_L(\overset{c}{\to}Q), \backslash c\,|_R(\overset{\bar{c}}{\to}T)\}$, $\varsigma(t_d) = \{\backslash c\,|_L +_R |_L(\overset{d}{\to}R)\}$ and $\varsigma(t_e) = \{\backslash c\,|_L +_R |_R(\overset{e}{\to}S)\}$. The derivations t_d and $t_e \in \mathit{Tr}$ can be seen as *concurrent*, because their synchrons come from opposite sides of the same parallel composition; one would expect that after one of them occurs, a variant of the other is still possible. Indeed, there is a transition $((d.R|S)|\bar{c}.T)\backslash c \xrightarrow{d} ((R|S)|\bar{c}.T)\backslash c$. Let t'_d be its unique derivation. The derivation t_d and t'_d are surely different, for they have a different source state. Even their synchrons are different: $\varsigma(t'_d) = \{\backslash c\,|_L\,|_L(\overset{d}{\to}R)\}$. Nevertheless, t'_d can be recognised as a future variant of t_d: its only synchron has merely lost an argument $+_R$. This choice got resolved when taking the transition t_e.

We proceed to formalise the concepts "future variant" and "concurrent" that occur above, by defining two binary relations $\leadsto\, \subseteq \mathit{Tr}^\bullet \times \mathit{Tr}^\bullet$ and $\smile\!\!\!\bullet\, \subseteq \mathit{Tr}^\bullet \times \mathit{Tr}$ such that the following properties hold:

The relation $\leadsto$ is reflexive and transitive. (4)

If $t \leadsto t'$ and $t \mathrel{\smile\!\!\!\bullet} v$, then $t' \mathrel{\smile\!\!\!\bullet} v$. (5)

If $t \mathrel{\smile\!\!\!\bullet} v$ with $\mathit{src}(t) = \mathit{src}(v)$ then $\exists t'$ with $\mathit{src}(t') = \mathit{target}(v)$ and $t \leadsto t'$. (6)

If $t \leadsto t'$ then $\ell(t') = \ell(t)$ and $t \mathrel{\not\smile\!\!\!\bullet} t'$. (7)

With $t \mathrel{\smile\!\!\!\bullet} v$ we mean that the possible occurrence of t is unaffected by the occurrence of v. Although for CCS the relation $\smile\!\!\!\bullet$ is symmetric (and $\mathit{Tr}^\bullet = \mathit{Tr}$), for ABC and CCSS it is not:

Example 4 ([15]) Let P be the process $b!|(b? + c)$, and let t and v be the derivations of the $b!$- and c-transitions of P. The broadcast $b!$ is in our view completely under the control of the left component; it will occur regardless of whether the right component listens to it or not. It so happens that if $b!$ occurs in state P, the right component will listen to it, thereby disabling the possible occurrence of c. For this reason we have $t \mathrel{\smile\!\!\!\bullet} v$ but $v \mathrel{\not\smile\!\!\!\bullet} t$.

Example 5 Let P be the process $a\hat{\ }s|s$, and let t and v be the derivations of the a- and τ-transitions of P. The occurrence of a disrupts the emission of the signal s, thereby disabling the τ-transition. However, reading the signal does not affect the possible occurrence of a. For this reason we have $t \mathrel{\smile\!\!\!\bullet} v$ but $v \mathrel{\not\smile\!\!\!\bullet} t$.

Proposition 1 Assume (4)–(7). Then the LTS $(\mathbb{P}, \mathit{Tr}, \mathit{src}, \mathit{target}, \ell)$, augmented with the concurrency relation $\smile\!\!\!\bullet$, is an LTSC in the sense of Definition 4.

We now proceed to define the relations $\leadsto$ and $\smile\!\!\!\bullet$ on synchrons, and then lift them to derivations. Subsequently, we establish (4)–(7).

The elements $+_L$, $+_R$, A: and $\hat{r}$ of Arg are called *dynamic* [20]; the others are *static*. (Static operators stay around when their arguments perform transitions.) For $\sigma \in \mathit{Arg}^*$ let $\mathit{static}(\sigma)$ be the result of removing all dynamic elements from σ. For $\varsigma = \sigma\upsilon$ with $\upsilon \in \{(\stackrel{\alpha}{\rightarrow}P), (P^{\rightarrow s}), (b{:})\}$ let $\mathit{static}(\varsigma) := \mathit{static}(\sigma)\upsilon$.

Definition 6 A synchron ς' is a *possible successor* of a synchron ς, notation $\varsigma \leadsto \varsigma'$, if either $\varsigma' = \varsigma$, or ς has the form $\sigma_1|_D\varsigma_2$ for some $\sigma_1 \in \mathit{Arg}^*$, $D \in \{L, R\}$ and ς_2 a synchron, and $\varsigma' = \mathit{static}(\sigma_1)|_D\varsigma_2$.

Definition 7 Two synchrons ς and υ are *directly concurrent*, notation $\varsigma \smile_d \upsilon$, if ς has the form $\sigma_1|_D\varsigma_2$ and $\upsilon = \sigma_1|_E\upsilon_2$ with $\{D, E\} = \{L, R\}$. Two synchrons ς' and υ' are *concurrent*, notation $\varsigma' \smile \upsilon'$, if $\exists\varsigma,\upsilon.\, \varsigma' \leftsquigarrow \varsigma \smile_d \upsilon \leadsto \upsilon'$.

Necessary and active synchrons. All synchrons of the form $\sigma(\stackrel{\alpha}{\rightarrow}P)$ are *active*; their execution causes a transition $\alpha.P \xrightarrow{\alpha} P$ in the relevant component of the represented system. Synchrons $\sigma(P^{\rightarrow s})$ and $\sigma(b{:})$ are passive; they are not affecting any state change. Let $a\varsigma(t)$ denote the set of active synchrons of a derivation t. So a transition t is labelled by a signal, i.e. $\ell(t) \notin \mathit{Act}$, iff $a\varsigma(t) = \emptyset$.

Whether a synchron $\varsigma \in \varsigma(t)$ is *necessary* for t to occur is defined only for $t \in \mathit{Tr}^\bullet$. If t is the derivation of a broadcast transition, i.e., $\ell(t) = b!$ for some $b \in \mathscr{B}$, then exactly one synchron $\upsilon \in \varsigma(t)$ is of the form $\sigma(\stackrel{b!}{\rightarrow}P)$, while all the other $\varsigma \in \varsigma(t)$ are of the form $\sigma'(\stackrel{b?}{\rightarrow}Q)$ (or possibly $\sigma'(b{:})$ in ABCd). Only the synchron υ is necessary for the broadcast to occur, as a broadcast is unaffected by whether or not someone listens to it. Hence we define $n\varsigma(t) := \{\upsilon\}$. For all $t \in \mathit{Tr}^\bullet$ with $\ell(t) \notin \mathscr{B}!$ (i.e. $\ell(t) \in \mathscr{S} \cup \mathscr{C}_h \cup \bar{\mathscr{C}}_h \cup \{\tau\}$) we set $n\varsigma(t) := \varsigma(t)$, thereby declaring all synchrons of the derivation necessary.

Definition 8 A derivation $t' \in \mathit{Tr}^\bullet$ is a *possible successor* of a derivation $t \in \mathit{Tr}^\bullet$, notation $t \leadsto t'$, if t and t' have equally many necessary synchrons and each necessary synchron of t' is a possible successor of one of t; i.e., if $|n\varsigma(t)| = |n\varsigma(t')|$ and $\forall\varsigma' \in n\varsigma(t').\, \exists\varsigma \in n\varsigma(t).\, \varsigma \leadsto \varsigma'$.

This implies that the relation $\leadsto$ between $n\varsigma(t)$ and $n\varsigma(u)$ is a bijection.

Definition 9 Derivation $t \in \mathit{Tr}^\bullet$ is *unaffected by* u, notation $t \smile^{\bullet} u$, if $\forall\varsigma \in n\varsigma(t).\, \forall\upsilon \in a\varsigma(u).\, \varsigma \smile \upsilon$.

So t is unaffected by u if no active synchron of u interferes with a necessary synchron of t. Passive synchrons do not interfere at all.

In Example 3 one has $t_d \smile t_e$, $t_d \leadsto t'_d$ and $t'_d \smile t_e$. Here $t \smile u$ denotes $t \smile^{\bullet} u \wedge u \smile^{\bullet} t$.

Proposition 2 The relations $\leadsto$ and $\smile^{\bullet}$ satisfy the properties (4)–(7).

5 Components

This section proposes a concept of system components associated to a transition, with a classification of components as necessary and/or affected. We then define a concurrency relation $\smile^{\!\bullet}_{s}$ in terms of these components closely mirroring Definition 9 in Section 4 of the concurrency relation $\smile^{\!\bullet}$ in terms of synchrons. We show that $\smile^{\!\bullet}$ and $\smile^{\!\bullet}_{s}$, as well as the concurrency relation defined in terms of components in Section 2, give rise to the same concept of justness.

A *static component* is a string $\sigma \in \mathit{Arg}^*$ of static arguments. Let $\mathscr{C}$ be the set of static components. The *static component* $c(\varsigma)$ of a synchron ς is defined to be the largest prefix γ of ς that is a static component.

Let $\mathit{comp}(t) := \{c(\varsigma) \mid \varsigma \in \varsigma(t)\}$ be the set of *static components* of t. Moreover, $\mathit{npc}(t) := \{c(\varsigma) \mid \varsigma \in n\varsigma(t)\}$ and $\mathit{afc}(t) := \{c(\varsigma) \mid \varsigma \in a\varsigma(t)\}$ are the *necessary* and *affected* static components of $t \in \mathit{Tr}$. Since $n\varsigma(t) \subseteq \varsigma(t)$ and $a\varsigma(t) \subseteq \varsigma(t)$, we have $\mathit{npc}(t) \subseteq \mathit{comp}(t)$ and $\mathit{afc}(t) \subseteq \mathit{comp}(t)$.

Two static components γ and δ are *concurrent*, notation $\gamma \smile \delta$, if $\gamma = \sigma_1|_D\gamma_2$ and $\delta = \sigma_1|_E\delta_2$ with $\{D,E\} = \{L,R\}$.

Definition 10 Derivation $t \in \mathit{Tr}^\bullet$ is *statically unaffected by* u, $t \smile^{\!\bullet}_{s} u$, iff $\forall \gamma \in \mathit{npc}(t).\, \forall \delta \in \mathit{afc}(u).\, \gamma \smile \delta$.

Proposition 3 *If $t \smile^{\!\bullet}_{s} u$ then $t \smile^{\!\bullet} u$.*

In Example 3 we have $t_d \smile t_e$ but $t_d \not\smile_s t_e$, for $\mathit{npc}(t_e) = \mathit{comp}(t_e) = \mathit{comp}(t_d) = \mathit{afc}(t_d) = \{\backslash c\,|_L\}$. Here $t \smile_s u$ denotes $t \smile^{\!\bullet}_{s} u \wedge u \smile^{\!\bullet}_{s} t$. Hence the implication of Proposition 3 is strict.

Proposition 4 *The functions npc and afc* : $\mathit{Tr} \rightarrow \mathscr{P}(\mathscr{C})$ *satisfy closure property (3) of Section 2.*

The concurrency relation $\smile^{\!\bullet}_{c}$ defined in terms of static components according to the template in [16], recalled in Section 2, is not identical to $\smile^{\!\bullet}_{s}$:

Definition 11 Let t, u be derivations. Write $t \smile^{\!\bullet}_{c} u$ iff $\mathit{npc}(t) \cap \mathit{afc}(u) = \emptyset$.

Nevertheless, we show that for the study of justness it makes no difference whether justness is defined using the concurrency relation $\smile^{\!\bullet}$, $\smile^{\!\bullet}_{s}$ or $\smile^{\!\bullet}_{c}$.

Theorem 2 *A path is $\smile^{\!\bullet}$-B-just iff it is $\smile^{\!\bullet}_{c}$-B-just iff it is $\smile^{\!\bullet}_{s}$-B-just.*

6 A coinductive characterisation of justness

In this section we show that the $\smile^{\!\bullet}$-based concept of justness defined in this paper coincides with a coinductively defined concept of justness, for CCS and ABC originating from [15]. To state the coinductive definition of justness, we need to define the notion of the decomposition of a path starting from a process with a leading static operator.

Any derivation $t \in \mathit{Tr}$ of a transition with $\mathit{src}(t) = P|Q$ has the shape

- $u|Q$, with $\mathit{target}(t) = \mathit{target}(u)|Q$,
- $u|v$, with $\mathit{target}(t) = \mathit{target}(u)|\mathit{target}(v)$,
- or $P|v$, with $\mathit{target}(t) = P|\mathit{target}(v)$.

Let a path *of* a process P be a path as in Definition 2 starting with P. Now the *decomposition* of a path π of $P|Q$ into paths π_1 and π_2 of P and Q, respectively, is obtained by concatenating all left-projections of the states and transitions of π into a path of P and all right-projections into a path of Q—notation $\pi \Rrightarrow \pi_1|\pi_2$. Here it could be that π is infinite, yet either π_1 or π_2 (but not both) are finite.

Likewise, $t \in \mathit{Tr}$ with $\mathit{src}(t) = P[f]$ has the shape $u[f]$ with $\mathit{target}(t) = \mathit{target}(u)[f]$. The *decomposition* π' of a path π of $P[f]$ is the path obtained by leaving out the outermost $[f]$ of all states and transitions in π, notation $\pi \Rrightarrow \pi'[f]$. In the same way one defines the decomposition of a path of $P\backslash c$.

The following co-inductive definition of the family B-justness of predicates on paths, with one family member of each choice of a set B of blocking actions, stems from [15, Appendix E]—here $\bar{D} := \{\bar{c} \mid c \in D\}$.

Definition 12 *B-justness*, for $\mathscr{B}? \subseteq B \subseteq \mathit{Act}$, is the largest family of predicates on the paths in the LTS of ABC such that

- a finite B-just path ends in a state that admits actions from B only;
- a B-just path of a process $P|Q$ can be decomposed into a C-just path of P and a D-just path of Q, for some $C, D \subseteq B$ such that $\tau \in B \vee C \cap \bar{D} = \emptyset$;
- a B-just path of $P\backslash L$ can be decomposed into a $B \cup L \cup \bar{L}$-just path of P;
- a B-just path of $P[f]$ can be decomposed into an $f^{-1}(B)$-just path of P;
- and each suffix of a B-just path is B-just.

Intuitively, justness is a completeness criterion, telling which paths can actually occur as runs of the represented system. A path is B-just if it can occur in an environment that may block the actions in B. In this light, the first, third, fourth and fifth requirements above are intuitively plausible. The second requirement first of all says that if $\pi \Rrightarrow \pi_1|\pi_2$ and π can occur in the environment that may block the actions in B, then π_1 and π_2 must be able to occur in such an environment as well, or in environments blocking less. The last clause in this requirement prevents a C-just path of P and a D-just path of Q to compose into a B-just path of $P|Q$ when C contains an action c and D the complementary action $\bar{c}$ (except when $\tau \in B$). The reason is that no environment (except one that can block τ-actions) can block both actions for their respective components, as nothing can prevent them from synchronising with each other.

The fifth requirement helps characterising processes of the form $b+(A|b)$ and $a.(A|b)$, with $A \overset{def}{=} a.A$. Here, the first transition 'gets rid of' the choice and of the leading action a, respectively, and this requirement reduces the justness of paths of such processes to their suffixes.

Example 6 To illustrate Definition 12 consider the unique infinite path of the process Alice|Cataline of Example 2 in which the transition t does not occur. Taking the empty set of blocking actions, we ask whether this path is $\emptyset$-just. If it were, then by the second requirement of Definition 12 the projection of this path on the process Cataline would need to be $\emptyset$-just as well. This is the path 1 (without any transitions) in Example 1. It is not $\emptyset$-just by the first requirement of Definition 12, because its last state 1 admits a transition.

We now establish that the concept of justness from Definition 12 agrees with the concept of justness defined earlier in this paper.

Theorem 3 A path is $\smile\!\!\!\!^{\bullet}_s$-$B$-just iff it is B-just in the sense of Definition 12.

If a path π is B-just then it is C-just for any $C \supseteq B$. Moreover, the collection of sets B such that a given path π is B-just is closed under arbitrary intersection, and thus there is a least set B_π such that π is B-just. Actions $\alpha \in \mathscr{B}_\pi$ are called *π-enabled* [14]. A path is called *just* (without a predicate B) iff it is B-just for some $\mathscr{B}? \subseteq B \subseteq \mathscr{B}? \mathbin{\dot\cup} \mathscr{C}_h \mathbin{\dot\cup} \bar{\mathscr{C}}_h \mathbin{\dot\cup} \mathscr{S}$ [15,14,6,3], which is the case iff it is $\mathscr{B}? \mathbin{\dot\cup} \mathscr{C}_h \mathbin{\dot\cup} \bar{\mathscr{C}}_h \mathbin{\dot\cup} \mathscr{S}$-just.

In [3] a definition of justness for CCS with signal transition appears, very similar to Definition 12; it also applies to CCSS as presented here. Generalising Theorem 3, one can show that a path is ($\smile\!\!\!\!^{\bullet}_s$ or $\smile\!\!\!\!^{\bullet}_c$ or) $\smile\!\!\!\!^{\bullet}$-just iff it is just in this sense. The same holds for the coinductive definition of justness from [6].

7 Conclusion

We advocate justness as a reasonable completeness criterion for formalising liveness properties when modelling distributed systems by means of transition systems. In [16] we proposed a definition of justness in terms of a, possibly asymmetric, concurrency relation between transitions. The current paper defined such a concurrency relation for the transition systems associated to CCS, as well as its extensions with broadcast communication and signals, thereby making the definition of justness from [16] available to these languages. In fact, we provided three versions of the concurrency relation, and showed that they all give rise to the same concept of justness. We expect that this style of definition will carry over to many other process algebras. We showed that justness satisfies the criterion of feasibility, and proved that our formalisation agrees with previous coinductive formalisations of justness for these languages.

Concurrency relations between transitions in transition systems have been studied in [28]. Our concurrency relation $\smile\!\!\!\!^{\bullet}$ follows the same computational intuition. However, in [28] transitions are classified as concurrent or not only when they have the same source, whereas as a basis for the definition of justness here we need to compare transitions with different sources. Apart from that, our concurrency relation is more general in that it satisfies fewer closure properties, and moreover is allowed to be asymmetric.

Concurrency is represented explicitly in models like Petri nets [26], event structures [29], or asynchronous transition systems [27,2,30]. We believe that the semantics of CCS in terms of such models agrees with its semantics in terms of labelled transition systems with a concurrency relation as given here. However, formalising such a claim requires a choice of an adequate justness-preserving semantic equivalence defined on the compared models. Development of such semantic equivalences is a topic for future research.

Acknowledgement. I am grateful to Peter Höfner, Victor Dyseryn and Filippo de Bortoli for valuable feedback.

References

1. K.R. Apt, N. Francez & S. Katz (1988): *Appraising Fairness in Languages for Distributed Programming. Distributed Computing* 2(4), pp. 226–241, https://doi.org/10.1007/BF01872848.
2. M. Bednarczyk (1987): *Categories of asynchronous systems.* Ph.D. thesis, Computer Science, University of Sussex, Brighton.
3. M.S. Bouwman (2018): *Liveness analysis in process algebra: simpler techniques to model mutex algorithms.* Technical Report, Eindhoven University of Technology. Available at `http://www.win.tue.nl/~timw/downloads/bouwman_seminar.pdf`.
4. M. Coppo, M. Dezani-Ciancaglini, L. Padovani & N. Yoshida (2013): *Inference of Global Progress Properties for Dynamically Interleaved Multiparty Sessions.* In: Proc. Coordination'13, LNCS 7890, Springer, pp. 45–59, https://doi.org/10.1007/978-3-642-38493-6_4.
5. R. De Nicola & F.W. Vaandrager (1995): *Three Logics for Branching Bisimulation. Journal of the ACM* 42(2), pp. 458–487, https://doi.org/10.1145/201019.201032.
6. V. Dyseryn, R.J. van Glabbeek & P. Höfner (2017): *Analysing Mutual Exclusion using Process Algebra with Signals.* In K. Peters & S. Tini, editors: Proc. Combined 24th International Workshop on *Expressiveness in Concurrency* and 14th Workshop on *Structural Operational Semantics*, *Electronic Proceedings in Theoretical Computer Science* 255, Open Publishing Association, pp. 18–34, https://doi.org/10.4204/EPTCS.255.2.
7. E.A. Emerson & E.M. Clarke (1982): *Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons. Science of Computer Programming* 2(3), pp. 241–266, https://doi.org/10.1016/0167-6423(83)90017-5.
8. E.A. Emerson & J.Y. Halpern (1986): *'Sometimes' and 'Not Never' revisited: on branching time versus linear time temporal logic. Journal of the ACM* 33(1), pp. 151–178, https://doi.org/10.1145/4904.4999.
9. A. Fehnker, R.J. van Glabbeek, P. Höfner, A.K. McIver, M. Portmann & W.L. Tan (2012): *A Process Algebra for Wireless Mesh Networks.* In H. Seidl, editor: Proc. ESOP'12, LNCS 7211, Springer, pp. 295–315, https://doi.org/10.1007/978-3-642-28869-2_15.
10. A. Fehnker, R.J. van Glabbeek, P. Höfner, A.K. McIver, M. Portmann & W.L. Tan (2013): *A Process Algebra for Wireless Mesh Networks used for Modelling, Verifying and Analysing AODV.* Technical Report 5513, NICTA. Available at `http://arxiv.org/abs/1312.7645`.
11. R.J. van Glabbeek (2015): *Structure Preserving Bisimilarity, Supporting an Operational Petri Net Semantics of CCSP.* In R. Meyer, A. Platzer & H. Wehrheim, editors: Proceedings *Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday*, LNCS 9360, Springer, pp. 99–130, https://doi.org/10.1007/978-3-319-23506-6_9. Available at `http://arxiv.org/abs/1509.05842`.
12. R.J. van Glabbeek (2016): *Ensuring Liveness Properties of Distributed Systems (A Research Agenda).* Position paper. Available at `https://arxiv.org/abs/1711.04240`.
13. R.J. van Glabbeek (2018): *Justness: A Completeness Criterion for Capturing Liveness Properties.* Technical Report, Data61, CSIRO. Available at `http://www.cse.unsw.edu.au/~rvg/synchrons.pdf`. Full version of the present paper.
14. R.J. van Glabbeek & P. Höfner (2015): *CCS: It's not fair! Acta Informatica* 52(2-3), pp. 175–205, https://doi.org/10.1007/s00236-015-0221-6.

15. R.J. van Glabbeek & P. Höfner (2015): *Progress, Fairness and Justness in Process Algebra.* Technical Report 8501, NICTA. Available at `http://arxiv.org/abs/1501.03268`.
16. R.J. van Glabbeek & P. Höfner (2018): *Progress, Justness and Fairness.* Survey paper, Data61, CSIRO, Sydney, Australia. Available at `https://arxiv.org/abs/1810.07414`.
17. R. Kuiper & W.-P. de Roever (1983): *Fairness Assumptions for CSP in a Temporal Logic Framework.* In D. Bjørner, editor: *Formal Description of Programming Concepts II*, North-Holland, pp. 159–170.
18. L. Lamport (1977): *Proving the correctness of multiprocess programs. IEEE Transactions on Software Engineering* 3(2), pp. 125–143, https://doi.org/10.1109/TSE.1977.229904.
19. L. Lamport (2000): *Fairness and hyperfairness. Distributed Computing* 13(4), pp. 239–245, https://doi.org/10.1007/PL00008921.
20. R. Milner (1980): *A Calculus of Communicating Systems.* LNCS 92, Springer, https://doi.org/10.1007/3-540-10235-3.
21. J. Misra (1988): *A Rebuttal of Dijkstra's Position on Fairness.* Available at `http://www.cs.utexas.edu/users/misra/Notes.dir/fairness.pdf`.
22. J. Misra (2001): *A Discipline of Multiprogramming — Programming Theory for Distributed Applications.* Springer, https://doi.org/10.1007/978-1-4419-8528-6.
23. S.S. Owicki & L. Lamport (1982): *Proving Liveness Properties of Concurrent Programs. ACM TOPLAS* 4(3), pp. 455–495, https://doi.org/10.1145/357172.357178.
24. A. Pnueli (1977): *The Temporal Logic of Programs.* In: Proc. 18th Annual Symposium on *Foundations of Computer Science* (FOCS'77), IEEE, pp. 46–57, https://doi.org/10.1109/SFCS.1977.32.
25. K.V.S. Prasad (1991): *A Calculus of Broadcasting Systems.* In S. Abramsky & 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), LNCS 493, Springer, pp. 338–358, https://doi.org/10.1007/3-540-53982-4_19.
26. W. Reisig (2013): *Understanding Petri Nets — Modeling Techniques, Analysis Methods, Case Studies.* Springer, https://doi.org/10.1007/978-3-642-33278-4.
27. M.W. Shields (1985): *Concurrent machines. The Computer Journal* 28(5), pp. 449–465, https://doi.org/10.1093/comjnl/28.5.449.
28. E.W. Stark (1989): *Concurrent transition systems. Theoretical Computer Science* 64(3), pp. 221–269, https://doi.org/10.1016/0304-3975(89)90050-9.
29. G. Winskel (1987): *Event structures.* In W. Brauer, W. Reisig & G. Rozenberg, editors: *Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II; Proceedings of an Advanced Course*, Bad Honnef, September 1986, LNCS 255, Springer, pp. 325–392, https://doi.org/10.1007/3-540-17906-2_31.
30. G. Winskel & M. Nielsen (1995): *Models for Concurrency.* In S. Abramsky, D. Gabbay & T. Maibaum, editors: *Handbook of Logic in Computer Science*, chapter 1, 4: Semantic Modelling, Oxford University Press, pp. 1–148.