Justness
A Completeness Criterion for Capturing Liveness Properties

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

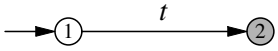
This paper poses that transition systems constitute a good model of distributed systems only in combination with a criterion telling which paths model complete runs of the represented systems. Among such criteria, progress is too weak to capture relevant liveness properties, and fairness is often too 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 $\mathscr{D}$ corresponds to a state P in a transition system $\mathbb{T}$—the initial state of $\mathscr{D}$. The other states of $\mathscr{D}$ are the processes in $\mathbb{T}$ that are reachable from P by following the transitions. A run of $\mathscr{D}$ 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 $\mathscr{D}$, 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 [21]. 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 $\mathscr{G}$ [18]. 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, $1t$ and $1t2$. The path $1t2$ 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 $1t$ models a run in which Cataline starts eating it, but never

An extended abstract of this paper will appear in the proceedings of FoSSaCS'19.

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" [29]. 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 [24]. A progress assumption is built into the temporal logics LTL [30], CTL [8] and CTL* [9], namely by disallowing states without outgoing transitions and evaluating temporal formulas by quantifying over infinite paths only.[2] In [20] the 'multiprogramming axiom' is a progress assumption, whereas in [1] progress is assumed as a 'fundamental liveness property'.

Definition 1 ([18]) Completeness criterion F is *stronger* than completeness criterion H iff F rules out (as incomplete) at least all paths that are ruled out by H.

As we argued in [11, 17, 18], a progress assumption as above is too strong in the context of reactive systems. There, a transition typically represents an interaction between the 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 [32], where blocking and non-blocking transitions are called *cold* and *hot*, respectively.

In [11, 17, 18] 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 $\mathscr{G}$

[1] Misra [25, 26] calls this the 'minimal progress assumption'. In [26] 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.

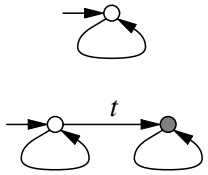
[2] Exceptionally, states without outgoing transitions are allowed, and then quantification is over all *maximal* paths, i.e. paths that are infinite or end in a state without outgoing transitions [5].

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 [11, 17, 18]. 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 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. [18]



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 [11, Section 9.1], [16, Section 10], [12, Section 1.4], [13] and [7, Section 4]. The criterion of justness aims to ensure the liveness property occurring in these examples. In [18] 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.

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* [10], was provided in [11]. In the same vain, [17] offered a formalisation for the transition systems generated by CCS, the *Calculus of Communicating Systems* [24], and its extension ABC, the *Algebra of Broadcast Communication* [17], a variant of CBS, the *Calculus of Broadcasting Systems* [31]. The same was done for CCS extended with *signals* in [7]. These formalisations coinductively define *B-justness*, where B ranges over sets of transitions that are 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, and perhaps more convincing, formalisation of justness occurred in [18]. There it is defined directly on transition systems that are 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 [18] 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 [7]. 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 process algebras justness is *feasible*. Here feasibility is a requirement on completeness criteria advocated in [1, 22, 18]. Finally, we establish agreement between the formalisation of justness from [18] and the present paper, and the original coinductive ones from [17] and [7].

Fairness Fairness assumptions are special kinds of completeness criteria. They postulate that if certain activities *can* happen often enough, they *will* in fact happen.

Example 3 Suppose Bart stands behind a bar and wants to order a beer. But by lack of any formal queueing protocol many other customers get their beer before Bart does. This situation can be modelled as a transition system where in each state in which Bart is not served yet there is an outgoing transition modelling that Bart gets served, but there are also outgoing transitions modelling that someone else gets served instead. The essence of fairness is the assumption that Bart will get his beer eventually. Fairness rules out as unfair, and thereby incomplete, any path in which Bart could have gotten a beer any time, but never will.

Fairness comes in two flavours: *weak* and *strong* fairness. Weak fairness merely rules out paths in which some task is enabled in each state, yet never occurs. Strong fairness also rules out paths in which some task is enabled infinitely often, yet never occurs. Here a *task* is an appropriate set of transitions, in Example 3 all transitions giving Bart a beer. In Example 3 the liveness property that Bart will get a beer holds under the assumption of weak fairness, and thus certainly when assuming strong fairness. It does not hold when merely assuming justness, let alone when merely assuming progress.

Our survey paper [18] proposes a unifying definition of strong and weak fairness, parametrised by the definition of a task. Many notions of fairness found in the literature are cast as instances of this definition, differing only in how to define tasks. The same paper also offers a taxonomy of completeness criteria, ordered by strength (cf. Definition 1). This taxonomy contains the criteria progress and justness, as well as all these fairness criteria. Besides strong and weak fairness we also consider a form of fairness even weaker than weak fairness, requiring a task to be be enabled in each state on a path, as well as "during each transition". We are not aware of any completeness criteria occurring in the literature that is not progress, justness or one of these forms of fairness—or the weakest possible completeness criterion, declaring all paths complete, thereby ensures almost no liveness properties.

In [18] we argue that fairness assumptions are by default unwarranted. In real-world situations akin to Example 3 there is in fact no guarantee that Bart will ever get a beer. This is in contrast to justness, which by default is warranted. One could argue that a formalisation of justness is not necessary to arrive at a model of concurrency in which Cataline will eat her croissant, as fairness is an alternative to justness that accomplishes the same goal. But here we reject that argument on grounds that fairness tends to rule out as incomplete more paths than necessary. As argued in [13], this can lead to false guarantees about the satisfaction of certain liveness properties, e.g. Bart getting a beer in Example 3.

Reading guide In Section 2, following [18], we present transition systems with a concurrency relation satisfying some closure property, and define justness as a predicate on paths in any such transition system. Again following [18], we also propose an optional characterisation of the concurrency relation in terms of more primitive notions of the *necessary* and *affected components* of a transition.

To properly capture reactive system, we work with *labelled* transition systems, where each transition is labelled with the *action* that occurs when taking this transition. The labelling is typically used to describe how the transition synchronises with, and thus is dependant on, the environment. Whether a transition is blocking is then completely determined by its label. Hence we work with sets B of blocking actions and regard a transition as blocking iff it is labelled by an action in B.

In Section 3 we show liberal conditions under which B-justness meets the requirement of feasibility.

In Section 4 we recall the unifying definition of fairness from [18], and show how progress can be cast as particular fairness property. In spite of this we continue to see progress as a completeness criterion essentially different from fairness. We cannot cast justness as a fairness property.

Section 5 recalls the syntax and semantics of CCS, and its extensions ABC and CCSS with broadcast communication and signals, respectively. It also proposes a simplification of the operational semantics of CCSS by encoding signals as transitions, and recalls an alternative presentation of ABC that avoids negative premises in the operational semantics.

In Section 6 we associate transition systems with a concurrency relation to each of the five process algebras from Section 5, and show that they satisfies the closure property of Section 2. The concurrency relation is defined in terms of *synchrons*, novel particles out of which transitions are seen to be composed. Sections 2 and 6 together constitute a definition of justness valid for the five process algebras of Section 5. For CCS the concurrency relation is symmetric, but for the other four process algebras it is not. The alternative presentations of CCSS and ABC feature *signal transitions* that do not model state changes of the represented system; these need to be excepted from the justness requirement.

Section 7 revisits the component-based characterisation of the concurrency relation contemplated in Section 2, and proposes two alternative concepts of system components associated to a transition, with for each a classification of components as necessary and/or affected. The *dynamic components* give rise to the exact same concurrency relation as defined in terms of synchrons in Section 6, whereas the *static components* yield an underapproximation—a strictly smaller concurrency relation. However, only the static components satisfy a closure property proposed in [18].

Section 8 provides two computational interpretations of CCS and its extensions, the default one corresponding to the concurrency relation of Section 6, and thus the dynamic concurrency relation of Section 7; the other corresponding to the static concurrency relation of Section 7. We also provide a natural sublanguage on which the two concurrency relations coincide.

Section 9 shows that the dynamic and static concurrency relations give rise to the very same concept of justness. Hence, for the study of justness we may use whichever of these concurrency relations is the most convenient. Using this, in Section 10 we apply the results of Section 3 to show that B-justness is feasible for full CCS [23] and its extensions with broadcast communication and/or signals.

Section 11 shows that the concurrency relation of Section 6 agrees with the one defined earlier in [17] on pairs of transitions for which both are defined. Yet, the concurrency relation from [17] was defined only for transitions with the same source, and hence is not suitable for the formalisation of justness

In Sections 12 and 13 we establish that the concept of justness based on a concurrency relation between transitions, as proposed in [18] and applied to CCS and its extension in the present paper, coincides with the original coinductively defined concepts of justness from [17] and [7].

Section 14 summarises, and reviews related and future work.

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

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 2 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 of transition labels $\mathscr{L}$.

Here we work with LTSs labelled over a structured set of labels $(\mathscr{L}, Act, Rec)$, where $Rec \subseteq Act \subseteq \mathscr{L}$.

In [7] and in Sections 5.3–5.5 one encounters LTSs $\mathbb{T}$ enriched with *signals*. While these are naturally modelled as unary predicates on the states of $\mathbb{T}$, it is technically possible to model them as ordinary transitions t, satisfying $source(t) = target(t)$ [3]. This is formalised by declaring a set of *actions* $Act \subseteq \mathscr{L}$. Transitions t model the occurrence of an action $\ell(t)$ if $\ell(t) \in Act$, or the emission of the signal $\ell(t)$ otherwise. Signal transitions are largely ignored in the definitions below.

$Rec \subseteq Act$ is the set of *receptive* actions. Sets $B \subseteq Act$ of blocking actions must always contain Rec. In CCS and most other process algebras $Rec = \emptyset$ and $Act = \mathscr{L}$. Let $Tr^\bullet = \{t \in Tr \mid \ell(t) \in Act \setminus Rec\}$ be the set of transitions that are neither signals nor receptive.

Definition 3 A *path* in a transition system $(S, Tr, source, target)$ 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 $source(t_i) = s_{i-1}$ and $target(t_i) = s_i$ for all relevant i.

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

Definition 4 Let $B \subseteq Act$ be a set of actions with $Rec \subseteq B$—the *blocking* ones. Then $Tr^\bullet_{\neg B} := \{t \in Tr^\bullet \mid \ell(t) \notin B\}$ 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 $t \in Tr^\bullet_{\neg B}$.

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

Definition 5 A *labelled transition system with concurrency* (LTSC) is a tuple $(S, Tr, source, target, \ell, \frown)$ consisting of a LTS $(S, Tr, source, target, \ell)$ and a *concurrency relation* $\frown \subseteq Tr^\bullet \times Tr$, such that:

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

$$\begin{array}{l}\text{if } t \in Tr^\bullet \text{ and } \pi \text{ is a path from } source(t) \text{ to } s \in S \text{ such that } t \frown v \text{ for all transitions } v \\ \text{occurring in } \pi, \text{ then there is a } u \in Tr^\bullet \text{ such that } source(u) = s,\ \ell(u) = \ell(t) \text{ and } t \not\frown u.\end{array} \tag{2}$$

Informally, $t \frown v$ 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 $\frown$ 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 6 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 $s := source(t) \in \pi$, a transition u occurs in π past the occurrence of s, such that $t \not\frown u$.

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.

In reasonable extensions of $\frown$ to $Tr \times Tr$, signal transitions t would satisfy $t \frown t$, meaning that execution of t in no way affects any resources needed to execute t again. It therefore makes no sense to impose closure property (2), or the justness requirement, on signal transitions (see Example 9).

Components Instead of introducing $\smile\!\!\bullet$ as a primitive, it is possible to obtain it as a notion derived from two functions $npc, afc : Tr \to \mathscr{P}(\mathscr{C})$, 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

$$t \smile\!\!\bullet\, u \;\Leftrightarrow\; npc(t) \cap afc(u) = \emptyset$$

saying that u interferes with t if and only if a necessary participant in t is affected by u.

Most material in this section stems from [18]. 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, closure property (2) is new here; it is the weakest closure property that supports Theorem 1 and Proposition 1 below. In [18] only the model in which $\smile\!\!\bullet$ is derived from functions *npc* and *afc* comes with a closure property:

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

Trivially (3) implies (2).

3 Feasibility

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 [22] 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 [18] 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.

Proof: We present an algorithm for extending any given finite path π_0 into a B-just path π. We build an $\mathbb{N} \times \mathbb{N}$-matrix with a column for the—to be constructed—prefixes π_i of π, for $i \geq 0$. The columns π_i will list the transitions from $Tr^\bullet_{\neg B}$ enabled in the last state of π_i, leaving empty most slots if there are only finitely many. An entry in the matrix is either (still) empty, filled in with a transition, or crossed out. Let $f : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ be an enumeration of the entries in this matrix.

At the beginning only π_0 is known, and all columns of the matrix are empty. At each step $i \geq 0$ we fill in column i, extend the path π_i into π_{i+1} if possible by appending one transition (and its target state), and cross out some transitions occurring in the matrix. As an invariant, we maintain that a transition t occurring in the k-th column is already crossed out when reaching step $i > k$ iff a transition u occurs in the extension of π_k into π_i such that $t \not\smile\!\!\bullet\, u$. At each step $i \geq 0$ we proceed as follows:

Since π_i is known, we fill in column i by listing all transitions from $Tr^\bullet_{\neg B}$ enabled in the last state of π_i. We take n to be the smallest value such that entry $f(n) \in \mathbb{N} \times \mathbb{N}$ is already filled in, say with $t \in Tr^\bullet_{\neg B}$, but not yet crossed out. If such an n does not exist, the algorithm terminates, with output π_i. Let k be the column in which $f(n)$ appears. By our invariant, all transitions v occurring in the extension of π_k into π_i satisfy $t \smile\!\!\bullet\, v$. By (2) there is a transition $u \in Tr^\bullet_{\neg B}$ enabled in the last state of π_i such that $t \not\smile\!\!\bullet\, u$. We now

extend π_i into π_{i+1} by appending u to it, while crossing out all entries t' in the matrix for which $t' \not\leadsto u$, including entry $f(j)$. This maintains our invariant.

Obviously, π_i is a prefix of π_{i+1}, for $i \geq 0$. The desired path π is the limit of all the π_i. It is B-just, using the invariant, because each transition $t \in Tr^{\bullet}_{\neg B}$ that is enabled in a state of π will appear in the matrix, which acts like a priority queue, and be eventually crossed out. □

It is possible to strengthen Theorem 1 somewhat by calling two transitions t and t' *equivalent* if $t \leadsto u \Leftrightarrow t' \leadsto u$ for all $u \in Tr^{\bullet}$. An equivalence class of transitions is *enabled* iff one of its elements is.

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

Proof: The proof is the same as the one above, except that the matrix now contains equivalence classes of enabled transitions. □

4 Fairness

Let $Tr^{\circ} = \{t \in Tr \mid \ell(t) \in Act\}$. To formalise fairness we use LTSs $(S, Tr, source, target, \ell, \mathscr{T})$ that are augmented with a set $\mathscr{T} \subseteq \mathscr{P}(Tr^{\circ})$ of *tasks* $T \subseteq Tr^{\circ}$, each being a set of transitions. The concept of J-*fairness* from [18] is defined only for LTSCs $(S, Tr, source, target, \leadsto, \ell, \mathscr{T})$ augmented with such a $\mathscr{T}$.

Definition 7 ([18]) For an augmented LTS $\mathbb{T} = (S, Tr, source, target, \ell, [\leadsto,] \mathscr{T})$ and a set $Rec \subseteq B \subseteq Act$ of blocking actions, a task $T \in \mathscr{T}$ is *B-enabled* in a state $s \in S$ if there exists a non-blocking transition $t \in T$ with $\ell(t) \notin B$ and $source(t) = s$. [It is B-enabled *during the execution* of a transition $u \in Tr$ if there exists a $t \in T$ with $\ell(t) \notin B$, $source(t) = source(u)$ and $t \leadsto u$.] The task is said to be *perpetually B-enabled* on a path π in $\mathbb{T}$, if it is B-enabled in every state of π. [It is said to be *continuously B-enabled* on π, if it is B-enabled in every state and during every transition of π.] It is *relentlessly B-enabled* on π, if each suffix of π contains a state in which it is B-enabled.[3] It *occurs* in π if π contains a transition $t \in T$.

A path π in $\mathbb{T}$ is *weakly B-fair* if, for every suffix π' of π, each task that is perpetually B-enabled on π', occurs in π'. [A path π in $\mathbb{T}$ is J-*B-fair* if, for every suffix π' of π, each task that is continuously B-enabled on π', occurs in π'.] A path π in $\mathbb{T}$ is *strongly B-fair* if, for every suffix π' of π, each task that is relentlessly enabled on π', occurs in π'.

When the set B is defined once and for all or clear from context, we may omit the parameter B. This was the situation in [18].

In [18] many notions of fairness occurring in the literature were casts as instances of this definition. For each of them the set of tasks $\mathscr{T}$ was derived, in different ways, from some other structure present in the model of distributed systems from the literature. In fact, [18] considers 7 ways to construct the collection $\mathscr{T}$, and speaks of fairness of *actions*, *transitions*, *instructions*, *synchronisations*, *components*, *groups of components* and *events*. This yields 21 notions of fairness. To compare them, each is defined formally on a fragment of CCS, and the 21 fairness notions (together with progress, justness, and a few concepts of fairness found in the literature that are not instances of Definition 7) are ordered by strength by placing them in a lattice.

Progress can be casts as a fairness notion in the sense of Definition 7 by taking $\mathscr{T}$ to be the collection of only one task, namely Tr°. Clearly weak, strong and J-fairness all coincide for this $\mathscr{T}$. Likewise, the

[3] This is the case if the task is B-enabled in infinitely many states of π, in a state that occurs infinitely often in π, or in the last state of a finite π.

Table 1: Structural operational semantics of CCS

$$\alpha.P \xrightarrow{\alpha} P \text{ (ACT)} \qquad \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \text{ (SUM-L)} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'} \text{ (SUM-R)}$$

$$\frac{P \xrightarrow{\eta} P'}{P|Q \xrightarrow{\eta} P'|Q} \text{ (PAR-L)} \qquad \frac{P \xrightarrow{c} P',\ Q \xrightarrow{\bar{c}} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \text{ (COMM)} \qquad \frac{Q \xrightarrow{\eta} Q'}{P|Q \xrightarrow{\eta} P|Q'} \text{ (PAR-R)}$$

$$\frac{P \xrightarrow{\ell} P'}{P\backslash L \xrightarrow{\ell} P'\backslash L} \ (\ell, \bar{\ell} \notin L) \text{ (RES)} \qquad \frac{P \xrightarrow{\ell} P'}{P[f] \xrightarrow{f(\ell)} P'[f]} \text{ (REL)} \qquad \frac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'} \ (A \stackrel{def}{=} P) \text{ (REC)}$$

trivial completeness criterion, declaring all paths complete, coincides with weak, strong and J-fairness when taking $\mathscr{T} = \emptyset$. Nevertheless, it would be confusing to address these completeness criteria as fairness assumptions.

We do not see how justness can be cast a fairness notion in the sense of Definition 7. However, we now show that there exists a form of fairness according to Definition 7 that is at least as strong as justness. Namely take $\mathscr{T} := \{T_t \mid t \in Tr^\bullet\}$ where $T_t := \{u \in Tr^\circ \mid t \not\frown\!\!\bullet\ u\}$.

Proposition 1 Given this $\mathscr{T}$ and $B \subseteq Act$, any path that is strongly or weakly B-fair is certainly B-just.

Proof: Any path that is strongly B-fair is certainly weakly B-fair. This follows trivially from the definitions, for any choice of $\mathscr{T}$. (Likewise, any path that is weakly B-fair is certainly J-B-fair.)

Suppose π is weakly B-fair. We show it is B-just. Suppose that $t \in Tr^\bullet_{\neg B}$ is enabled in a state s of π, i.e., $source(t) = s \in \pi$, but all transitions v in π past the occurrence of s satisfy $t \frown\!\!\bullet\ v$. Let π' be the suffix of π starting in s. Closure property (2) guarantees that for every state s' of π' there is a $u \in Tr^\bullet$ such that $source(u) = s'$, $\ell(u) = \ell(t)$ and $t \not\frown\!\!\bullet\ u$. Hence task T_t is perpetually B-enabled on π'. By weak B-fairness T_t must occur in π', meaning that π' contains a transition $u \in Tr^\circ$ with $t \not\frown\!\!\bullet\ u$. This contradicts the assumptions. □

5 CCS and its extensions with broadcast communication and signals

This section presents five process algebras: Milner's *Calculus of Communicating Systems* (CCS) [24], its extensions with broadcast communication ABC [17] and signals CCSS [7], an alternative presentation of CCSS where signals are encoded as transitions, and an alternative presentation of ABC that avoids negative premises in favour of *discard* transitions.

5.1 CCS

CCS [24] 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 $A \stackrel{def}{=} P$ 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 *(handshake communication) co-names*. Complementation is extended to $\bar{\mathscr{C}}_h$ by setting $\bar{\bar{c}} = c$. $Act := \mathscr{C}_h \dot\cup \bar{\mathscr{C}}_h \dot\cup \{\tau\}$ 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 $f : \mathscr{C}_h \to \mathscr{C}_h$; it extends to Act by $f(\bar{c}) = \overline{f(c)}$ and $f(\tau) := \tau$. The set $\mathbb{P}_{\text{CCS}}$ of CCS expressions or *processes* is the smallest set including:

$\mathbf{0}$		*inaction*
$\alpha.P$	for $\alpha \in Act$ and $P \in \mathbb{P}_{\mathrm{CCS}}$	*action prefixing*
$P+Q$	for $P,Q \in \mathbb{P}_{\mathrm{CCS}}$	*choice*
$P\|Q$	for $P,Q \in \mathbb{P}_{\mathrm{CCS}}$	*parallel composition*
$P\backslash L$	for $L \subseteq \mathscr{C}_h$ and $P \in \mathbb{P}_{\mathrm{CCS}}$	*restriction*
$P[f]$	for f a relabelling and $P \in \mathbb{P}_{\mathrm{CCS}}$	*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 $\rightarrow \subseteq \mathbb{P}_{\mathrm{CCS}} \times Act \times \mathbb{P}_{\mathrm{CCS}}$, where transitions $P \xrightarrow{\ell} Q$ are derived from the rules of Table 1. The process $\alpha.P$ performs the action α first and subsequently acts as P. The choice operator $P+Q$ may act as either P or Q, depending on which of the processes is able to act at all. The parallel composition $P|Q$ executes an action from P, an action from Q, or in the case where P and Q can perform complementary actions c and $\bar{c}$, the process can perform a synchronisation, resulting in an internal action τ. The restriction operator $P\backslash L$ inhibits execution of the actions from L and their complements. The relabelling $P[f]$ acts like process P with all labels ℓ replaced by $f(\ell)$. Finally, the rule for agent identifiers says that an agent A has the same transitions as the body P of its defining equation.

5.2 ABC—The Algebra of Broadcast Communication

The Algebra of Broadcast Communication (ABC) [17] is parametrised with sets $\mathscr{A}$ of *agent identifiers*, $\mathscr{B}$ of *broadcast names* and $\mathscr{C}_h$ of *handshake communication 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 $\sharp \in \{!,?\}$. $Act := \mathscr{B}! \,\dot\cup\, \mathscr{B}? \,\dot\cup\, \mathscr{C}_h \,\dot\cup\, \bar{\mathscr{C}}_h \,\dot\cup\, \{\tau\}$ 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 $f\colon (\mathscr{B} \to \mathscr{B}) \cup (\mathscr{C}_h \to \mathscr{C}_h)$. It extends to Act by $f(\bar{c}) = \overline{f(c)}$, $f(b\sharp) = f(b)\sharp$ and $f(\tau) := \tau$. The set $\mathbb{P}_{\mathrm{ABC}}$ of ABC expressions is defined exactly as $\mathbb{P}_{\mathrm{CCS}}$. 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) [31]. 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.

Table 2: Structural operational semantics of ABC broadcast communication

$$\frac{P \xrightarrow{b\sharp_1} P',\ Q \xrightarrow{b?}\!\!\!\!\!\!\rightarrow}{P|Q \xrightarrow{b\sharp_1} P'|Q}\ \text{(Bro-L)} \qquad \frac{P \xrightarrow{b\sharp_1} P',\ Q \xrightarrow{b\sharp_2} Q'}{P|Q \xrightarrow{b\sharp} P'|Q'}\ \sharp_1\circ\sharp_2{=}\sharp{\neq}\_ \text{ with } \begin{array}{c|ccc} \circ & ! & ? \\ \hline ! & \_ & ! \\ ? & ! & ? \end{array}\ \text{(Bro-C)} \qquad \frac{P \xrightarrow{b?}\!\!\!\!\!\!\rightarrow,\ Q \xrightarrow{b\sharp_2} Q'}{P|Q \xrightarrow{b\sharp_2} P|Q'}\ \text{(Bro-R)}$$

Table 3: Structural operational semantics for signals of CCSS

$$(P\hat{}s)^{\curvearrowright s} \qquad \frac{P \xrightarrow{\alpha} P'}{P\hat{}r \xrightarrow{\alpha} P'} \qquad \frac{P^{\curvearrowright s}}{(P+Q)^{\curvearrowright s}} \qquad \frac{Q^{\curvearrowright s}}{(P+Q)^{\curvearrowright s}}$$

$$\frac{P^{\curvearrowright s}}{(P|Q)^{\curvearrowright s}} \qquad \frac{P^{\curvearrowright s},\ Q \xrightarrow{s} Q'}{P|Q \xrightarrow{\tau} P|Q'} \qquad \frac{P \xrightarrow{s} P',\ Q^{\curvearrowright s}}{P|Q \xrightarrow{\tau} P'|Q} \qquad \frac{Q^{\curvearrowright s}}{(P|Q)^{\curvearrowright s}}$$

$$\frac{P^{\curvearrowright s}}{(P\hat{}r)^{\curvearrowright s}} \qquad \frac{P^{\curvearrowright s}}{(P\backslash L)^{\curvearrowright s}}\ (s \notin L) \qquad \frac{P^{\curvearrowright s}}{P[f]^{\curvearrowright f(s)}} \qquad \frac{P^{\curvearrowright s}}{A^{\curvearrowright s}}\ (A \stackrel{def}{=} P)$$

5.3 CCS with signals

CCS with signals (CCSS) [7] 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 $Act := \mathscr{S} \dot\cup \bar{\mathscr{C}}_h \dot\cup \bar{\mathscr{C}}_h \dot\cup \{\tau\}$. A relabelling is a function $f : (\mathscr{S} \to \mathscr{S}) \cup (\mathscr{C}_h \to \mathscr{C}_h)$. As before it extends to Act by $f(\bar{c}) = \overline{f(c)}$ and $f(\tau) := \tau$. The set $\mathbb{P}_{\mathrm{CCSS}}$ of CCSS expressions is defined just as $\mathbb{P}_{\mathrm{CCS}}$, but now also $P\hat{}s$ is a process for $P \in \mathbb{P}_{\mathrm{CCSS}}$ and $s \in \mathscr{S}$, and restriction also covers signals.

The semantics of CCSS is given by the labelled transition relation $\to \subseteq \mathbb{P}_{\mathrm{CCSS}} \times Act \times \mathbb{P}_{\mathrm{CCSS}}$ and a predicate $^{\curvearrowright} \subseteq \mathbb{P}_{\mathrm{CCSS}} \times \mathscr{S}$ that are derived from the rules of CCS (Table 1, where η, α, ℓ range over Act and $L \subseteq \mathscr{C}_h \cup \mathscr{S}$), and the rules of Table 3. The predicate $P^{\curvearrowright s}$ indicates that process P emits the signal s, whereas a transition $P \xrightarrow{s} P'$ indicates that P reads the signal s and thereby turns into P'. The first rule is the base case showing that a process $P\hat{}s$ emits the signal s. The second rule of Table 3 models the fact that signalling cannot prevent a process from making progress. After having taken an action, the signalling process loses its ability to emit the signal. The two rules in the middle of Table 3 state that the action of reading a signal by one component in (parallel) composition together with the emission of the same signal by another component, results in an internal transition τ; similar to the case of handshake communication. Note that the component emitting the signal does not change through this interaction. All the other rules of Table 3 lift the emission of s by a subprocess P to the overall process.

5.4 Encoding signals as transitions

A more compact presentation of CCSS can be obtained by encoding a signal $P^{\curvearrowright s}$ as a transition $P \xrightarrow{\bar{s}} P$; this is done in [3]. The price to be paid for the resulting simplification of the operational semantics is that the new transitions $P \xrightarrow{\bar{s}} P$ should not be counted in the definition of justness, since they do not model changes in the state of the represented system.

In this presentation of CCSS the set of labels is defined as $\mathscr{L} := Act \dot\cup \bar{\mathscr{S}}$, where Act is as in the previous section and $\bar{\mathscr{S}} := \{\bar{s} \mid s \in \mathscr{S}\}$. Complementation is extended to $\bar{\mathscr{C}}_h \cup \bar{\mathscr{S}}$ by setting $\bar{\bar{c}} = c$, with $c \in \mathscr{C}_h \cup \mathscr{S}$. 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 semantics is given by the labelled transition relation $\to \subseteq \mathbb{P}_{\mathrm{CCSS}} \times \mathscr{L} \times \mathbb{P}_{\mathrm{CCSS}}$ 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 4.

Table 4: Structural operational semantics of CCSS when signals are encoded as transitions

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

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

5.5 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.[4] To this end, let $\mathscr{B}{:} := \{b{:} \mid b \in \mathscr{B}\}$ be the set of *broadcast discards*, and $\mathscr{L} := \mathscr{B}{:} \dot{\cup} Act$ the set of *transition labels*, with Act as in Section 5.2. 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 5.

Lemma 1 [31] $P \xrightarrow{b:} Q$ iff $Q = P \wedge P \xrightarrow{b?}\!\!\!\!\!\!/\ $, for $P, Q \in \mathbb{P}_{\mathrm{ABC}}$ and $b \in \mathscr{B}$.

Proof: A straightforward induction on derivability of transitions. □

Corollary 2 The structural operational semantics of ABC from Sections 5.2 and 5.5 yield the same labelled transition relation $\longrightarrow$ when transitions labelled $b{:}$ are ignored. □

This approach stems from the Calculus of Broadcasting Systems (CBS) [31].

6 An LTS with concurrency for CCS and its extensions

The forthcoming material applies to each of the process algebras from Section 5, or combinations thereof. Let $\mathbb{P}$ be the set of processes or expressions in the appropriate language.

We allocate an LTS as in Definition 2 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 $source(t) = P$, $target(t) = Q$ and $\ell(t) = \ell$. Here a *derivation* of a formula φ (either a transition $P \xrightarrow{\ell} Q$ or a predicate $P^{\frown s}$) is a well-founded tree with the nodes labelled by formulas, such that the root has label φ, 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 1–5.

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

Table 5: Structural operational semantics of ABC broadcast communication with discard transitions

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

$$\frac{P \xrightarrow{b\sharp_1} P',\ Q \xrightarrow{b\sharp_2} Q'}{P|Q \xrightarrow{b\sharp} P'|Q'}\ \sharp_1 \circ \sharp_2 = \sharp \neq \_ \text{ with}$$

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

$$\frac{P \xrightarrow{b:} P'}{A \xrightarrow{b:} A}\ (A \stackrel{def}{=} P)$$

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 [17], 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 $\xrightarrow{\alpha}P$. 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 the syntactic structure of derivations: $t|P \neq P|t$ and $(t|u)|v \neq t|(u|v)$.

For CCSS as in Section 5.3 there are also derivations $\xi \notin Tr$ of signals. The unique derivation of the signal $(P\hat{}s)^{\frown s}$ using the first rule of Table 3 is called $P^{\rightarrow s}$. The other rules of Table 3 yield derivations $t\hat{}r$, $\xi+Q$, $P+\xi$, $\xi|Q$, $\xi|t$, $t|\xi$, $P|\xi$, $\xi\hat{}r$, $\xi\backslash L$, $\xi[f]$ and $A{:}\xi$, where ξ is the derivation of the signal premise, and t of the transition premise of the rule. The derivations of Section 5.4 are (named) the same as in Section 5.3, but now they are all derivations of transitions $t \in Tr$; in particular $P^{\rightarrow s}$ is now the unique derivation of the transition $P\hat{}s \xrightarrow{\bar{s}} P\hat{}s$ using the first rule of Table 4.

The derivations obtained by application of the rules of Table 5 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 of these rules.

Synchrons Let $Arg := \{+_{\mathrm{L}}, +_{\mathrm{R}}, |_{\mathrm{L}}, |_{\mathrm{R}}, \backslash L, [f], A{:}, \hat{}r \mid L \subseteq \mathscr{C}_h \wedge f \text{ a relabelling} \wedge A \in \mathscr{A} \wedge r \in \mathscr{S}\}$. A *synchron* is an expression $\sigma(\xrightarrow{\alpha}P)$ 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 $\varsigma(P)$ of a CCS, ABC or CCSS process P is inductively defined by

$$\begin{array}{lll}
\varsigma(\mathbf{0}) = \emptyset & & \varsigma(\alpha.P) = \{(\xrightarrow{\alpha}P)\} \\
\varsigma(P+Q) = +_L\varsigma(P) \cup +_R\varsigma(Q) & & \varsigma(P|Q) = |_L\varsigma(P) \cup |_R\varsigma(Q) \\
\varsigma(P\backslash L) = \backslash L\varsigma(P) & & \varsigma(P[f]) = [f]\varsigma(P) \\
\varsigma(A) = A{:}\varsigma(P) \text{ when } A \stackrel{def}{=} P. & & \varsigma(P\hat{}s) = \{(P^{\rightarrow s})\} \cup \hat{}s\,\varsigma(P)
\end{array}$$

Thus, a synchron of a process Q can be seen as a path in the parse tree of Q to an unguarded subexpression $\alpha.P$ or $P\hat{}s$ of Q—except that recursion $A \stackrel{def}{=} P$ gets unfolded in the construction of such a path. Here a subexpression of Q occurs *unguarded* if it does not lay within a subexpression $\beta.R$ of Q.

For ABCd we amend the clauses for inaction and prefixing:

$$\varsigma(\mathbf{0}) = \{(b{:}) \mid b \in B\} \qquad \varsigma(\alpha.P) = \{(b{:}) \mid b \in B \wedge b? \neq \alpha\} \cup \{(\xrightarrow{\alpha}P)\}$$

The set of synchrons $\varsigma(t)$ of a derivation t of a transition $P \xrightarrow{\ell} Q$ or signal $P^{\frown s}$ is defined by

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

Thus, a synchron of t represents a path in the proof-tree t from its root to a leaf. Note that we use the symbol ς as a variable ranging over synchrons, and as the name of two functions—context disambiguates.

Lemma 2 If t is a derivation of $P \xrightarrow{\ell} Q$ or $P^{\frown s}$ then $\varsigma(t) \subseteq \varsigma(P)$.

Proof: A trivial structural induction on t. □

Each transition derivation can be seen as the synchronisation of one or more synchrons.

Example 4 The CCS process $P = \big((c.Q + (d.R|e.S))|\bar{c}.T\big)\setminus c$ has 3 outgoing transitions: $P \xrightarrow{\tau} (Q|T)\setminus c$, $P \xrightarrow{d} ((R|e.S)|\bar{c}.T)\setminus c$ and $P \xrightarrow{e} ((d.R|S)|\bar{c}.T)\setminus c$. Let t_τ, t_d and $t_e \in Tr$ be the unique derivations of these transitions. Then t_τ is a synchronisation of two synchrons, whereas t_d and $t_e \in Tr$ have only one each: $\varsigma(t_\tau) = \{\setminus c\,|_L + _L(\xrightarrow{c} Q), \setminus c\,|_R(\xrightarrow{\bar{c}} T)\}$, $\varsigma(t_d) = \{\setminus c\,|_L + _R\,|_L(\xrightarrow{d} R)\}$ and $\varsigma(t_e) = \{\setminus c\,|_L + _R\,|_R(\xrightarrow{e} S)\}$. The derivations t_d and $t_e \in Tr$ 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 $((d.R|S)|\bar{c}.T)\setminus c \xrightarrow{d} ((R|S)|\bar{c}.T)\setminus c$. 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: $\varsigma(t'_d) = \{\setminus c\,|_L\,|_L(\xrightarrow{d} R)\}$. 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 $\rightsquigarrow \subseteq Tr^\bullet \times Tr^\bullet$ and $\smile\!\!\bullet \subseteq Tr^\bullet \times Tr$ such that the following properties hold:

The relation $\rightsquigarrow$ is reflexive and transitive. (4)
If $t \rightsquigarrow t'$ and $t \smile\!\!\bullet\, v$, then $t' \smile\!\!\bullet\, v$. (5)
If $t \smile\!\!\bullet\, v$ with $source(t) = source(v)$ then $\exists t' \in Tr^\bullet$ with $source(t') = target(v)$ and $t \rightsquigarrow t'$. (6)
If $t \rightsquigarrow t'$ then $\ell(t') = \ell(t)$ and $t \not\frown\!\!\bullet\, t'$. (7)

With $t \smile\!\!\bullet\, v$ 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 $Tr^\bullet = Tr$), for ABC and CCSS it is not:

Example 5 ([17]) 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 $t \smile\!\!\bullet\, v$ but $v \not\frown\!\!\bullet\, t$.

Example 6 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 $t \smile\!\!\bullet\, v$ but $v \not\frown\!\!\bullet\, t$.

Proposition 2 Assume (4)–(6). If $t \in Tr^\bullet$ and π is a path from $source(t)$ to $P \in \mathbb{P}$ such that $t \smile\!\!\bullet\, v$ for all transitions v occurring in π, then there is a $t' \in Tr^\bullet$ such that $source(t') = P$ and $t \rightsquigarrow t'$.

Proof: By induction on the length of π.

The induction base is trivial, taking $t' := t$, and applying the reflexivity of $\rightsquigarrow$.

So assume $t \in Tr^\bullet$ and π is a path from $source(t)$ with as last transition v' with $source(v') = P$ and $target(v') = Q$, such that $t \smile\!\!\bullet\, v$ for all transitions v occurring in π. By induction, there is a $t' \in Tr^\bullet$ such that $source(t') = P$ and $t \rightsquigarrow t'$. By (5) $t' \smile\!\!\bullet\, v'$. By (6) there is a $t'' \in Tr^\bullet$ such that $source(t'') = Q$ and $t' \rightsquigarrow t''$. Now apply the transitivity of $\rightsquigarrow$. □

Corollary 3 Assume (4)–(7). If $t \in Tr^\bullet$ and π is a path from $source(t)$ to $P \in \mathbb{P}$ such that $t \smile\!\!\bullet\, v$ for all transitions v occurring in π, then there is a $t' \in Tr^\bullet$ such that $source(t') = P$, $\ell(t') = \ell(t)$ and $t \not\frown\!\!\bullet\, t'$.

Proof: Immediately from Proposition 2 and (7). □

It follows that the LTS $(\mathbb{P}, Tr, source, target, \ell)$, augmented with the concurrency relation $\frown\!\!\!\bullet$, is an LTSC in the sense of Definition 5. By (4) and (7) $\frown\!\!\!\bullet$ is irreflexive on $Tr^\bullet$, so property (1) holds. That (2) holds is stated by Corollary 3.

We now proceed to define the relations $\rightsquigarrow$ and $\frown\!\!\!\bullet$ 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* [24]; the others are *static*. (Static operators stay around when their arguments perform transitions.) For $\sigma \in Arg^*$ let $static(\sigma)$ be the result of removing all dynamic elements from σ. Moreover, for $\varsigma = \sigma\upsilon$ with $\upsilon \in \{(\stackrel{\alpha}{\rightarrow}P), (P^{\rightarrow s}), (b{:})\}$ let $static(\varsigma) := static(\sigma)\upsilon$.

Definition 8 A synchron ς' is a *possible successor* of a synchron ς, notation $\varsigma \rightsquigarrow \varsigma'$, if either $\varsigma' = \varsigma$, or ς has the form $\sigma_1|_D\varsigma_2$ for some $\sigma_1 \in Arg^*$, $D \in \{L,R\}$ and ς_2 a synchron, and $\varsigma' = static(\sigma_1)|_D\varsigma_2$.

Definition 9 Two synchrons ς and υ are *directly concurrent*, notation $\varsigma \frown_d \upsilon$, 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 $\varsigma' \frown \upsilon'$, if $\exists \varsigma, \upsilon.\, \varsigma' \leftsquigarrow \varsigma \frown_d \upsilon \rightsquigarrow \upsilon'$.

Lemma 3 If $\varsigma, \upsilon \in \varsigma(P)$ for some $P \in \mathbb{P}$ and $\varsigma \frown \upsilon$, then $\varsigma \frown_d \upsilon$.

Proof: By assumption, there are $\varsigma^\dagger, \upsilon^\dagger$ with $\varsigma \leftsquigarrow \varsigma^\dagger \frown_d \upsilon^\dagger \rightsquigarrow \upsilon$. W.l.o.g. we choose $\varsigma^\dagger$ and $\upsilon^\dagger$ such that either $\varsigma^\dagger = \varsigma$ or $\upsilon^\dagger = \upsilon$. The synchrons ς and υ describe paths in the parse tree of P, so the first symbol where they differ must be a right versus left argument of the same binary operator *op*. The possibility that $op = +$ quickly leads to a contradiction, so $\varsigma \frown_d \upsilon$. □

Necessary and active synchrons All synchrons of the form $\sigma(\stackrel{\alpha}{\rightarrow}P)$ are *active*; their execution causes a transition $\alpha.P \stackrel{\alpha}{\longrightarrow} P$ 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. It follows that Tr° (see Section 4) is the set of transitions $t \in Tr$ with $a\varsigma(t) \neq \emptyset$.

Whether a synchron $\varsigma \in \varsigma(t)$ is *necessary* for t to occur is defined only for $t \in Tr^\bullet$. 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 $t \in Tr^\bullet$ 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.

Lemma 4 If $t \in Tr^\bullet$ and $\varsigma, \upsilon \in n\varsigma(t) \cup a\varsigma(t)$ with $\varsigma \neq \upsilon$, then $\varsigma \frown_d \upsilon$.

Proof: A trivial structural induction on t. □

Lemma 5 If $\varsigma \rightsquigarrow \varsigma'$ and $\varsigma \rightsquigarrow \varsigma''$ then $\varsigma' \not\frown \varsigma''$. Also, if $\varsigma \rightsquigarrow \varsigma''$ and $\varsigma' \rightsquigarrow \varsigma''$ then $\varsigma \not\frown \varsigma'$.

Proof: Note that $\varsigma \rightsquigarrow \varsigma'$ implies $static(\varsigma) = static(\varsigma')$, and $\varsigma \frown_d \upsilon$ implies $static(\varsigma) \frown_d static(\upsilon)$. Hence $\varsigma \frown \upsilon$ implies $static(\varsigma) \frown_d static(\upsilon)$. Moreover, $\frown_d$ (and hence $\frown$) is irreflexive: $\varsigma \not\frown_d \varsigma$.

Suppose $\varsigma \rightsquigarrow \varsigma'$ and $\varsigma \rightsquigarrow \varsigma''$. Then $static(\varsigma') = static(\varsigma) = static(\varsigma'')$. So $static(\varsigma') \not\frown_d static(\varsigma'')$, and hence $\varsigma' \not\frown \varsigma''$. The other statement follows in the same way. □

Definition 10 A derivation $t' \in Tr^\bullet$ is a *possible successor* of a derivation $t \in Tr^\bullet$, notation $t \rightsquigarrow t'$, 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 $\forall \varsigma' \in n\varsigma(t').\, \exists \varsigma \in n\varsigma(t).\, \varsigma \rightsquigarrow \varsigma'$.

By Lemmas 4 and 5 this implies that the relation $\rightsquigarrow$ between the necessary synchrons of t and t' is a bijection.

Definition 11 Derivation $t \in Tr^\bullet$ 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 4 one has $t_d \smile t_e$, $t_d \rightsquigarrow t'_d$ and $t'_d \smile t_e$. Here $t \smile u$ denotes $t \smile\!\!\bullet\, u \wedge u \smile\!\!\bullet\, t$.

Proposition 3 The relation $\rightsquigarrow$ on $Tr^\bullet$ is reflexive, transitive, and disjoint with $\smile\!\!\bullet$.

Proof: The relation $\rightsquigarrow$ on synchrons is reflexive and transitive by definition. That it is disjoint with $\smile$ follows as in the proof of Lemma 5. The lifting of these properties to derivations follows directly from the definitions, using that $n\varsigma(u) \cap a\varsigma(u) \neq \emptyset$ for all $u \in Tr^\bullet$. □

Proposition 4 If $t \rightsquigarrow t'$ and $t \smile\!\!\bullet\, v$, then $t' \smile\!\!\bullet\, v$.

Proof: Let $\varsigma' \in n\varsigma(t')$ and $\upsilon \in a\varsigma(v)$. We have to show that $\varsigma' \smile \upsilon$. By assumption $\exists \varsigma \in n\varsigma(t). \varsigma \rightsquigarrow \varsigma'$. Furthermore, $\varsigma \smile \upsilon$, since $t \smile\!\!\bullet\, v$. So $\exists \varsigma^\dagger, \upsilon^\dagger. \varsigma \leftsquigarrow \varsigma^\dagger \smile_d \upsilon^\dagger \rightsquigarrow \upsilon$. By the transitivity of $\rightsquigarrow$ this entails $\varsigma' \leftsquigarrow \varsigma^\dagger \smile_d \upsilon^\dagger \rightsquigarrow \upsilon$, so $\varsigma' \smile \upsilon$. □

Proposition 5 If $t \rightsquigarrow t'$ then $\ell(t') = \ell(t)$.

Proof: If $t \rightsquigarrow t'$ then the derivation t' must be obtainable from t by reducing some subterms of the form $u + Q$, $P + u$, $A{:}u$ or $u\hat{}r$ to u, and/or to change some receptive or discarding partners in a broadcast communication. Given rules (SUM-L), (SUM-R), (REC), etc., this does not alter the label of this derivation. □

Propositions 3–5 establish the required properties (4,5,7). It remains to establish (6)—see Proposition 6.

Definition 12 A set $\Sigma \subseteq \varsigma(P)$ of synchrons of P is *P-consistent* if there is a derivation $t \in Tr^\bullet$ with $n\varsigma(t) = \Sigma$ and $source(t) = P$.

Lemma 6 Let $P, Q \in \mathbb{P}$ be processes, $\Sigma \subseteq \varsigma(P)$ and $\Sigma' \subseteq \varsigma(Q)$ such that Σ is P-consistent, $|\Sigma'| = |\Sigma|$ and $\forall \varsigma \in \Sigma. \exists \varsigma' \in \Sigma'. \varsigma \rightsquigarrow \varsigma'$. Then Σ' is Q-consistent.

Proof: Let $t \in Tr$ be such that $n\varsigma(t) = \Sigma$ and $source(t) = P$. So $\Sigma \neq \emptyset$. Each synchron $\varsigma \in \Sigma$ represents a path in the derivation t from its root to a leaf. If $\varsigma \rightsquigarrow \varsigma'$ than ς' represents a version if the same path, but in which certain nodes of t, labelled $+Q'$, $P'+$, $A{:}$ or $\hat{}r$, are marked as being deleted by ς'. Since $\Sigma' \subseteq \varsigma(Q)$, this marking of deleted nodes is consistent, in the the sense that no node of t is deleted according to one element of Σ', but kept according to another. In fact, whether a node of t is marked as deleted depends entirely on the syntactic shape of Q. In view of the rules (SUM-L), (SUM-R), (REC), etc., actually deleting the indicated nodes from the derivation t yields another derivation t', with $n\varsigma(t') = \Sigma'$ and $source(t') = Q$. Depending on the syntactic shapes of P and Q, some receptive or discarding partners in broadcast communications may have been altered between t and t' as well. □

Write $\varsigma \smile\!\!\bullet_d u$ for ς a synchron and $u \in Tr$ if $\varsigma \smile_d \upsilon$ for all $\upsilon \in a\varsigma(u)$.

Definition 13 Let ς, υ be synchrons with $\varsigma \smile_d \upsilon$, i.e., $\varsigma = \sigma|_D \varsigma'$ and $\upsilon = \sigma|_E \upsilon'$ for some $\sigma \in Arg^*$, synchrons ς', υ' and $\{D, E\} = \{L, R\}$. Define $\varsigma @ \upsilon$, where @ is pronounced "after", to be $static(\sigma)|_D \varsigma'$.

For $u \in Tr^\bullet$ with $\varsigma \smile\!\!\bullet_d u$ let $\varsigma @ u := \varsigma @ \upsilon$ for the $\upsilon \in a\varsigma(u)$ that is "closest" to ς, in the sense that it has the largest prefix in common with it. For $u \in Tr \setminus Tr^\bullet$ let $\varsigma @ u := \varsigma$.

In Example 4 $\backslash c\,|_L + _R\,|_L(\stackrel{d}{\rightarrow} R) @ t_e = \backslash c\,|_L\,|_L(\stackrel{d}{\rightarrow} R)$.

Observation 1 Let $u \in Tr$, $P = source(u)$ and $Q = target(u)$. If $\varsigma \in \varsigma(P)$ with $\varsigma \smile\!\!\bullet_d u$ then $\varsigma@u \in \varsigma(Q)$.

Proposition 6 If $t \in Tr^\bullet$, $v \in Tr$ and $t \smile\!\!\bullet v$ with $source(t) = source(v)$ then there is a derivation $t' \in Tr^\bullet$ with $source(t') = target(v)$ and $t \rightsquigarrow t'$.

Proof: Let $P := source(v)$ and $Q := target(v)$. Then $\Sigma := n\varsigma(t)$ is P-consistent. By Lemma 3 $\varsigma \smile\!\!\bullet_d v$ for all $\varsigma \in \Sigma$. Let $\Sigma' := \{\varsigma@v \mid \varsigma \in \Sigma\}$. By Observation 1 $\Sigma' \subseteq \varsigma(Q)$. By Definition 13 $\varsigma \rightsquigarrow \varsigma@v$ for all $\varsigma \in \Sigma$. By Lemma 5 $|\Sigma'| = |\Sigma|$. So by Lemma 6 Σ' is Q-consistent, that is, there exists a $t' \in Tr^\bullet$ with $n\varsigma(t') = \Sigma'$ and $source(t') = Q$. By Definition 10 and Lemma 5 $t \rightsquigarrow t'$. □

7 Components

This section proposes two concepts of system components associated to a transition, with for each a classification of components as necessary and/or affected. We then apply a definition of a concurrency relation in terms of these components closely mirroring Definition 11 in Section 6 of the concurrency relation $\smile\!\!\bullet$ in terms of synchrons. The *dynamic components* give rise to the exact same concurrency relation $\smile\!\!\bullet$ from Definition 11, whereas the *static components* yield a strictly smaller concurrency relation $\smile\!\!\bullet_s$. However, only the static components satisfy closure property (3). Finally, we present three alternative versions of $\smile\!\!\bullet_s$ that all give rise to the same concept of justness.

7.1 Dynamic components

A *(dynamic) component* is either the empty string ε or any string $\sigma\iota$ with $\sigma \in Arg^*$ and $\iota \in Arg$ a static argument. Each synchron ς can be uniquely written as $\gamma\varsigma'$ with γ a component and ς' a synchron with only dynamic arguments. The *dynamic component* $C(\varsigma)$ of such a synchron ς is defined to be γ.

The set of dynamic components $COMP(P)$ of a process P is defined as $\{C(\varsigma) \mid \varsigma \in \varsigma(P)\}$.

The set of dynamic components $COMP(t)$ of a derivation t is defined as $\{C(\varsigma) \mid \varsigma \in \varsigma(t)\}$.

The set of *necessary* dynamic components $NC(t)$ of a derivation t is defined as $\{C(\varsigma) \mid \varsigma \in n\varsigma(t)\}$.

The set of *affected* dynamic components $AC(t)$ of a derivation t is defined as $\{C(\varsigma) \mid \varsigma \in a\varsigma(t)\}$.

A component γ' is a *possible successor* of a component γ, notation $\gamma \rightsquigarrow \gamma'$, if either $\gamma' = \gamma$ or γ has the form $\sigma_1|_D\gamma_2$, with $\sigma_1 \in Arg^*$, $D \in \{L,R\}$ and γ_2 a component, and $\gamma' = static(\sigma_1)|_D\gamma_2$.

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

These definitions imply that $\varsigma \rightsquigarrow \upsilon \Rightarrow C(\varsigma) \rightsquigarrow C(\upsilon)$ and $\varsigma \smile \upsilon \Leftrightarrow C(\varsigma) \smile C(\upsilon)$.

The next lemma, whose proof is trivial, say that the concurrency relation $\smile\!\!\bullet$ on derivations could equally well have be defined in terms of dynamic components (rather than synchrons).

Lemma 7 Derivation $t \in Tr^\bullet$ is unaffected by u, $t \smile\!\!\bullet u$, iff $\forall \gamma \in NC(t). \forall \delta \in AC(u). \gamma \smile \delta$. □

The following shows that the functions NC and AC do not satisfy closure property (3) of Section 2.

Example 7 In Example 4 with $S := \mathbf{0}$, $t_d, t_e \in Tr^\bullet$ with $source(t_d) = source(t_e)$ and $NC(t_d) \cap AC(t_e) = \emptyset$. Yet, there is no $u \in Tr^\bullet$ with $source(u) = target(t_e)$, $\ell(u) = \ell(t_d) = d$ and $NC(u) = NC(t_d)$. In fact, the unique $u \in Tr^\bullet$ with $source(u) = target(t_e)$ and $\ell(u) = d$ is t'_d. However, $NC(t_d) = \{\backslash c\,|_L + _R\,|_L\}$, whereas $NC(t'_d) = \{\backslash c\,|_L\,|_L\}$.

7.2 Static components

A *static component* is a string $\sigma \in Arg^*$ 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.

The set of static components $comp(P)$ of a process P is defined as $\{c(\varsigma) \mid \varsigma \in \varsigma(P)\}$.

The set of static components $comp(t)$ of a derivation t is defined as $\{c(\varsigma) \mid \varsigma \in \varsigma(t)\}$.

The set of *necessary* static components $npc(t)$ of a derivation t is defined as $\{c(\varsigma) \mid \varsigma \in n\varsigma(t)\}$.

The set of *affected* static components $afc(t)$ of a derivation t is defined as $\{c(\varsigma) \mid \varsigma \in a\varsigma(t)\}$.

Since $n\varsigma(t) \subseteq \varsigma(t)$ and $a\varsigma(t) \subseteq \varsigma(t)$, we have $npc(t) \subseteq comp(t)$ and $afc(t) \subseteq comp(t)$. Moreover, by Lemma 2, $comp(t) \subseteq comp(source(t))$.

The following lemma shows how the relations $\rightsquigarrow$ and $\smile$ simplify when applied to static components.

Lemma 8 If $\gamma \in \mathscr{C}$ and $\gamma \rightsquigarrow \gamma'$ then $\gamma' = \gamma$. Moreover, for $\gamma, \delta \in \mathscr{C}$, $\gamma \smile \delta$ iff $\gamma \smile_d \delta$.

Proof: The first statement and direction "if" of the second are trivial. So let $\gamma', \delta' \in \mathscr{C}$ with $\gamma' \smile \delta'$. Then $\gamma' \leftsquigarrow \gamma \smile_d \delta \rightsquigarrow \delta'$ for some components γ and δ. Thus, using insights from the proof of Lemma 5, $\gamma' = static(\gamma') = static(\gamma) \smile_d static(\delta) = static(\delta') = \delta'$. □

The next result says that any two different static components of the same process are concurrent.

Lemma 9 Let $\gamma, \delta \in comp(P)$ for some $P \in \mathbb{P}$. Then $\gamma \smile \delta$ iff $\gamma \neq \delta$.

Proof: "Only if" is trivial. "If" follows by a straightforward structural induction on P. □

We now define a static concurrency relation $\smile\!\!\bullet_s$ between derivations in terms of their static components in the same way that the (dynamic) concurrency relation $\smile\!\!\bullet$ is characterised (by Lemma 7) in terms of their dynamic components:

Definition 14 Derivation $t \in Tr^\bullet$ is *statically unaffected by* u, $t \smile\!\!\bullet_s u$, iff $\forall \gamma \in npc(t). \forall \delta \in afc(u). \gamma \smile \delta$.

The following shows that $\smile\!\!\bullet_s$ is strictly contained in $\smile\!\!\bullet$.

Proposition 7 If $t \smile\!\!\bullet_s u$ then $t \smile\!\!\bullet u$.

Proof: Suppose $t \smile\!\!\bullet_s u$. Let $\varsigma \in n\varsigma(t)$ and $\upsilon \in a\varsigma(u)$. Then $c(\varsigma) \in npc(t)$, $c(\upsilon) \in afc(u)$, so $c(\varsigma) \smile c(\upsilon)$. Hence $c(\varsigma) \smile_d c(\upsilon)$ by Lemma 8, and thus $\varsigma \smile_d \upsilon$. □

In Example 4 we have $t_d \smile t_e$ but $t_d \not\smile_s t_e$, for $npc(t_e) = comp(t_e) = comp(t_d) = afc(t_d) = \{\backslash c\,|_L\}$. Here $t \smile_s u$ denotes $t \smile\!\!\bullet_s u \wedge u \smile\!\!\bullet_s t$. Hence the implication of Proposition 7 is strict.

Lemma 10 Let $source(t) = source(v)$. Then $t \smile\!\!\bullet_s v$ iff $npc(t) \cap afc(v) = \emptyset$.

Proof: Immediately from Lemma 9. □

Write $\varsigma \smile\!\!\bullet_s v$ for ς a synchron and $v \in Tr$ if $c(\varsigma) \smile \gamma$ for all $\gamma \in afc(v)$.

Observation 2 If $\varsigma \smile\!\!\bullet_s v$ then $\varsigma @ v = \varsigma$.

Henceforth we write $t \equiv u$, for $t, u \in Tr^\bullet$, when $n\varsigma(t) = n\varsigma(u)$. In that case $npc(t) = npc(u)$ and $t \rightsquigarrow u$, and thus, by (7), $\ell(t) = \ell(u)$.

Proposition 8 If $t \in Tr^\bullet$, $v \in Tr$ and $t \smile\!\!\bullet_s v$ with $source(t) = source(v)$ then there is a derivation $u \in Tr^\bullet$ with $source(u) = target(v)$ and $t \equiv u$.

Proof: By Proposition 7 $t \smile\!\!\bullet v$. Hence the proof of Proposition 6 finds a $u \in Tr^\bullet$ with $source(u) = target(v)$ and $t \rightsquigarrow u$, such that $n\varsigma(u) = \{\varsigma @ v \mid \varsigma \in n\varsigma(t)\}$. Since $t \smile\!\!\bullet_s v$, $\varsigma \smile\!\!\bullet_s v$ for all $\varsigma \in n\varsigma(t)$, so by Observation 2 $n\varsigma(u) = n\varsigma(t)$, i.e., $t \equiv u$. □

In view of Lemma 10, Proposition 8 says that the functions npc and $afc : Tr \to \mathscr{P}(\mathscr{C})$ satisfy closure property (3) of Section 2.

7.3 Two compatible definitions of the static concurrency relation

The concurrency relation $\smile\!\!\bullet_c$ between transitions defined in terms of static components according to the template in [18], recalled in Section 2, is not identical to the concurrency relation $\smile\!\!\bullet_s$ of Definition 14.

Definition 15 Let t,u be derivations. Write $t \smile\!\!\bullet_c u$ iff $npc(t) \cap afc(u) = \emptyset$.

The following shows that $\smile\!\!\bullet_s$ is strictly included in $\smile\!\!\bullet_c$.

Proposition 9 If $t \smile\!\!\bullet_s u$ then $t \smile\!\!\bullet_c u$.

Proof: This follows immediately from the irreflexivity of $\smile \subseteq \mathscr{C} \times \mathscr{C}$ (Lemma 9). □

Example 8 Let t_1 and t_2 be the unique derivations of the transitions $c[f] \xrightarrow{c} \mathbf{0}[f]$ and $c\backslash L \xrightarrow{c} \mathbf{0}\backslash L$, where $f(c) = c$ and $c \notin L$. Then $(n)\varsigma(t_1) = \{[f](\xrightarrow{c}\mathbf{0})\}$ and $(a)\varsigma(t_2) = \{\backslash L(\xrightarrow{c}\mathbf{0})\}$, so $npc(t_1) = \{[f]\}$ and $afc(t_2) = \{\backslash L\}$. Since $[f] \not\smile \backslash L$ and $[f] \neq \backslash L$, one has $t_1 \not\smile\!\!\bullet_s t_2$ (and $t_1 \not\smile\!\!\bullet t_2$) but $t_1 \smile\!\!\bullet_c t_2$.

Since in Example 4 we have $t_d \smile t_e$ but ($t_d \not\smile\!\!\bullet_s t_e$ and) $t_d \not\smile\!\!\bullet_c t_e$, since $npc(t_d) = afc(t_e) = \{\backslash c\,|_L\}$, it follows also that $\smile\!\!\bullet_c$ is incomparable with $\smile\!\!\bullet$.

Nevertheless, we show that for the study of justness it makes no difference whether justness is defined using the concurrency relation $\smile\!\!\bullet_s$ or $\smile\!\!\bullet_c$.

Lemma 11 If $t \in Tr^\bullet$ and π is a path from $source(t)$ to a state P' such that $t \smile\!\!\bullet_s v$ for all transitions v on π, then there is a derivation $t' \in Tr^\bullet$ with $source(t') = P'$ and $t \equiv t'$.

Proof: This is a corollary of Proposition 8, obtained by a simple induction on the length of π, using the reflexivity and transitivity of $\equiv$, and that $t \smile\!\!\bullet_s v$ and $t \equiv t'$ implies $t' \smile\!\!\bullet_s v$. □

Definition 16 Let $\mathbb{T} = (S, Tr, source, target, \ell)$ be an LTS, and $\smile\!\!\bullet_x \subseteq Tr \times Tr$ a concurrency relation between the transitions, satisfying (1) and (2). Call a path π in $\mathbb{T}$ $\smile\!\!\bullet_x$-B-just, for $B \subseteq Act$, if according to Definition 6 it is B-just in the LTSC $(S, Tr, source, target, \ell, \smile\!\!\bullet_x)$.

Proposition 10 A path is $\smile\!\!\bullet_c$-B-just iff it is $\smile\!\!\bullet_s$-B-just.

Proof: "Only if" is immediate from Definition 6 and Proposition 9.

"If": Let π be $\smile\!\!\bullet_s$-B-just, and let $t \in Tr^\bullet_{\neg B}$ with $P := source(t) \in \pi$. By Definition 6 a transition u occurs in π past the occurrence of P, such that $t \not\smile\!\!\bullet_s u$. W.l.o.g. we take u to be the first such transition in π past P. If suffices to show that $t \not\smile\!\!\bullet_c u$. For all transitions v in π between P and $source(u)$ we have $t \smile\!\!\bullet_s v$. Hence, by Lemma 11, there is a $t' \in Tr^\bullet$ with $source(t') = source(u)$ and $t \equiv t'$. Moreover, $t \not\smile\!\!\bullet_s u$ implies $t' \not\smile\!\!\bullet_s u$, which implies $t' \not\smile\!\!\bullet_c u$ by Lemma 10, which implies $t \not\smile\!\!\bullet_c u$. □

The above proof shows that for the study of justness, we need to know whether two transitions $t \in Tr^\bullet$ and $u \in Tr$ are related by the static concurrency relation or not, only when $\exists t' \in Tr^\bullet$ with $t \equiv t'$ and $source(t') = source(u)$. And restricted to such pairs (t,u) the relations $\smile\!\!\bullet_s$ and $\smile\!\!\bullet_c$ coincide.

7.4 A more abstract definition of static components

The arguments of unary operators occurring in the definition of a static components are essentially redundant. Consider the following alternative definitions:

An *abstract static component* is a string $\sigma \in \{|_L, |_R\}^*$. The *abstract static component* $c'(\varsigma)$ of a synchron ς is defined to be the result of leaving out all argument $\backslash L$ and $[f]$ from $c(\varsigma)$. For a derivation t let $comp'(t) := \{c'(\varsigma) \mid \varsigma \in \varsigma(t)\}$, $npc'(t) := \{c'(\varsigma) \mid \varsigma \in n\varsigma(t)\}$ and $afc'(t) := \{c'(\varsigma) \mid \varsigma \in a\varsigma(t)\}$.

Analogously to Definitions 14 and 15, write $t \smile\!\!\bullet'_s u$ iff $\forall \gamma \in npc'(t).\, \forall \delta \in afc'(u).\, \gamma \smile \delta$, and write $t \smile\!\!\bullet'_c u$ iff $npc'(t) \cap afc'(u) = \emptyset$. Note that $c(\varsigma) \smile c(\upsilon)$ implies $c'(\varsigma) \smile c'(\upsilon)$ for all synchrons ς and υ. Likewise, $c'(\varsigma) \neq c'(\upsilon)$ implies $c(\varsigma) \neq c(\upsilon)$. Hence $\smile\!\!\bullet_s \subseteq \smile\!\!\bullet'_s \subseteq \smile\!\!\bullet'_c \subseteq \smile\!\!\bullet_c$. Thus, Proposition 10 implies that a path is $\smile\!\!\bullet_s$-B-just iff it is $\smile\!\!\bullet'_s$-B-just iff it is $\smile\!\!\bullet'_c$-B-just iff it is $\smile\!\!\bullet_c$-B-just.

8 Computational interpretations

The classical computational interpretation of CCS and related languages aligns with the (dynamic) concurrency relation $\smile\!\!\bullet$ of Sections 6 and 7.1, rather than the static concurrency relation $\smile\!\!\bullet_s$ of Section 7.2. This is illustrated by the transitions t_d and t_e of Example 4, which are generally regarded as concurrent. This computational interpretation also aligns with the semantics of CCS in terms of event structures and Petri nets, where concurrency is made more explicit [35, 19].

Below, we first define a sublanguage of CCS with broadcast communication and/or signals on which the static and dynamic concurrency relations coincide—so it does not include the process P of Example 4. Using this, we propose an alternative computational interpretation of CCS and its extensions that aligns with the static concurrency relation.

The underlying intuition is that a choice necessary needs to be made locally, so that one can not have two truly parallel actions d and e for which the execution of either one constitutes the same choice. In Example 4, the transitions t_τ and t_d are mutually exclusive—one rules out the other—and therefore should be co-located. The same holds for t_τ and t_d, and consequently t_d should be co-located with t_e and not concurrent. This intuition stems from [14, 15]. Similarly, a signal can be emitted only locally, and for convenience we treat recursion in the same vain, so that all dynamic operators can be applied to sequential processes only.

Definition 17 The *dynamically sequential* fragment of CCS with broadcast communication and/or signals is given by the context-free grammar

$$\begin{array}{rcl} S & ::= & \mathbf{0} \mid \alpha.P \mid S+S \mid S\hat{}s \mid A \mid S\backslash L \mid S[f] \\ P & ::= & S \mid P|Q \mid P\backslash L \mid P[f] \end{array}$$

where S is the sort of *sequential processes*, and P the sort of *parallel processes*. Defining equations for agent identifiers should have the form $A \stackrel{def}{=} S$.

This language is crafted in such a way that in all synchrons a dynamic argument will never precede a parallel composition argument $|_{\mathrm{L}}$ or $|_{\mathrm{R}}$. As a consequence, we obtain, for synchrons ς and υ, that $\varsigma \rightsquigarrow \upsilon \Leftrightarrow \varsigma = \upsilon$ and that

$$\varsigma \smile \upsilon \Leftrightarrow \varsigma \smile_d \upsilon \Leftrightarrow C(\varsigma) \smile_d C(\upsilon) \Leftrightarrow c(\varsigma) \smile c(\upsilon)\,.$$

Hence, on this fragment, the concurrency relations $\smile\!\!\bullet$ and $\smile\!\!\bullet_s$ coincide.

Next, we introduce a new unary operator sq, that turns a parallel process into a sequential one. Thus "$|\ sq(P)$" can be added to the line for $S ::=$ in the context-free grammar above. Its operational rules are

$$\frac{P \stackrel{\alpha}{\longrightarrow} P'}{sq(P) \stackrel{\alpha}{\longrightarrow} P'} \qquad \frac{P \stackrel{\kappa}{\longrightarrow} P'}{sq(P) \stackrel{\kappa}{\longrightarrow} sq(P')}$$

where κ ranges over $b!\in\mathscr{B}!$ and $\bar{s}\in\bar{\mathscr{S}}$, so that it changes its argument as little as possible. However, the argument sq is now added to synchrons, counting as dynamic, and Definition 9 of $\smile_d$ is upgraded by the requirement that the argument sq does not occur in σ_1, with $\smile$ redefined to equal $\smile_d$. Consequently, the only effect of sq is that any concurrency between outgoing transitions of its arguments is removed. The process $sq(d.R|e.S)$, for instance, behaves exactly like $d(R|e.S)+e(d.R|S)$.

On this extension of the dynamically sequential fragment of CCS with broadcast communication and/or signals we still have that $\varsigma \smile \upsilon \Leftrightarrow \varsigma \smile_d \upsilon \Leftrightarrow C(\varsigma) \smile_d C(\upsilon) \Leftrightarrow c(\varsigma) \smile c(\upsilon)$, and consequently $\smile\!\!\bullet$ and $\smile\!\!\bullet_s$ coincide.

Finally, we propose a language that has the same syntax as CCS, possibly extended with broadcast communication and/or signals, but is technically a sublanguage of language proposed above, because whenever the operators $+$ or $\hat{}s$ are applied to parallel arguments, $P+Q$ is taken to be an abbreviation of $sq(P)+sq(Q)$, and $P\hat{}s$ of $sq(P)\hat{}s$. Likewise, $A \stackrel{def}{=} P$ can be seen as an abbreviation of $A \stackrel{def}{=} sq(P)$. This language can be seen as an alternative computational interpretation of CCS (plus extensions) that aligns with the static concurrency relation $\smile\!\!\bullet_s$.

Interestingly, the operational Petri net semantics of [6] follows the static computational interpretation above, whereas its modification in [27, 28] follows the classical (dynamic) interpretation of concurrency.

9 The dynamic and static accounts of justness agree

We now show that the concurrency relations $\smile\!\!\bullet$ and $\smile\!\!\bullet_s$ (and thus also the variants $\smile\!\!\bullet'_s$, $\smile\!\!\bullet_c$ and $\smile\!\!\bullet'_c$ of $\smile\!\!\bullet_s$ studied in Sections 7.3 and 7.4) give rise to the same concept of justness.

Each derivation $t \in Tr$ has only finitely many synchrons, and each synchron contains finitely many dynamic arguments. Let $d(t)$ be the sum, over $\varsigma \in n\varsigma(t)$, of the number of dynamic arguments in ς.

Theorem 2 A path is $\smile\!\!\bullet$-B-just iff it is $\smile\!\!\bullet_s$-B-just.

Proof: "Only if" is immediate from Definition 6 and Proposition 7.

"If": Let π be $\smile\!\!\bullet_s$-B-just, and let $t \in Tr^\bullet_{\neg B}$ with $source(t) \in \pi$. By induction on $d(t)$ we find a transition u occurring in π past the occurrence of $source(t)$, such that $t \not\smile\!\!\bullet\, u$.

By Definition 6 a transition u' occurs in π past the occurrence of $source(t)$, such that $t \not\smile\!\!\bullet_s u'$. W.l.o.g. we take $u' \in Tr^\bullet$ to be the first such transition in π past $source(t)$. By Lemma 11 there is a derivation $t' \in Tr^\bullet$ with $source(t') = source(u')$ and $t \equiv t'$. So $t' \not\smile\!\!\bullet_s u'$. Hence there are $\varsigma \in n\varsigma(t')$ and $\upsilon \in a\varsigma(u')$ with $c(\varsigma) = c(\upsilon)$ by Lemma 10. In case $t' \not\smile\!\!\bullet\, u'$ then $t \smile\!\!\bullet\, u$ and we are done. So suppose $t' \smile\!\!\bullet\, u'$. Then $\varsigma \smile \upsilon$, so $\varsigma \smile_d \upsilon$ by Lemmas 2 and 3. Thus ς has the form $\sigma_1|_D\varsigma_2$ and $\upsilon = \sigma_1|_E\upsilon_2$ with $\{D,E\} = \{L,R\}$. Since $c(\varsigma) = c(\upsilon)$, a dynamic operator must occur in σ_1. So by Definition 13 $\varsigma@\upsilon$ contains fewer dynamic arguments than ς, and hence $\varsigma@u'$ contains fewer dynamic arguments than ς. Moreover, for $\varsigma' \neq \varsigma$, $\varsigma'@u'$ contains at most as many dynamic arguments as ς'.

The proof of Proposition 6 finds a $t'' \in Tr^\bullet$ with $source(t'') = target(u)$ and $t' \rightsquigarrow t''$, such that $n\varsigma(t'') = \{\varsigma@u' \mid \varsigma \in n\varsigma(t')\}$. It follows that $d(t'') < d(t')$. By Proposition 5, $\ell(t'') = \ell(t') = \ell(t)$, and so $t'' \in Tr^\bullet_{\neg B}$. By the induction hypothesis we find a transition u occurring in π past the occurrence of $source(t'')$, such that $t'' \not\smile\!\!\bullet\, u$. So u also occurs past the occurrence of $source(t)$ and $t \not\smile\!\!\bullet\, u$, using Proposition 4. □

10 Justness is feasible even with infinitary choice

A straightforward induction of the length of derivations shows that for each process $P \in \mathbb{P}$ in any of the languages of Section 5 there are only countably many derivations $t \in Tr^\bullet$ with $source(t) = P$. Consequently, Theorem 1 says that, for any set $B \subseteq Act$ with $Rec \subseteq B$, B-justness is feasible. However, the standard version of CCS [23] features the infinitary choice operator $\sum_{i\in I} P_i$ for any index set I, which was omitted in Section 5 (and many of the references). Its operational rule is

$$\frac{P_j \xrightarrow{\alpha} P'_j}{\sum_{i\in I} P_i \xrightarrow{\alpha} P'_j} \ (j \in I).$$

The work reported here can be straightforwardly extended with this infinitary choice operator. Instead of $+_{\mathrm{L}}$ and $+_{\mathrm{R}}$ it gives rise to dynamic arguments $\sum^j$ appearing in synchrons. But then we have processes

P with uncountably many outgoing transitions, so that Theorem 1 no longer applies. Nevertheless, B-justness is feasible, as follows from Corollary 1, in conjunction with Theorem 2. For if $t \equiv t'$ (defined in Section 7.2), then $t \smile\!\bullet_s u \Leftrightarrow t' \smile\!\bullet_s u$ for all $u \in Tr^\bullet$. As an $\equiv$-equivalence class is completely determined by a finite set of static components, and the set $\mathscr{C}$ of static components is countable, so is the collection of $\equiv$-equivalence classes of transitions, and thus the set of equivalence classes used in Corollary 1.

11 An inductive characterisation of the concurrency relations $\smile\!\bullet_d$ and $\smile\!\bullet_s$

As a variant of Definition 11 in Section 6, write $t \smile\!\bullet_d u$ if $\forall \varsigma \in n\varsigma(t). \forall \upsilon \in a\varsigma(u). \varsigma \smile_d \upsilon$. By Lemmas 2 and 3, when $source(t) = source(u)$ then $t \smile\!\bullet u \Leftrightarrow t \smile\!\bullet_d u$.

The idea of an asymmetric concurrency relation $\smile\!\bullet$ is not new here. A similar relation, here called $\smile\!\bullet_{15}$, appeared in [17]. That relation was defined only between derivations t and u with $source(t) = source(u)$. Here we show that $\smile\!\bullet_{15}$ agrees with our $\smile\!\bullet$, in the sense that

$$t \smile\!\bullet_{15} u \qquad \text{iff} \qquad t \smile\!\bullet u \ \wedge \ source(t) = source(u)$$

for all $t \in Tr^\bullet$ and $u \in Tr$. In order to prove this, we give an inductive characterisation of $\smile\!\bullet_d$. This effort also yields an inductive characterisation of $\smile\!\bullet_s$, which will be used in Section 12 to provide a coinductive characterisation of justness, in the spirit of the definitions of justness from [11, 17, 7, 3].

Proposition 11 The relation $\smile\!\bullet_d$ is the smallest relation $\smile\!\bullet_x \subseteq Tr^\bullet \times Tr$ such that

- $t|P \smile\!\bullet_x Q|u$ and $P|t \smile\!\bullet_x u|Q$,
- $t|v \smile\!\bullet_x Q|u$ and $v|t \smile\!\bullet_x u|Q$ if $\ell(t) \in \mathscr{B}!$,
- $t \smile\!\bullet_x u$ implies $t+P \smile\!\bullet_x u+R$, $P+t \smile\!\bullet_x R+u$, $t|P \smile\!\bullet_x u|Q$ and $P|t \smile\!\bullet_x Q|u$,
- $t \smile\!\bullet_x u$ implies $t|P \smile\!\bullet_x u|w$, $t|v \smile\!\bullet_x u|Q$, $P|t \smile\!\bullet_x w|u$ and $v|t \smile\!\bullet_x Q|u$,
- $t \smile\!\bullet_x u$ implies $t|v \smile\!\bullet_x u|w$ and $v|t \smile\!\bullet_x w|u$ if $\ell(t) \in \mathscr{B}!$,
- $t \smile\!\bullet_x u \wedge v \smile\!\bullet_x w$ implies $t|v \smile\!\bullet_x u|w$, and
- $t \smile\!\bullet_x u$ implies $t\backslash L \smile\!\bullet_x u\backslash L$, $t[f] \smile\!\bullet_x u[f]$, $A{:}t \smile\!\bullet_x A{:}u$ and $t\hat{}r \smile\!\bullet_x u\hat{}r$ for any $L \subseteq \mathscr{C}_h$, relabelling f, $A \in \mathscr{A}$ and $r \in \mathscr{S}$,
- $t \smile\!\bullet_x \xi$ for any derivation ξ of a signal transition,

for arbitrary t, u, v, w, P, Q and R, where t, v and derivations of signals or transitions, u and w are derivations of non-signal transitions, $P, R \in \mathbb{P}$ are expressions, and Q is either an expression or the derivation of a signal or signal transition—provided that the composed derivations exist and are of the right type.

Proof: It is straightforward to check that $\smile\!\bullet_d$ satisfies all the properties listed in Proposition 11, so the smallest relation $\smile\!\bullet_x$ is contained in $\smile\!\bullet_d$. For the other direction we prove by structural induction on t that if $t' \smile\!\bullet_d u'$ then $t' \smile\!\bullet_x u'$ can be derived by the rules of Proposition 11.

- If u' is the derivation of a signal transition then $a\varsigma(u') = \emptyset$, so $t' \smile\!\bullet_d u'$. Correspondingly, $t' \smile\!\bullet_x u'$, by the last requirement of Proposition 11. So below u' is the derivation of a non-signal transition.
- If t' has the form $\xrightarrow{\alpha} P$ then $t' \smile\!\bullet_d u'$ for no u', so there is nothing to show.
- Let $t' = t[f]$. Then all synchrons of t' start with $[f]$, so for $t' \smile\!\bullet_d u'$ to hold, all active synchrons of u' must start with $[f]$ as well. In fact u' must have the form $u[f]$ such that $t \smile\!\bullet_d u$. By induction $t \smile\!\bullet_x u$ and hence $t' \smile\!\bullet_x u'$ by the seventh requirement of Proposition 11.
- The cases $t' = t\backslash L$, $t' = \mathscr{A}{:}t$, $t' = t\hat{}r$, $t' = t+P$ and $t' = P+t$ proceed in the same way.
- Let $t' = t|P$. We make a further case distinction on u'.
  - Let $u' = Q|u$. Then always $t' \smile\!\bullet_d u'$, and indeed $t' \smile\!\bullet_x u'$ by the first requirement on $\smile\!\bullet_x$.

  - Let $u' = u|Q$. Then all synchrons of t' and all active synchrons of u' start with $|_{\mathrm{L}}$, and stripping those off shows that $t \frown\!\!\bullet_d u$. By induction $t \frown\!\!\bullet_x u$ and hence $t' \frown\!\!\bullet_x u'$ by the third requirement.
  - Let $u' = u|w$. Then all synchrons of t' and some of u' start with $|_{\mathrm{L}}$, and stripping those off shows that $t \frown\!\!\bullet_d u$. By induction $t \frown\!\!\bullet_x u$ and hence $t' \frown\!\!\bullet_x u'$ by the fourth requirement on $\frown\!\!\bullet_x$.
  - If u has any other shape, then $t' \not\frown\!\!\bullet_d u'$, so there is nothing to show.
- The case $t' = P|t$ proceeds symmetrically.
- Let $t' = t|v$. We make a further case distinction on u'.
  - Let $u' = Q|u$. First consider the case that $\ell(t) \in \mathscr{B}!$. Then $\ell(v) \in \mathscr{B}? \cup \mathscr{B}:$ and all necessary synchrons of t' start with $|_{\mathrm{L}}$. Since all active synchrons of u' start with $|_{\mathrm{R}}$, we have $t' \frown\!\!\bullet_d u'$. Accordingly, $t' \frown\!\!\bullet_x u'$ by the second requirement on $\frown\!\!\bullet_x$.
    In case $\ell(t) \notin \mathscr{B}!$, some necessary synchrons of t' and all active synchrons of u start with $|_{\mathrm{R}}$, and stripping those off shows that $v \frown\!\!\bullet_d u$. By induction $v \frown\!\!\bullet_x u$ and hence $t' \frown\!\!\bullet_x u'$ by the fourth requirement of Proposition 11 (reversing the roles of t and v).
  - The case $u' = u|Q$ proceeds symmetrically.
  - Let $u' = u|w$. First consider the case that $\ell(t) \in \mathscr{B}!$. Then $\ell(v) \in \mathscr{B}? \cup \mathscr{B}:$, and all necessary synchrons of t' start with $|_{\mathrm{L}}$. Stripping those off shows that $t \frown\!\!\bullet_d u$. By induction $t \frown\!\!\bullet_x u$ and hence $t' \frown\!\!\bullet_x u'$ by the fifth requirement on $\frown\!\!\bullet_x$.
    The case that $\ell(v) \in \mathscr{B}!$ proceeds symmetrically.
    Otherwise, we obtain $t \frown\!\!\bullet_d u$ and $v \frown\!\!\bullet_d w$. By induction $t \frown\!\!\bullet_x u$ and $v \frown\!\!\bullet_x w$ and hence $t' \frown\!\!\bullet_x u'$ by the sixth requirement of Proposition 11
  - If u has any other shape, then $t' \not\frown\!\!\bullet_d u'$, so there is nothing to show. □

The relation $\frown\!\!\bullet_{15}$ was defined in [17] for ABC. Its definition is almost the same as the one of $\frown\!\!\bullet_x$ in Proposition 11, but simplified because there are no signals or signal transitions, and adding the requirement that $\mathit{source}(t) = \mathit{source}(u)$.

Corollary 4 Let $t \in Tr^\bullet$ and $u \in Tr$. Then $t \frown\!\!\bullet_{15} u$ iff $t \frown\!\!\bullet u \wedge \mathit{source}(t) = \mathit{source}(u)$.

Proof: A trivial structural induction on t. □

In spite of this agreement between $\frown\!\!\bullet_{15}$ and $\frown\!\!\bullet$, the former is not suitable as an alternative for the latter for the purposes of this paper, because the formalisation of justness depends on judgements $t \not\frown\!\!\bullet u$ for transitions t and u with $\mathit{source}(t) \neq \mathit{source}(u)$.

Proposition 12 The relation $\frown\!\!\bullet_s$ from Section 7.2 is the smallest relation $\frown\!\!\bullet_x \subseteq Tr^\bullet \times Tr$ such that

- $t|P \frown\!\!\bullet_x Q|u$ and $P|t \frown\!\!\bullet_x u|Q$,
- $t|v \frown\!\!\bullet_x Q|u$ and $v|t \frown\!\!\bullet_x u|Q$ if $\ell(t) \in \mathscr{B}!$,
- $t \frown\!\!\bullet_x u$ implies $t|P \frown\!\!\bullet_x u|Q$ and $P|t \frown\!\!\bullet_x Q|u$,
- $t \frown\!\!\bullet_x u$ implies $t|P \frown\!\!\bullet_x u|w$, $t|v \frown\!\!\bullet_x u|Q$, $P|t \frown\!\!\bullet_x w|u$ and $v|t \frown\!\!\bullet_x Q|u$,
- $t \frown\!\!\bullet_x u$ implies $t|v \frown\!\!\bullet_x u|w$ and $v|t \frown\!\!\bullet_x w|u$ if $\ell(t) \in \mathscr{B}!$,
- $t \frown\!\!\bullet_x u \wedge v \frown\!\!\bullet_x w$ implies $t|v \frown\!\!\bullet_x u|w$, and
- $t \frown\!\!\bullet_x u$ implies $\mathrm{p}\, t\backslash L \frown\!\!\bullet_x u\backslash L$ and $t[f] \frown\!\!\bullet_x u[f]$ for any $L \subseteq \mathscr{C}_h$ and relabelling f,
- $t \frown\!\!\bullet_x \xi$ for any derivation of a signal transition ξ,

for arbitrary t, u, v, w, P, Q and R, where t, v and derivations of signals or transitions, u and w are derivations of non-signal transitions, $P, R \in \mathbb{P}$ are expressions, and Q is either an expression or the derivation of a signal or signal transition—provided that the composed derivations exist are of the right type.

Proof: A trivial adaptation of the proof of the previous proposition. □

12 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 [17].

To obtain agreement between our $\smile\!\!\bullet$-based and coinductive definitions for CCSS, we first extend the $\smile\!\!\bullet$-based concept of B-justness to the case where also CCSS-signals from $\bar{\mathscr{S}}$ may appear in B. Since this extension is unsuitable as a completeness criterion, and hence should not be confused with the proper concept of justness, we have not treated this extension from the start of the paper, and give it another name: B-sigjustness. This extension is needed because in the coinductive definition, some cases of proper B-justness depend on cases of C-sigjustness, where C involves signals.

12.1 An extension of the notion of justness dealing with signals

Let $Tr^{s\bullet} := \{t \in Tr \mid \ell(t) \in \mathscr{L} \setminus (\mathscr{B}? \cup \mathscr{B}:)\}$. Then $Tr^{\bullet} \subseteq Tr^{s\bullet}$, and $Tr^{s\bullet} \setminus Tr^{\bullet} = \{t \in Tr \mid \ell(t) \in \bar{\mathscr{S}}\}$. So the difference between $Tr^{s\bullet}$ and $Tr^{\bullet}$ shows up only for CCSS with signals modelled as transitions, and consists of the signal transitions. We extend the definition of the necessary synchrons of a transition t to $t \in Tr^{s\bullet}$ by declaring $n\varsigma(t) := \varsigma(t)$ when $\ell(t) \in \bar{\mathscr{S}}$. Using $Tr^{s\bullet}$ instead of $Tr^{\bullet}$ extends Definition 10 of $\rightsquigarrow$ and Definition 11 of $\smile\!\!\bullet$ to relations $\rightsquigarrow \subseteq Tr^{s\bullet} \times Tr^{s\bullet}$ and $\smile\!\!\bullet \subseteq Tr^{s\bullet} \times Tr$. Likewise, the relations $\smile\!\!\bullet_s$, $\smile\!\!\bullet_c$, $\smile\!\!\bullet'_s$, $\smile\!\!\bullet'_c$ of Section 7 extend to $Tr^{s\bullet} \times Tr$. Now all results of Section 6 continue to hold when substituting $Tr^{s\bullet}$ for $Tr^{\bullet}$, except for the disjointness claim of Proposition 3, and consequently (a) the part of (7) saying that $t \rightsquigarrow t'$ implies $t \smile\!\!\bullet t'$, (b) Corollary 3, and (c) the statement that the LTS $(\mathbb{P}, Tr, source, target, \ell, \smile\!\!\bullet)$, augmented with the concurrency relation $\smile\!\!\bullet$, is an LTSC in the sense of Definition 5.

Definition 18 A path π in an LTSC is *B-sigjust*, for $\mathscr{B}? \subseteq B \subseteq Act \cup \bar{\mathscr{S}}$, if for each transition $t \in Tr^{s\bullet}_{\neg B}$ with $s := source(t) \in \pi$, a transition u occurs in π past the occurrence of s, such that $t \not\smile\!\!\bullet u$.

B-sigjust corresponds with what is called *$\bar{B} \cap \mathscr{S}$-signalling and $B \setminus \bar{\mathscr{S}}$-just* in [7]. Here we save double work by collapsing the similar concepts *signalling* and *just* from [7]. Note that a path is B-just in the sense of Definition 6, where $\mathscr{B}? \subseteq B \subseteq Act$, iff it is $B \cup \bar{\mathscr{S}}$-sigjust according to Definition 18.

Substituting $Tr^{s\bullet}$ for $Tr^{\bullet}$ and "sigjust" for "just", also all results of Sections 7, 9 and 11 continue to hold. However, sigjustness is unsuitable as a completeness criterion, because it fails the requirement of feasibility.

Example 9 The process $\mathbf{0}\hat{}s$ has only one path π, and π has no transitions. This path is not $\emptyset$-sigjust, since a transition $\mathbf{0}^{\rightarrow s}$ is enabled in its only state. So π can not be extended into an $\emptyset$-sigjust path.

Changing the definition of a path to allow signal transitions does not help; this allows an infinite path π' containing the transition $\mathbf{0}^{\rightarrow s}$ infinitely often. But since $\mathbf{0}^{\rightarrow s} \smile\!\!\bullet \mathbf{0}^{\rightarrow s}$, also π' fails to be $\emptyset$-sigjust.

12.2 A coinductive definition of justness

To state our 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 $t \in Tr$ of a transition with $source(t) = P|Q$ has the shape

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

Let a path *of* a process P be a path as in Definition 3 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, $t \in Tr$ with $source(t) = P[f]$ has the shape $u[f]$ with $target(t) = target(u)[f]$. 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 [17, Appendix E].[5] To interpret the word "largest", one can see justness equivalently as single predicate on $\mathscr{P}(Act) \times \Pi$, where Π denotes the set of all paths. To see that there actually exists a largest such predicate, check that the class of all such predicates is closed under arbitrary unions.

Definition 19 *B-justness*, for $\mathscr{B}? \subseteq B \subseteq Act$, 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 (cf. Footnote 4 on 12);
- 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 $\tau \in B \vee C \cap \bar{D} = \emptyset$—here $\bar{D} := \{\bar{c} \mid c \in D\}$;
- 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.

To make this definition apply to CCSS (but with the semantics of Section 5.4 only), as well as CCS, ABC and ABCd, read "sigjust" for "just" throughout, and allow $\mathscr{B}? \subseteq B \subseteq Act \cup \bar{\mathscr{S}}$.

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 $A \stackrel{def}{=} a.A$. 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 10 To illustrate Definition 19 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 19 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 19, because its last state 1 admits a transition.

[5] To be precise, the notion of Y-justness from [17] translates to $Y \cup \mathscr{B}?$-justness as occurs here. Furthermore, [17] restricts to the case that $Y \subseteq \mathscr{C}_h \cup \bar{\mathscr{C}}_h$. This makes sense, as in the default computational interpretation broadcast actions $b!$ and internal actions τ can not be blocked by the environment. The increased generality occurring in this paper is merely because it comes with no extra costs, and in fact saves us here and there from listing restrictions.

12.3 Agreement between the concurrency-based and coinductive definitions of justness

We now establish that the concept of justness from Definition 19 agrees with the concept of justness defined earlier in this paper. The below applies to CCSS by reading $Tr^{s\bullet}$ for $Tr^{\bullet}$ and "sigjust" for "just".

Theorem 3 A path is $\leadsto_s$-B-just iff it is B-just in the sense of Definition 19.

Proof: "Only if": It suffices to show that $\leadsto_s$-B-justness satisfies the five requirements of Definition 19.

- Let π be a $\leadsto_s$-B-just path. It follows immediately from Definition 6 that its last state admits no transitions $t \in Tr^{\bullet}_{\neg B}$.
- Let π be a $\leadsto_s$-B-just path of a process $P|Q$. There is a unique decomposition $\pi = \pi_1|\pi_2$ of π into a path π_1 of P and a path π_2 of Q. Let C' be the set of actions α such that there is a $t \in Tr^{\bullet}$ with $s := source(t) \in \pi_1$ and $\ell(t) = \alpha$, but no transition u with $t \not\leadsto_s u$ occurs in π_1 past the occurrence of s. Take $C := C' \cup \mathscr{B}?$. Then π_1 is $\leadsto_s$-C-just. In fact, C is the smallest set $\mathscr{B}? \subseteq X \subseteq Act$ such that π_1 is X-just. Likewise, let D be the smallest set such that $\mathscr{B}? \subseteq D \subseteq Act$ and π_2 is D-just. It remains to be shown that $C, D \subseteq B$ and $\tau \in B \vee C \cap \bar{D} = \emptyset$.
  Let $\alpha \in C'$. Then there is a state $P'|Q'$ in π and a $t \in Tr^{\bullet}$ with $P' = source(t) \in \pi_1$ and $\ell(t) = \alpha$, but no transition u with $t \not\leadsto_s u$ occurs in π_1 past the occurrence of P'. We claim that $\alpha \in B$.
  - Let $\alpha \in \mathscr{S} \cup \bar{\mathscr{S}} \cup \mathscr{C}_h \cup \bar{\mathscr{C}}_h \cup \{\tau\}$. Then $t|Q' \in Tr^{\bullet}$ with $P'|Q' = source(t|Q') \in \pi$ and $\ell(t|Q') = \alpha$. Suppose, towards a contradiction, that $\alpha \notin B$. Then, using the $\leadsto_s$-B-justness of π, a transition $t^\dagger$ must occur in π past the occurrence of $P'|Q'$, such that $t|Q' \not\leadsto_s t^\dagger$. Since $t^\dagger$ occurs in π, $source(t^\dagger)$ has the form $P''|Q''$. By Proposition 12, $t^\dagger$ must have the form $u|v$ or $u|Q''$, with $t \not\leadsto_s u$. Hence a transition u with $t \not\leadsto_s u$ occurs in π past the occurrence of P'—a contradiction. So $\alpha \in B$.
  - Let $\alpha = b!$ with $b \in \mathscr{B}!$. Then either $t|Q' \in Tr^{\bullet}$ with $\ell(t|Q') = \alpha$, or $t|v \in Tr^{\bullet}$ with $\ell(v) = b?$ or $\ell(v) = b:$ and $\ell(t|v) = b! = \alpha$. In both cases the argument proceeds as above.

  It follows that $C \subseteq B$. By symmetry also $D \subseteq B$.
  Let $c \in C$ and $\bar{c} \in D$. Then there are states $P_1|Q_1$ and $P_2|Q_2$ in π and $t_1, t_2 \in Tr^{\bullet}$ with
  - $P_1 = source(t_1) \in \pi_1$ and $\ell(t_1) = c$, but no u with $t_1 \not\leadsto_s u$ occurs in π_1 past P_1, and
  - $Q_2 = source(t_2) \in \pi_2$ and $\ell(t_2) = \bar{c}$, but no w with $t_2 \not\leadsto_s w$ occurs in π_2 past Q_2.

  Assume that either $P_1|Q_1 = P_2|Q_2$ or the state $P_2|Q_2$ occurs in π past the state $P_1|Q_1$—the other case will follow by symmetry. By Lemma 11 there is a $t_1' \in Tr^{\bullet}$ with $source(t_1') = P_2$ and $t_1 \equiv t_1'$. So $\ell(t') = c$. Now $t_1'|t_2 \in Tr^{\bullet}$ and $source(t_1'|t_2) = P_2|Q_2$. Moreover, $\ell(t_1'|t) = \tau$. Assume that $\tau \notin B$. Then, using the $\leadsto_s$-B-justness of π, a transition $t^\dagger$ must occur in π past the occurrence of $P_2|Q_2$, such that $t_1'|t_2 \not\leadsto_s t^\dagger$. By Proposition 12 $t^\dagger$ must have the form $P'|v$ or $u|v$ or $u|Q'$ with $t_1' \not\leadsto_s u$ or $t_2 \not\leadsto_s v$. Again we obtain a contradiction.
- Let π be a $\leadsto_s$-B-just path of a process $P\backslash L$. Let π' be the decomposition of π. We have to show that π' is $\leadsto_s$-$(B \cup \{c, \bar{c} \in Act \mid c \in L\})$-just. So assume $t \in Tr^{\bullet}$ with $\ell(t) \notin B \cup \{c, \bar{c} \in Act \mid c \in L\}$, and $P' := source(t) \in \pi'$. Then $P'\backslash L = source(t\backslash L) \in \pi$ and $\ell(t\backslash L) \notin B$. By the $\leadsto_s$-B-justness of π, a transition $t^\dagger$ must occur in π past the occurrence of $P'\backslash L$, such that $t\backslash L \not\leadsto_s t^\dagger$. Now $t^\dagger$ must have the form $u\backslash L$, and by Proposition 12 $t \not\leadsto_s u$. So a transition u occurs in π' past the occurrence of P', such that $t \not\leadsto_s u$.
- The case that π is a $\leadsto_s$-B-just path of a process $P[f]$ goes likewise.
- Finally, it follows directly from Definition 6 that each suffix of a $\leadsto_s$-B-just path is $\leadsto_s$-B-just.

"If": Let $t \in Tr^{\bullet}_{\neg B}$ with $s := source(t) \in \pi$ for a path π that is B-just in the sense of Definition 19. We have to show that a transition $t^\dagger$ with $t \not\leadsto_s t^\dagger$ occurs in π past the occurrence of s. Using the last requirement

of Definition 19 we may assume, without loss of generality, that s is the first state of π. We proceed by structural induction on t.

- Let t have the form $\stackrel{\alpha}{\rightarrow}P$ or $P^{\rightarrow r}$ or $P+u$ or $u+Q$ or $A{:}u$ or $t\hat{}r$. Then $npc(t)=\{\varepsilon\}$. Using the first requirement of Definition 19, s cannot be the last state of π, for it admits a transition t with $\ell(t)\notin B$. Since s has the form $\alpha.P$ or $P+Q$ or A or $P\hat{}r$, the first transition v of π satisfies $afc(v)=\{\varepsilon\}$, and thus $t\not\frown^{\bullet}_{s} v$.
- Let t have the form $u|v$. Then s has the form $P|Q$, with $P:=source(u)$ and $Q:=source(v)$. By the second requirement of Definition 19, $\pi\Rrightarrow\pi_1|\pi_2$, with π_1 a C-just path of P and π_2 a D-just path of Q, for some $C,D\subseteq B$ such that $\tau\in B\vee C\cap\bar{D}=\emptyset$.
  - Let $\ell(u)=c\in\mathscr{S}\cup\bar{\mathscr{S}}\cup\mathscr{C}_h\cup\bar{\mathscr{C}}_h$. Then $\ell(v)=\bar{c}$. Since $t\in Tr^{\bullet}_{\neg B}$, $\tau=\ell(t)\notin B$. So either $c\notin C$ or $\bar{c}\notin D$—by symmetry assume the former.
  - Let $\ell(u)=\ell(t)=b!$ with $b\in\mathscr{B}$. (The case $\ell(v)=b!$ follows by symmetry.) Then $b!\notin B\supseteq C$.

  So in all relevant cases $u\in Tr^{\bullet}_{\neg C}$. By induction, a transition $u^\dagger$ with $u\not\frown^{\bullet}_{s} u^\dagger$ occurs in π_1. Consequently, a transition $t^\dagger$ of the form $u^\dagger|Q'$ or $u^\dagger|v^\dagger$ occurs in π. By Proposition 12 $t\not\frown^{\bullet} t^\dagger$.
- Let t have the form $u|Q$. Then s has the form $P|Q$, with $P:=source(u)$. By the second requirement of Definition 19, $\pi\Rrightarrow\pi_1|\pi_2$, with π_1 a C-just path of P and π_2 a D-just path of Q, for some $C,D\subseteq B$. Since $\ell(u)=\ell(t)\notin B\supseteq C$, $u\in Tr^{\bullet}_{\neg C}$. The argument proceeds as above.
- The case that t has the form $P|v$ follows by symmetry.
- Let t have the form $u\backslash L$. Then s has the form $P\backslash L$, with $P:=source(u)$. Moreover $\ell(u)=\ell(t)\notin B\cup\{c,\bar{c}\in Act\mid c\in L\}$. By the third requirement of Definition 19, the decomposition π' of π is $(B\cup\{c,\bar{c}\in Act\mid c\in L\})$-just. So by induction, a transition $u^\dagger$ with $u\not\frown^{\bullet}_{s} u^\dagger$ occurs in π'. Consequently, a transition $t^\dagger=u^\dagger\backslash L$ occurs in π. By Proposition 12 $t\not\frown^{\bullet} t^\dagger$.
- Let t have the form $u[f]$. Then s has the form $P[f]$, with $P:=source(u)$. Moreover $\ell(u)\notin f^{-1}(B)$. By the fourth requirement of Definition 19, the decomposition π' of π is $f^{-1}(B)$-just. So by induction, a transition $u^\dagger$ with $u\not\frown^{\bullet}_{s} u^\dagger$ occurs in π'. Consequently, a transition $t^\dagger=u^\dagger[f]$ occurs in π. By Proposition 12 $t\not\frown^{\bullet} t^\dagger$. □

13 Justness on abstract paths

By Definition 3, a path is an alternating sequence of states and non-signal transitions. These non-signal transitions are, in the LTS for CCS and its extensions constructed in Section 6, actually *derivations* of transitions $P\stackrel{\alpha}{\longrightarrow}Q$ according to the structural operational semantics of these languages. Now define an *abstract path* to be an alternating sequence of states and actual transitions $P\stackrel{\alpha}{\longrightarrow}Q$.

Definition 20 Let $\widehat{\cdot}$ be the function that takes a derivation $t\in Tr^{\circ}$ into the transition $P\stackrel{\alpha}{\longrightarrow}Q$ derived by t. Given a path $\pi=s_0t_1s_1t_2s_2\cdots$, let $\widehat{\pi}:=s_0\widehat{t_1}s_1\widehat{t_2}s_2\cdots$. An *abstract path* is such an object $\widehat{\pi}$.

The concept of justness naturally lifts from path to abstract paths:

Definition 21 An abstract path ρ is B-just iff there exists a B-just (concrete) path π such that $\rho=\widehat{\pi}$.

This definition fits with the intuition that a path is just iff it models a run that can actually occur.

The following variant of Definition 6 defines $\not\frown^{\bullet}_{s}$-B-justness directly on abstract paths.

Definition 22 A abstract path ρ is $\not\frown^{\bullet}_{s}$-B-*just*, for $\mathscr{B}?\subseteq B\subseteq Act$, if for each derivation $t\in Tr^{\bullet}_{\neg B}$ with $P:=source(t)\in\rho$, there is a $u\in Tr$ with $t\not\frown^{\bullet}_{s} u$ such that $\widehat{u}$ occurs in ρ past the occurrence of P.

We proceed to show that Definitions 21 and 22 agree.

Proposition 13 An abstract path is $\not\rightarrow^{\bullet}_{s}$-$B$-just in the sense of Definition 22 iff it is B-just in the sense of Definition 21, i.e. iff it has the form $\widehat{\pi}$ for a concrete path π that is $\not\rightarrow^{\bullet}_{s}$-$B$-just in the sense of Definition 16.

Proof: "If": Let π be a concrete path that is $\not\rightarrow^{\bullet}_{s}$-$B$-just. Then by Definition 22 $\widehat{\pi}$ is $\not\rightarrow^{\bullet}_{s}$-$B$-just.

"Only if": Let ρ be an abstract path that is $\not\rightarrow^{\bullet}_{s}$-$B$-just in the sense of Definition 22. We present an algorithm for constructing a concrete path π that is $\not\rightarrow^{\bullet}_{s}$-$B$-just in the sense of Definition 16, such that $\rho = \widehat{\pi}$. Following the idea behind the proof of Theorem 1, we build an $\mathbb{N} \times \mathbb{N}$-matrix with a column for each of the states $P_0, P_1, P_2, \ldots$ of ρ. The column P_i lists the transitions from $Tr^{\bullet}_{\neg B}$ enabled in state P_i, leaving empty most slots if there are only finitely many.[6] Incrementally, we construct prefixes π_i of π. As an invariant, we maintain that $\widehat{\pi}_i$ is the prefix of ρ with i transitions. So π_i ends in state P_i. An entry in the matrix is either empty, filled in with a transition, or crossed out. Let $f : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ be an enumeration of the entries in this matrix.

At the beginning, take π_0 to be the path consisting of the first state P_0 of ρ only. At each step $i \geq 0$ we extend the path π_i into π_j for some $j > i$, if possible, and cross out some transitions occurring in the matrix. As an invariant, we maintain that a transition t occurring in the k-th column is already crossed out when reaching step $i > k$ iff a transition u occurs in the extension of π_k into π_i such that $t \not\rightarrow^{\bullet}_{s} u$. Furthermore, when reaching step i, no entry in a column $\ell \geq i$ is already crossed out. At each step $i \geq 0$ we proceed as follows:

We take $n \in \mathbb{N}$ to be the smallest value such that entry $f(n) = (k,m) \in \mathbb{N} \times \mathbb{N}$—with k a column number—satisfies $k \leq i$ and is filled in, say with $t \in Tr^{\bullet}_{\neg B}$, but not yet crossed out. If such an n does not exist, just extend π_i with an arbitrary transition u such that $\widehat{u}$ is the next transition of ρ; if ρ ends in P_i, the algorithm terminates, with output π_i. By our invariant, all transitions v occurring in the extension of π_k into π_i satisfy $t \smile^{\bullet}_{s} v$. By Lemma 11 there is a $t' \in Tr^{\bullet}_{\neg B}$ with $source(t') = P'$ and $t \equiv t'$. Since ρ is $\not\rightarrow^{\bullet}_{s}$-$B$-just, there is a $u \in Tr$ with $t' \not\rightarrow^{\bullet}_{s} u$ such that $\widehat{u}$ occurs in ρ past the occurrence of P. So also $t \not\rightarrow^{\bullet}_{s} u$. Now extend π_i into π_j, for $j > i$, such that π_j ends with the transition u. Cross out all entries in the matrix up to row j necessary to maintain the invariant above. This includes entry $f(n)$.

The desired path π is the limit of all the π_i. It is $\not\rightarrow^{\bullet}_{s}$-$B$-just, using the invariant, because each transition $t \in Tr^{\bullet}_{\neg B}$ that is enabled in a state of π appears in the matrix, which acts like a priority queue, and is eventually crossed out. □

Corollary 5 Let ρ be an abstract path. If ρ is B-just then it is C-just for any $C \supseteq B$.
If ρ is C-just as well as D-just, then it is $C \cap D$-just.

In fact the collection of sets B such that a given abstract path ρ is B-just is closed under arbitrary intersection, and thus there is a least set B_ρ such that ρ is B-just. Actions $\alpha \in \mathscr{B}_\rho$ are called ρ-*enabled* [16]. An action α is ρ-enabled iff there is a derivation $t \in Tr^{\bullet}_{\neg B}$ with $P := source(t) \in \rho$, such that $t \smile^{\bullet}_{s} v$ for all $v \in Tr$ such that $\widehat{v}$ occurs in ρ past the occurrence of P. As a consequence of Definition 6, the same closure properties apply to justness on concrete paths, but for abstract paths these results are much less trivial.

We now show that the concepts of justness on abstract paths from Definitions 21 and 22 both agree with the original definition of justness from [17]. This requires lifting the definition of decomposition from concrete to abstract paths.

Definition 23 An abstract path ρ of a process $P|Q$ *can be decomposed* into abstract paths ρ_1 of P and ρ_2 of Q, notation $\rho \in \rho_1|\rho_2$, if there exist paths π, π_1 and π_2 such that $\pi \Rrightarrow \pi_1|\pi_2$, $\rho = \widehat{\pi}$ and $\rho_i = \widehat{\pi}_i$.

Likewise, an abstract path ρ of $P[f]$ *can be decomposed* into an abstract path ρ' if there are π and π' with $\pi \Rrightarrow \pi'[f]$, $\rho = \widehat{\pi}$ and $\rho' = \widehat{\pi}'$. The decomposition of an abstract path ρ of $P\backslash L$ is defined likewise.

[6] In case we have an infinite choice operator in the language, the set of transitions t with $source(t) = P_i$ is not necessarily countable. Then we work with $\equiv$-classes of transitions, just as in Section 10.

In [17, Section 4.3] the decomposition of an abstract path was defined in a different style, but using [17, Observation E.3] the resulting notion of decomposition is the same.

In [17, Definition 4.1] B-justness was defined directly on abstract paths. The definition is the same as the one for concrete paths—see Definition 19—but reading "abstract path" for "path". The following theorem says that that definition agrees with Definition 21 above.

Theorem 4 An abstract path is B-just in the sense of Definition 19 iff it is B-just in the sense of Definition 21, i.e. iff it has the form $\widehat{\pi}$ for a concrete path π that is B-just in the sense of Definition 19.

Proof: "If": It suffices to show that the family of predicates B-justness on abstract paths according to Definition 21 satisfies the five requirements of Definition 19. This is straightforward to check, and spelled out in [17, Proof of Proposition E.4].

"Only if": Let ρ be an abstract path that is B-just in the sense of Definition 19. By Proposition 13 and Theorem 3, it suffices to show that ρ is $\not\rightarrow^\bullet_s$-$B$-just in the sense of Definition 22. This proceeds just as in the "If"-part of the proof of Theorem 3. □

All definitions and results in this section apply equally well to "sigjustness" in the role of "justness", allowing $\mathscr{B}? \subseteq B \subseteq Act \cup \bar{\mathscr{S}}$. In this form B-sigjustness is the same as B-justness as defined in [3].[7] Finally, we compare our definitions of justness to the one in [7].

Proposition 14 An abstract path is Y-signalling as defined in [7] iff it is $\bar{Y} \cup Act$-sigjust by Definition 19.

Proof: "If": $\bar{Y} \cup Act$-sigjustness trivially satisfies the five conditions for Y-signalling of [7, Definition 2]. The second condition (for paths starting in $P|Q$) uses the first statement of Corollary 5.

"Only if": Call an abstract path ρ B-sigjust† iff $B = \bar{Y} \cup Act$ for an $Y \subseteq \bar{\mathscr{S}}$ such that ρ is Y-signalling as defined in [7]. Then trivially B-sigjustness† satisfies the five conditions of Definition 19. □

Proposition 15 An abstract path is Y-just as defined in [7] iff it is B-sigjust according to Definition 19 for some B with $Y = B \cap Act$, which is the case iff it is $Y \cup \bar{\mathscr{S}}$-sigjust according to Definition 19.

Proof: "If": Call an abstract path Y-just† iff it is B-sigjust by Definition 19 for some B with $Y = B \cap Act$. Then trivially B-justness† satisfies the five conditions of [7, Definition 3]. The second condition (for paths starting in $P|Q$) uses Proposition 14 in conjunction with the first statement of Corollary 5.

"Only if": It suffices to show that each abstract path ρ that is Y-just as defined in [7] is also $\not\rightarrow^\bullet_s$-$Y \cup \bar{\mathscr{S}}$-sigjust according to Definition 22. So for each $t \in Tr^\bullet$ with $\ell(t) \in Act \setminus Y$ and $s := source(t) \in \rho$ for such a path ρ, we have to find a transition $t^\dagger$ with $t \not\rightarrow^\bullet_s t^\dagger$ such that $\widehat{t^\dagger}$ occurs in ρ past the occurrence of s. The proof of this statement is similar to direction "If" of the proof of Theorem 3. Using the last requirement of [7, Definition 3] we may assume, without loss of generality, that s is the first state of ρ. We proceed by structural induction on t. The only case that deviates from the proof of Theorem 3 is where t has the form $u|v$. In this case s has the form $P|Q$, with $P := source(u)$ and $Q := source(v)$. By the second requirement of [7, Definition 3], $\rho \Rrightarrow \rho_1|\rho_2$, with ρ_1 an X-just and X'-signalling abstract path of P and ρ_2 a Z-just and Z'-signalling abstract path of Q, for some $X, Z \subseteq Y$ and $X', Z' \subseteq \mathscr{S}$, such that (when $\tau \notin Y$) $X \cap \bar{Z} = \emptyset$, $X \cap Z' = \emptyset$ and $X' \cap Z = \emptyset$.

- Let $\ell(u) = c \in \mathscr{C}_h \cup \bar{\mathscr{C}}_h$. Then $\ell(v) = \bar{c}$ and $\tau = \ell(t) \notin Y$. So either $c \notin X$ or $\bar{c} \notin Z$—by symmetry assume the former.
- Let $\ell(u) = s \in \mathscr{S}$. Then $\ell(v) = \bar{s} \in \bar{\mathscr{S}}$ and $\tau = \ell(t) \notin Y$. So either $s \notin X$ or $s \notin Z'$.
- The case $\ell(v) = c \in \mathscr{S}$ will follow by symmetry.
- Let $\ell(u) = \ell(t) = b!$ with $b \in \mathscr{B}$. (The case $\ell(v) = b!$ follows by symmetry.) Then $b! \notin Y \supseteq X$.

[7] Following [17, 16], [3] restricts to the case that $B \subseteq \mathscr{C}_h \cup \bar{\mathscr{C}}_h \cup \mathscr{S} \cup \bar{\mathscr{S}}$—cf. Footnote 5. Also, in [16, 3] one has $\mathscr{B}? = \emptyset$.

So in all but one of the relevant cases $u \in Tr^\bullet$ with $\ell(u) \in Act \setminus X$. By induction, there is a transition $u^\dagger$ with $u \not\smile\!\!\bullet_s u^\dagger$ such that $\widehat{u^\dagger}$ occurs in ρ_1. Consequently, there is a transition $t^\dagger$ of the form $u^\dagger|Q'$ or $u^\dagger|v^\dagger$ such that $\widehat{t^\dagger}$ occurs in ρ. By Proposition 12 $t \not\smile\!\!\bullet\, t^\dagger$.

In the remaining case $\ell(v) = \bar{s} \in \bar{\mathscr{S}}$ and ρ_2 is Z'-signalling with $s \notin Z'$. Using Proposition 14, Theorems 4 and 3 and Proposition 13, ρ_2 is $\not\smile\!\!\bullet_s$-$\bar{Z}' \cup Act$-sigjust in the sense of Definition 22, so there is a $v^\dagger \in Tr$ with $v \not\smile\!\!\bullet_s v$ such that $\widehat{v^\dagger}$ occurs in ρ_2. Consequently, there is a transition $t^\dagger$ of the form $P'|v^\dagger$ or $u^\dagger|v^\dagger$ such that $\widehat{t^\dagger}$ occurs in ρ. By Proposition 12 $t \not\smile\!\!\bullet\, t^\dagger$. □

Corollary 6 An abstract path is B-just as defined in [7] iff it is B-just according to Definition 6.

Corollary 7 An abstract path is B-sigjust according to Definition 19 iff it is $\bar{B} \cap \mathscr{S}$-signalling as well as $B \cap Act$-just as defined in [7].

Proof: "Only if": Let ρ be B-sigjust according to Definition 19. By Corollary 5 it is also $B \cup Act$-sigjust, so by Proposition 14 it is $\bar{B} \cap \mathscr{S}$-signalling. By Proposition 15 ρ is $B \cap Act$-just.

"If": Let ρ be $\bar{B} \cap \mathscr{S}$-signalling as well as $B \cap Act$-just as defined in [7]. By Proposition 14 it is $B \cup Act$-sigjust. By Proposition 15 it is $B \cup \bar{\mathscr{S}}$-sigjust. So by Corollary 5 it is B-sigjust. □

In [17, 7, 3] a(n abstract) path is called *just* (without a predicate B) iff it is B-just for some $\mathscr{B}? \subseteq B \subseteq \mathscr{B}? \dot\cup \mathscr{C}_h \dot\cup \bar{\mathscr{C}}_h \dot\cup \mathscr{S}$, which is the case iff it is $\mathscr{B}? \dot\cup \mathscr{C}_h \dot\cup \bar{\mathscr{C}}_h \dot\cup \mathscr{S}$-just. This amounts to making a default choice for the set B of blocking actions, in which CCS handshake synchronisations c and $\bar{c}$ as well as broadcast receive and signal read actions can always be blocked by the environment (namely by withholding a synchronisation partner, no failing to broadcast or to signal). Using this definition it follows that an abstract path is just as defined in [18] and the current paper (using Definition 6 with any of the five concurrency relations $\smile\!\!\bullet$, $\smile\!\!\bullet_s$, $\smile\!\!\bullet'_s$, $\smile\!\!\bullet_c$ or $\smile\!\!\bullet'_c$) iff it is just as defined in [17], [7] or [3].

14 Conclusion

We advocate justness as a reasonable completeness criterion for formalising liveness properties when modelling distributed systems by means of transition systems. In [18] 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 [18] available to these languages. In fact, we provided five 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 [34]. Our concurrency relation $\smile\!\!\bullet$ follows the same computational intuition. However, in [34] 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 [32], event structures [35], or asynchronous transition systems [33, 2, 36]. 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.

References

[1] K.R. Apt, N. Francez & S. Katz (1988): *Appraising Fairness in Languages for Distributed Programming*. *Distributed Computing* 2(4), pp. 226–241, doi: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, doi: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, doi:10.1145/201019.201032.

[6] P. Degano, R. De Nicola & U. Montanari (1987): *CCS is an (Augmented) Contact Free C/E System*. In M.V. Zilli, editor: *Mathematical Models for the Semantics of Parallelism*, LNCS 280, Springer, pp. 144–165, doi:10.1007/3-540-18419-8_13.

[7] 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, doi:10.4204/EPTCS.255.2.

[8] 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, doi:10.1016/0167-6423(83)90017-5.

[9] 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, doi:10.1145/4904.4999.

[10] 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, doi:10.1007/978-3-642-28869-2_15.

[11] 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`.

[12] 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, doi:10.1007/978-3-319-23506-6_9. Available at `http://arxiv.org/abs/1509.05842`.

[13] R.J. van Glabbeek (2016): *Ensuring Liveness Properties of Distributed Systems (A Research Agenda)*. Position paper. Available at `https://arxiv.org/abs/1711.04240`.

[14] R.J. van Glabbeek, U. Goltz & J.-W. Schicke (2008): *On Synchronous and Asynchronous Interaction in Distributed Systems*. In E. Ochmański & J. Tyszkiewicz, editors: Proceedings 33rd International Symposium on *Mathematical Foundations of Computer Science* (MFCS 2008), Toruń, Poland, August 2008, LNCS 5162, Springer, pp. 16–35, doi:10.1007/978-3-540-85238-4_2.

[15] R.J. van Glabbeek, U. Goltz & J.-W. Schicke-Uffmann (2013): *On Characterising Distributability*. *Logical Methods in Computer Science* 9(3):17, doi:10.2168/LMCS-9(3:17)2013.

[16] R.J. van Glabbeek & P. Höfner (2015): *CCS: It's not fair!* *Acta Informatica* 52(2-3), pp. 175–205, doi:10.1007/s00236-015-0221-6.

[17] 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`.

[18] 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`.

[19] R.J. van Glabbeek & F.W. Vaandrager (1987): *Petri net models for algebraic theories of concurrency (extended abstract)*. In J.W.d. Bakker, A.J. Nijman & P.C. Treleaven, editors: Proceedings *PARLE, Parallel Architectures and Languages Europe*, Eindhoven, The Netherlands, June 1987, Vol. II: Parallel Languages, LNCS 259, Springer, pp. 224–242, doi:10.1007/3-540-17945-3_13.

[20] 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.

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

[22] L. Lamport (2000): *Fairness and hyperfairness*. *Distr. Comp.* 13(4), pp. 239–245, doi:10.1007/PL00008921.

[23] 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.

[24] R. Milner (1980): *A Calculus of Communicating Systems*. LNCS 92, Springer, doi:10.1007/3-540-10235-3.

[25] J. Misra (1988): *A Rebuttal of Dijkstra's Position on Fairness*. Available at `http://www.cs.utexas.edu/users/misra/Notes.dir/fairness.pdf`.

[26] J. Misra (2001): *A Discipline of Multiprogramming — Programming Theory for Distributed Applications*. Springer, doi:10.1007/978-1-4419-8528-6.

[27] E.-R. Olderog (1987): *Operational Petri net semantics for CCSP*. In G. Rozenberg, editor: *Advances in Petri Nets 1987*, covers the 7th European Workshop on *Applications and Theory of Petri Nets*, Oxford, UK, June 1986, LNCS 266, Springer, pp. 196–223, doi:10.1007/3-540-18086-9_27.

[28] E.-R. Olderog (1991): *Nets, Terms and Formulas: Three Views of Concurrent Processes and their Relationship*. *Cambridge Tracts in Theor. Comp. Sc.* 23, Cambridge University Press.

[29] S.S. Owicki & L. Lamport (1982): *Proving Liveness Properties of Concurrent Programs*. *ACM TOPLAS* 4(3), pp. 455–495, doi:10.1145/357172.357178.

[30] A. Pnueli (1977): *The Temporal Logic of Programs*. In: Proc. 18th Annual Symposium on *Foundations of Computer Science* (FOCS'77), IEEE, pp. 46–57, doi:10.1109/SFCS.1977.32.

[31] 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, doi:10.1007/3-540-53982-4_19.

[32] W. Reisig (2013): *Understanding Petri Nets — Modeling Techniques, Analysis Methods, Case Studies*. Springer, doi:10.1007/978-3-642-33278-4.

[33] M.W. Shields (1985): *Concurrent machines*. *The Computer Journal* 28(5), pp. 449–465, doi:10.1093/comjnl/28.5.449.

[34] E.W. Stark (1989): *Concurrent transition systems*. *Theoretical Computer Science* 64(3), pp. 221–269, doi:10.1016/0304-3975(89)90050-9.

[35] 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, doi:10.1007/3-540-17906-2_31.

[36] 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.