# Progress, Justness and Fairness

ROB VAN GLABBEEK and PETER HÖFNER, Data61, CSIRO and UNSW, Australia

Fairness assumptions are a valuable tool when reasoning about systems. In this paper, we classify several fairness properties found in the literature and argue that most of them are too restrictive for many applications. As an alternative we introduce the concept of justness.

CCS Concepts: • **Theory of computation** → *Process calculi*; *Operational semantics*; *Program reasoning*; • **Software and its engineering** → *Software system models*; *Correctness*; *Semantics*; • **General and reference** → Surveys and overviews;

Additional Key Words and Phrases: Fairness, Justness, Liveness, Labelled Transition Systems

## 1 INTRODUCTION

*Fairness properties* reduce the sets of infinite potential runs of systems. They are used in specification and verification.

As part of a system specification, a fairness property augments a core behavioural specification, given for instance in process algebra or pseudocode. The core behavioural specification typically generates a transition system, and as such determines a set of finite and infinite potential runs of the specified system. A fairness property disqualifies some of the infinite runs. These 'unfair' runs are unintended by the overall specification. It is then up to the implementation to ensure that only fair runs can occur. This typically involves design decisions from which the specification chooses to abstract.

As part of a verification, a fairness property models an assumption on the specified or implemented system. Fairness assumptions are normally used when verifying *liveness properties*, saying that something good will eventually happen. Without making the fairness assumption, the liveness property may not hold. When verifying a liveness property of a specification under a fairness assumption, this guarantees that the liveness property holds for any implementation that correctly captures the fairness requirement. When verifying a liveness property of an implemented system under a fairness assumption, the outcome is a conditional guarantee, namely that the liveness property holds as long as the system behaves fairly; when the system does not behave fairly, all bets are off. Such a conditional guarantee tells us that there are no design flaws in the system other than the—often necessary—reliance on fairness.

*Progress properties* reduce the sets of finite potential runs of systems. They play the same role as fairness assumptions do for infinite runs. In the verification of liveness properties, progress assumptions are essential. Although many interesting liveness properties hold without making fairness assumptions of any kind, no interesting liveness property holds without assuming progress. On the other hand, whereas a fairness assumption may be far-fetched, in the sense that run-of-the-mill implementations tend not to be fair, progress holds in almost any context, and run-of-the-mill implementations almost always satisfy the required progress properties. For this reason, progress assumptions are often made implicitly, without acknowledgement that the proven liveness property actually depends on the validity of such an assumption.

Authors' addresses: R. van Glabbeek, P. Höfner, DATA61, CSIRO, Locked Bag 6016, UNSW, Sydney, NSW 1466, Australia.

Publication date: October 2018.

One contribution of this paper is a taxonomy of progress and fairness properties found in the literature. These are ordered by their strength in ruling out potential runs of systems, and thereby their strength as an aid in proving liveness properties. Our classification includes the 'classical' notions of fairness, such as weak and strong fairness, fairness of actions, transitions, instructions and components, as well as extreme fairness. We also include *probabilistic fairness* (cf. [51]) and what we call *full fairness* [3, 8]. These are methods to obtain liveness properties, that—like fairness assumptions—do not require the liveness property to hold for each infinite potential run. They differ from 'classical' fairness assumptions in that no specific set of potential runs is ruled out.

Another contribution is the introduction of the concept of *justness*. Justness assumptions form a middle ground between progress and fairness assumptions. They rule out a collection of infinite runs, rather than finite ones, but based on a criterion that is a progress assumption in a distributed context taken to its logical conclusion. Justness is weaker than most fairness assumptions, in the sense that fewer runs are ruled out. It is more fundamental than fairness, in the sense that many liveness properties of realistic systems do not hold without assuming at least justness, whereas a fairness assumption would be overkill. As for progress, run-of-the-mill implementations almost always satisfy the required justness properties.

The paper is organised as follows. Section 2 briefly recapitulates transitions systems and the concept of liveness (properties). Section 3 recalls the assumption of progress, saying that a system will not get stuck without reason. We recapitulate the classical notions of weak and strong fairness in Section 4, and formulate them in the general setting of transition systems. In Section 5 we capture as many as twelve notions of fairness found in the literature as instances of a uniform definition, and order them in a hierarchy. We evaluate these notions of fairness against three criteria from the literature, and one new one, in Section 6. In Section 7 we extend our taxonomy by strong weak fairness, an intermediate concept between weak and strong fairness. We argue that such a notion is needed as in realistic scenarios weak fairness can be too weak, and strong fairness too strong. Section 8 recalls the formulations of weak and strong fairness in linear-time temporal logic. In Section 9 we present the strongest notion of fairness conceivable, and call it full fairness. Section 10 proves that on finite-state transition systems, strong fairness of transitions, one of the notions classified in Section 5, is as strong as full fairness. Sections 11 and 12 relate our taxonomy to two further concepts: probabilistic and extreme fairness. We show that probabilistic fairness is basically identical to strong fairness of transitions, whereas extreme fairness is the strongest possible notion of fairness that (unlike full fairness) fits the unifying definition proposed in Section 4. Section 12 concludes the comparison of fairness notions found in the literature. We then argue, in Section 13, that the careless use of any fairness assumption in verification tasks can yield false results. To compensate for this deficiency, we discuss a stronger version of progress, called justness, which can be used without qualms. Section 14 develops fairness notions based on justness, and adds them to the hierarchy of fairness notions. Section 15 discusses fairness of events, another notion of fairness from the literature, and shows that, although defined in a rather different way, it coincides with justness. Sections 2–15 deal with closed systems, having no run-time interactions with the environment. This allows us a simplified treatment that is closer to the existing literature. Section 16 generalises almost all definitions and results to reactive systems, interacting with their environments through synchronisation of actions. This makes the fairness notions more widely applicable. The penultimate Section 17 evaluates the notions of fairness discussed in the paper against the criteria for appraising fairness properties. We relate our hierarchy to earlier classifications of fairness assumptions found in the literature in Section 18, and close with a short discussion about future work.

## 2 TRANSITION SYSTEMS AND LIVENESS PROPERTIES

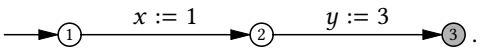
In order to formally define liveness as well as progress, justness and fairness properties, we take transition systems as our system model. Most other specification formalisms, such as process algebras, pseudocode, or Petri nets, have a straightforward interpretation in terms of transition systems. Using this, any liveness, progress, justness or fairness property defined on transition systems is also defined in terms of those specification formalisms.

Depending on the type of property, the transition systems need various augmentations. We introduce these as needed, starting with basic transition systems.

*Definition 2.1.* A *transition system* is a tuple $G = (S, Tr, source, target, I)$ with $S$ and $Tr$ sets (of *states* and *transitions*), $source, target : Tr \to S$, and $I \subseteq S$ (the *initial* states).

Later we will augment this definition with various attributes of transitions, such that different transitions between the same two states may have different attributes. It is for this reason that we do not simply introduce transitions as ordered pairs of states.

*Example 1.* The program $x := 1;\ y := 3$ is represented by the transition system




Here the short arrow without a source state indicates an initial state.

Progress and fairness assumptions are often made when verifying liveness properties. A *liveness property* says that "something [good] must happen" eventually [36]. One of the ways to formalise this in terms of transition systems is to postulate a set of good states $\mathscr{G} \subseteq S$. In Example 1, for instance, the good thing could be $y = 3$, so that $\mathscr{G}$ consists of state 3 only—indicated by shading. The liveness property given by $\mathscr{G}$ is now defined to hold iff each system run—a concept yet to be formalised—reaches a state $g \in \mathscr{G}$.

We now formalise a potential run of a system as a *rooted path* in its representation as a transition system.

*Definition 2.2.* A *path* in a transition system $G = (S, Tr, source, target, I)$ is an alternating sequence $\pi = s_0\, t_1\, s_1\, t_2\, s_2 \cdots$ of states and 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 $i$; it is *rooted* if $s_0 \in I$.

Example 1 has three potential runs, represented by the rooted paths $1\ t\ 2\ u\ 3$, $1\ t\ 2$ and 1, where $t$ and $u$ denote the two transitions corresponding to the instructions $x := 1$ and $y := 3$. The rooted path $1\ t\ 2$ models a run consisting of the execution of $x := 1$ only, without ever following this up with the instruction $y := 3$. In that run the system stagnates in state 2. Likewise, the rooted path 1 models a run where nothing ever happens.

Including these potential system runs as actual runs leads to a model of concurrency in which no interesting liveness property ever holds, e.g. the liveness property $\mathscr{G}$ for Example 1.[1]

Progress, justness and fairness assumptions are all used to exclude some of the rooted paths as system runs. Each of them can be formulated as, or gives rise to, a predicate on paths in transition systems. We call a path *progressing*, *just* or *fair*, if it meets the progress, justness or fairness assumption currently under consideration. We call it *complete* if it meets all progress, justness and fairness assumptions we currently impose, so that a rooted path is complete iff it models a run of the represented system that can actually occur.

[1] A *partial run* is an initial part of a system run. Naturally, the paths $1\ t\ 2$ and 1 always model partial runs in Example 1, even when we exclude them as runs. Partial runs play no role in this paper.

## 3 PROGRESS

If the rooted path 1 $t$ 2 of Example 1 is not excluded and models an actual system run, the system is able to stagnate in state 2, and the program does not satisfy the liveness property $\mathscr{G}$. The assumption of *progress* excludes this type of behaviour.[2] For closed systems, it says that

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

In other words, a run will never get stuck in a state with outgoing transitions.

*Definition 3.1.* A path in a transition system representing a closed system is *progressing* if either it is infinite or its last state is the source of no transition.

When assuming progress—one only considers progressing paths—the program of Example 1 satisfies $\mathscr{G}$.

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 [43]. A progress assumption is built into the temporal logics LTL [50], CTL [19] and CTL* [20], namely by disallowing states without outgoing transitions and evaluating temporal formulas by quantifying over infinite paths only.[3] In [33] the 'multiprogramming axiom' is a progress assumption, whereas in [1] progress is assumed as a 'fundamental liveness property'. In many other papers on fairness a progress assumption is made implicitly [2].

An assumption related to progress is that 'atomic actions always terminate' [46]. Here we have built this assumption into our definition of a path. To model runs in which the last transition never terminates, one could allow paths to end in a transition rather than a state.

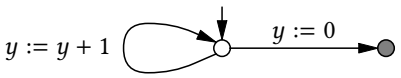
## 4 FAIRNESS

*Example 2.* Consider the following program [39].

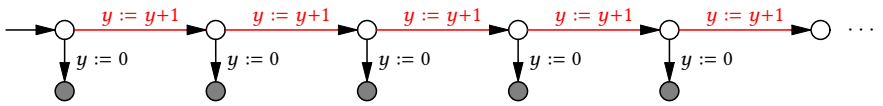
$$\textbf{initialise } y \textbf{ to } 1$$
$$y := 0 \;\|\; \textbf{while } (y > 0) \textbf{ do } y := y + 1 \textbf{ od}$$

This program runs two parallel threads: the first simply sets $y$ to 0, while the second one repeatedly increments $y$, as long as $y > 0$. We assume that in the second thread, the evaluation of the guard $y > 0$ and the assignment $y := y + 1$ happen in one atomic step, so that it is not possible that first the guard is evaluated (positively), then $y := 0$ is executed, and subsequently $y := y + 1$.

The transition system on the right describes the behaviour of this program. Since **initialise** $y$ **to** 1 is an initialisation step, it is not shown in the transition system. In the first state $y$ has a positive value, whereas in the second it is 0.




A different transition system describing the same program is shown below.




[2] Misra [44, 45] calls this the 'minimal progress assumption'. In [45] he uses 'progress' as a synonym for 'liveness'. In session types, 'progress' and 'global progress' are used as names of particular liveness properties [11]; this use has no relation with ours.

[3] 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 [16].

A liveness property is that eventually $y = 0$. It is formalised by letting $\mathscr{G}$ consist of the rightmost state in the first, or of all the bottom states in the second transition system. When assuming progress only, this liveness property does not hold; the only counterexample is the red coloured path. This path might be considered 'unfair', because the left-hand thread is never executed. A suitable fairness assumption rules out this path as a possible system run.

To formalise fairness we use transition systems $G = (S, Tr, source, target, I, \mathscr{T})$ that are augmented with a set $\mathscr{T} \subseteq \mathscr{P}(Tr)$ of *tasks* $T \subseteq Tr$, each being a set of transitions.

*Definition 4.1.* For a given transition system $G=(S, Tr, source, target, I, \mathscr{T})$, a task $T \in \mathscr{T}$ is *enabled* in a state $s \in S$ if there exists a transition $t \in T$ with $source(t) = s$. The task is said to be *perpetually enabled* on a path $\pi$ in $G$, if it is enabled in every state of $\pi$. It is *relentlessly enabled* on $\pi$, if each suffix of $\pi$ contains a state in which it is enabled.[4] It *occurs* in $\pi$ if $\pi$ contains a transition $t \in T$.

A path $\pi$ in $G$ is *weakly fair* if, for every suffix $\pi'$ of $\pi$, each task that is perpetually enabled on $\pi'$, occurs in $\pi'$.

A path $\pi$ in $G$ is *strongly fair* if, for every suffix $\pi'$ of $\pi$, each task that is relentlessly enabled on $\pi'$, occurs in $\pi'$.

The purpose of these properties is to rule out certain paths from consideration (the unfair ones), namely those with a suffix in which a task is enabled perpetually (or, in the strong case, relentlessly) yet never occurs. To avoid the quantification over suffixes, these properties can be reformulated:

A path $\pi$ in $G$ is *weakly fair* if each task that from some state onwards is perpetually enabled on $\pi$, occurs infinitely often in $\pi$.

A path $\pi$ in $G$ is *strongly fair* if each task that is relentlessly enabled on $\pi$, occurs infinitely often in $\pi$.

Clearly, any path that is strongly fair, is also weakly fair.

*Weak* [*strong*] *fairness* is the assumption that only weakly [strongly] fair rooted paths represent system runs. When applied to pseudocode, process algebra expressions or other system specifications that translate to transition systems, the concepts of weak and strong fairness are parametrised by the way to extract the collection $\mathscr{T}$ of tasks from the specification. For one such extraction, called *fairness of directions* in Section 5, weak and strong fairness were introduced in [2]. Weak fairness was introduced independently, under the name *justice*, in [39], and strong fairness, under the name *fairness*, in [30] and [39]. Strong fairness is called *compassion* in [41, 42].

Recall that the transition systems of Example 2, when merely assuming progress, do not satisfy the liveness property $\mathscr{G}$. To change that verdict, we can assume fairness, by defining a collection $\mathscr{T}$ of tasks. We can, for instance, declare two tasks: $T_1$ being the set of all transitions labelled $y := y + 1$ and $T_2$ the set of all transitions labelled $y := 0$. Now the red path in the second transition system, as well as all its prefixes, becomes unfair, since task $T_2$ is perpetually enabled, yet never occurs. Thus, in Example 2, $\mathscr{G}$ does hold when assuming weak fairness, with $\mathscr{T} = \{T_1, T_2\}$. In fact, it suffices to take $\mathscr{T} = \{T_2\}$. Assuming strong fairness gives the same result.

*Example 3.* Consider the following basic mutual exclusion protocol [39].

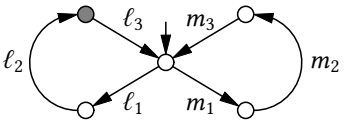
**initialise** $y$ **to** 1

| **while** $(true)$ | | **while** $(true)$ |
|---|---|---|
| $\ell_0$ **noncritical section** | $\parallel$ | $m_0$ **noncritical section** |
| $\ell_1$ **await**$(y > 0)\{y := y-1\}$ | | $m_1$ **await**$(y > 0)\{y := y-1\}$ |
| $\ell_2$ **critical section** | | $m_2$ **critical section** |
| $\ell_3$ $y := y+1$ | | $m_3$ $y := y+1$ |

[4] This is the case if the task is enabled in infinitely many states of $\pi$, in a state that occurs infinitely often in $\pi$, or in the last state of a finite $\pi$.

Here instructions $\ell_1$ and $m_1$ wait until $y > 0$ (possibly forever) and then atomically execute the instruction $y := y-1$, meaning without allowing the other process to change the value of $y$ between the evaluation $y > 0$ and the assignment $y := y - 1$. Dijkstra [17] abbreviates this instruction as P($y$); it models entering of a critical section, with $y$ as semaphore. Instructions $\ell_3$ and $m_3$ model leaving the critical section, and are abbreviated V($y$).

The induced transition system is depicted on the right, when ignoring lines $\ell_0$ and $m_0$, which play no significant role.




Let $\mathscr{G}$ contain the single marked state; so the good thing we hope to accomplish is the occurrence of $\ell_2$, saying that the left-hand process executed its critical section. Our tasks could be $L = \{\ell_1, \ell_2, \ell_3\}$ and $M = \{m_1, m_2, m_3\}$. Here weak fairness is insufficient to ensure $\mathscr{G}$, but under the assumption of strong fairness this liveness property holds.

In [30, 39], the assumptions of weak and strong fairness restrict the set of infinite runs only. These papers consider a path $\pi$ in $G$ weakly fair if either it is finite, or each task that from some state onwards is perpetually enabled on $\pi$, occurs infinitely often in $\pi$. Likewise, a path is considered strongly fair if either it is finite, or each task that is relentlessly enabled on $\pi$ occurs infinitely often in $\pi$. This makes it necessary to assume progress in addition to fairness when evaluating the validity of liveness properties.

In this paper we have dropped the exceptions for finite paths. The effect of this is that the progress assumption is included in weak or strong fairness, at least when $\bigcup_{T\in\mathscr{T}} T = Tr$. The latter condition can always be realised by adding the task $Tr$ to $\mathscr{T}$—this has no effect on the resulting notions of weak and strong fairness for infinite paths. Moreover, dropping the exceptions for finite paths poses no further restrictions on finite paths beyond progress.

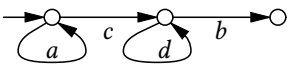
## 5 A TAXONOMY OF FAIRNESS PROPERTIES

For a given transition system, the concepts of strong and weak fairness are completely determined by the collection $\mathscr{T}$ of tasks. We now address the question how to obtain $\mathscr{T}$. The different notions of fairness found in the literature can be cast as different answers to that question.

As a first classification we distinguish local and global fairness properties.

A *global fairness* property is formulated for transition systems that are not naively equipped with a set $\mathscr{T}$, but may have some other auxiliary structure, inherited from the system specification (such as program code or a process algebra expression) that gave rise to that transition system. The set $\mathscr{T}$ of tasks, and thereby the concepts of strong and weak fairness, is then extracted in a systematic way from this auxiliary structure. An example of this is *fairness of actions*. Here transitions are labelled with *actions*, activities that happen when the transition is taken. In Example 2, the actions are $\{y := y+1,\ y := 0\}$, and in Example 3 $\{\ell_0, \ell_1, \ell_2, \ell_3, m_0, m_1, m_2, m_3\}$. Each action constitutes a task, consisting of all the transitions labelled with that action. Formally, $\mathscr{T} = \{T_a \mid a \in Act\}$, where $Act$ is the set of actions and $T_a = \{t \mid t \text{ is labelled by } a\}$. This is in fact the notion of fairness employed in Example 2, and using it for Example 3 would give rise to the same set of fair paths as the set of tasks employed there.

A *local fairness* property on the other hand is created in an ad hoc manner to add new information to a system specification. In the transition system on the right, for instance, we may want to make sure that incurring the cost $c$ is always followed by reaping the benefit $b$. This can be achieved by explicitly adding a fairness property to the specification, ruling out any path that gets stuck in the $d$-loop. This can be accomplished by declaring $\{b\}$ a task. At the same time, it may be acceptable that we never escape from the $a$-loop. Hence there is no need to declare a



task $\{c\}$. Whether we also declare tasks $\{a\}$ and $\{d\}$ makes no difference, provided that progress is assumed—as we do. A global fairness property seems inappropriate for this example, as it is hard to imagine why it would treat the $a$-loop with the $c$-exit differently from the $d$-loop with the $b$-exit.

Under local fairness, a system specification consists of two parts: a core part that generates a transition system, and a fairness component. This is the way fairness is incorporated in the specification language TLA$^+$ [38]. It also is the form of fairness we have chosen to apply in [22, Sect. 9] for the formal specification of the Ad hoc On-demand Distance Vector (AODV) protocol [48], a wireless mesh network routing protocol, using the process algebra AWN [21] for the first part and Linear-time Temporal Logic (LTL) [50] for the second. Under global fairness, a system specification consists of the core part only, generating a transition system with some auxiliary structure, together with just the choice of a notion of fairness, that extracts the set of tasks—or directly the set of fair paths—from this augmented transition system. Below we discuss some of these global notions of fairness, and cast them as a function from the appropriate auxiliary structure of transition systems to a set of tasks. Examples are given afterwards.

**Fairness of actions** (A) *Fairness of actions* is already defined above; the auxiliary structure it requires is a labelling $\ell : Tr \to Act$ of transitions by actions. This notion of fairness appears, for instance, in [37] and [5, Def. 3.44]. It has been introduced, under the name fairness of transitions, in [53, Def. 1].

**Fairness of transitions** (T) Taking the tasks to be the transitions, rather than their labels, is another way to define a collection $\mathscr{T}$ of tasks for a given transition system $G$. $\mathscr{T}$ consists of all singleton sets of transitions of $G$. In the first transition system of Example 2 $\mathscr{T}$ would consist of two elements, whereas the second transition system would yield an infinite set $\mathscr{T}$. This notion of fairness does not require any auxiliary structure on $G$. Fairness of transitions appears in [24, Page 127] under the name $\sigma$-fairness. A small variation, obtained by identifying all transitions that have the same source and target, originates from [53, Def. 2].

**Fairness of directions** (D) We need a transition system where each transition is labelled with the *instruction* or *direction* in the program text that gave rise to that transition. The auxiliary structure is a function $instr : Tr \to \mathcal{I}$, where $\mathcal{I}$ is a set (of *instructions* or *directions*). Now each instruction gives rise to a task: $\mathscr{T} := \{T_I \mid I \in \mathcal{I}\}$ with $T_I := \{t \in Tr \mid instr(t) = I\}$. This is the central notion of fairness in Francez [24]. If we would take $Act := \mathcal{I}$ and $\ell := instr$ then fairness of actions becomes the same as fairness of directions.

**Fairness of instructions** (I) When allowing instructions to synchronise, each transition is derived not from a single instruction, but from a nonempty set of them. So the auxiliary structure is a function $instr : Tr \to \mathscr{P}(\mathcal{I})$ with $instr(t) \neq \emptyset$ for all $t \in Tr$. The set $instr(t)$ has two elements for a handshake communication and N for N-way communication. There are two natural ways to generalise fairness of directions to this setting.

The first, which we call *fairness of instructions*, assigns a task to each instruction; a transition belongs to that task if that instruction contributed to it. So $\mathscr{T} := \{T_I \mid I \in \mathcal{I}\}$ with $T_I := \{t \in Tr \mid I \in instr(t)\}$. This type of fairness appears in [33] under the name guard fairness.

**Fairness of synchronisations** (Z) The second way assigns a task to each set of instructions; a transition belongs to that task if it is obtained through synchronisation of exactly that set of instructions: $\mathscr{T} := \{T_Z \mid Z \subseteq \mathcal{I}\}$ with $T_Z := \{t \in Tr \mid instr(t) = Z\}$. Here a local action, involving one component only, counts as a singleton synchronisation. This type of fairness appears in [33] under the name channel fairness, and in [1] under the name communication fairness.

**Fairness of components** (C) We need a transition system where each transition is labelled with the parallel *component* of the represented system that gave rise to that transition.[5] When allowing synchronisation, each transition stems from a nonempty set of components. So the auxiliary structure is a function $comp : Tr \to \mathscr{P}(\mathscr{C})$ with $comp(t) \neq \emptyset$ for all $t \in Tr$. Under *fairness of components* each component determines a task. A transition belongs to that task if that component contributed to it. So $\mathscr{T} := \{T_C \mid C \in \mathscr{C}\}$ with $T_C := \{t \in Tr \mid C \in comp(t)\}$. This is the type of fairness studied in [13, 15]; it also appears in [1, 33] under the name process fairness.

**Fairness of groups of components** (G) This is like fairness of components, except that each *set* of components forms a task, and a transition belongs to that task if it is obtained through synchronisation of exactly that set of components: $\mathscr{T} := \{T_G \mid G \subseteq \mathscr{C}\}$ with $T_G := \{t \in Tr \mid comp(t) = G\}$. This type of fairness appears in [1] under the name channel fairness when allowing only handshake communication, and under the name group fairness when allowing N-way communication.

Under each of the above notions of fairness, the first transition system of Example 2 has two tasks, each consisting of one transition. The transition system of Example 3 has six tasks of one transition each for fairness of actions, transitions, directions, instructions or synchronisations, but two tasks under fairness of components or groups of components. For Example 3, $\mathscr{G}$ holds in all cases when assuming strong fairness, but not when assuming weak fairness.

For transition systems derived from specification formalisms that do not feature synchronisation [2, 30, 39], fairness of instructions, of synchronisations, and of directions coincide. Likewise, fairness of groups of components coincides with fairness of components. The notion of fairness introduced in [2, 30] is fairness of directions, whereas [39] introduces fairness of components.

Let $xy$ be the fairness assumption with $x \in \{\mathrm{W}, \mathrm{S}\}$ indicating weak or strong and $y \in \{\mathrm{A}, \mathrm{T}, \mathrm{D}, \mathrm{I}, \mathrm{Z}, \mathrm{C}, \mathrm{G}\}$ one of the notions of fairness above.

For the forthcoming examples we use the process algebra CCS [43]. Given a set of *action names* $a$, $b, c, \ldots$, where each action $a$ has a complement $\bar{a}$, the set of CCS processes is built from the following constructs: $\mathbf{0}$ is the empty process (deadlock) offering no continuation; the process $a.E$ can perform an action $a$ and continue as $E$;[6] [7] the process $\tau.E$ can perform the internal action $\tau$ and continue as $E$; the process $E + F$ offers a choice and can proceed either as $E$ or as $F$; the parallel composition $E|F$ allows $E$ and $F$ to progress independently; moreover, in case $E$ can perform an action $a$ and $F$ can perform its complement $\bar{a}$ (or vice versa), the two processes can synchronise by together performing an internal step $\tau$; the process $E\backslash a$ modifies $E$ by inhibiting the actions $a$ and $\bar{a}$; and the relabelling $E[f]$ uses the function $f$ to rename action names in $E$. Infinite processes can be specified by process variables $X$ that are bound by *defining equations* $X \stackrel{def}{=} E$, where $E$ is a process expression constructed from the elements described above. CCS expressions generate transition systems where transitions are labelled with the actions a process performs—in case of synchronisation the label is $\tau$. The complete formal syntax and semantics is presented in Appendix A. There, we also define a fragment of CCS, and augment the transition systems generated by this fragment with the functions $\ell : Tr \to Act$, $instr : Tr \to \mathscr{P}(\mathscr{I})$ and $comp : Tr \to \mathscr{P}(\mathscr{C})$.

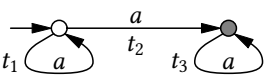
[5] A parallel component $c$ may itself have further parallel components $c_1$ and $c_2$, so that the collection of all parallel components in a given state of the system may be represented as a tree (with the entire system in that state as root). A transition that can be attributed to component $c_1$ thereby implicitly also belongs to $c$. We label such a transition with $c_1$ only, thus employing merely the leaves in such a tree. The number of components of a system is dynamic, as a single component may split into multiple components only after executing some instructions.

[6] Assuming progress, it *has* to perform the action.

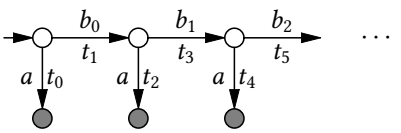
[7] We often abbreviate $a.\mathbf{0}$ by $a$.

In the following examples we use that fragment of CCS and always consider a liveness property $\mathscr{G}$, indicated by the shaded states in the depicted transition systems. Since $x$D-fairness coincides with both $x$I and $x$Z-fairness whenever it is defined, i.e., if $instr : Tr \to \mathscr{P}(\mathcal{I})$ maps to singleton sets only, we do not examine this notion separately. Similar to D we could have introduced a separate notion of fairness for the case that $comp(t)$ always yields a singleton set of components that specialises to C or G when it does not.

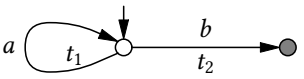
*Example 4.* Consider the process $a|X$ where $X \stackrel{def}{=} a.X$. Its transition system is depicted on the right, where the names $t_i$ differentiate transitions. The process specification contains two occurrences of the action $a$, which we will call $a_1$ and $a_2$. These will be our instructions: $instr : Tr \to \mathscr{P}(\mathcal{I})$ is given by $instr(t_1) = instr(t_3) = \{a_2\}$ and $instr(t_2) = \{a_1\}$. The process has three components, namely $a$ (named L), $X$ (named R), and the entire expression. Now $comp : Tr \to \mathscr{P}(\mathscr{C})$ is given by $comp(t_1) = comp(t_3) = \{\text{R}\}$ and $comp(t_2) = \{\text{L}\}$. Each of the notions of fairness I, Z, C and G yields two tasks: $\{t_1, t_3\}$ and $\{t_2\}$. On the only infinite path that violates liveness property $\mathscr{G}$, task $\{t_2\}$ is perpetually enabled, yet never occurs. Hence, when assuming $xy$-fairness with $x \in \{\text{W}, \text{S}\}$ and $y \in \{\text{I}, \text{Z}, \text{C}, \text{G}\}$, this path is unfair, and the liveness property $\mathscr{G}$ is satisfied. The same holds when assuming $x$T-fairness, even though this notion gives rise to three tasks. However, under the fairness assumption SA (and thus certainly under WA) property $\mathscr{G}$ is not satisfied, as all three transitions form one task.



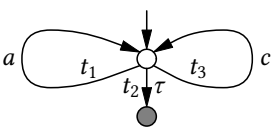

*Example 5.* Consider the process $a|X$ where $X \stackrel{def}{=} b_0.(X[f])$. Here $f$ is a relabelling operator with $f(b_i) = b_{i+1}$ for $i \geq 0$ and $f(a) = a$. When taking $y \in \{A, I, \text{Z}, C, G\}$ there is a task $\{t_0, t_2, t_4, \dots\}$, which is perpetually enabled until it occurs. So, under $xy$-fairness the process does satisfy $\mathscr{G}$. Yet under $x$T-fairness each transition is a separate task, and this property is not satisfied.




*Example 6.* Consider the process $X$ where $X \stackrel{def}{=} a.X + b$. Assuming $xy$-fairness with $y \in \{\text{A}, \text{T}, \text{I}, \text{Z}\}$ there is a task $\{t_2\}$, so that $\mathscr{G}$ is satisfied. Yet, since there is only one parallel component, SG- and SC-fairness allow an infinite path without transition $t_2$, so that $\mathscr{G}$ is not satisfied.




*Example 7.* Consider the process $(X|Y)\backslash b$ where $X \stackrel{def}{=} a.X + b$ and $Y \stackrel{def}{=} c.Y + \bar{b}$. The instructions are the action occurrences in the specification: $\mathcal{I} = \{a, b, c, \bar{b}\}$. We have $instr(t_1) = \{a\}$, $instr(t_2) = \{b, \bar{b}\}$ and $instr(t_3) = \{c\}$. There are three components, namely $X$ (called L), $Y$ (called R), and the entire expression$(X|Y)\backslash b$, with $comp(t_1) = \{\text{L}\}$, $comp(t_2) = \{\text{L}, \text{R}\}$ and $comp(t_3) = \{\text{R}\}$. So under fairness of synchronisations, or groups of components, there is a task $\{t_2\}$, making a path that never performs $t_2$ unfair. The same holds under fairness of actions or transitions, as well as instructions. Hence $\mathscr{G}$ is satisfied. However, under fairness of components, the tasks are $\{t_1, t_2\}$ and $\{t_2, t_3\}$ and $\mathscr{G}$ is not satisfied.




In the remainder of this section we establish a hierarchy of fairness properties; for this we introduce a partial order on fairness assumptions.

*Definition 5.1.* Fairness assumption F is *stronger* than fairness assumption H, denoted by $\text{H} \preceq \text{F}$, iff F rules out at least all paths that are ruled out by H.

Figure 1 classifies the above progress and fairness assumptions by strength, with arrows pointing towards the weaker assumptions. P stands for progress. Arrows derivable by reflexivity or transitivity of $\preceq$ are suppressed. The numbered arrows are valid only under the following assumptions:

(1) For each synchronisation $Z \subseteq \mathcal{I}$, and for each state $s$, there is at most one transition $t$ with $\mathit{instr}(t) = Z$ enabled in $s$.
(2) $\mathcal{I}$ is finite.
(3) There is a function $cp : \mathcal{I} \to \mathscr{C}$ such that $\mathit{comp}(t) = \{cp(I) \mid I \in \mathit{instr}(t)\}$ for all $t \in \mathit{Tr}$.

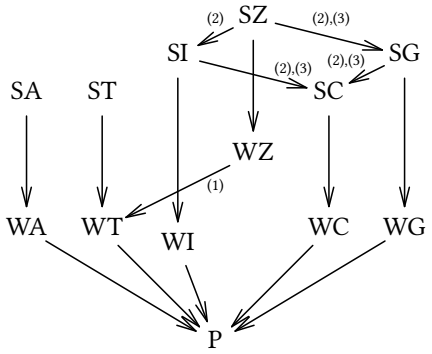
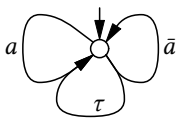


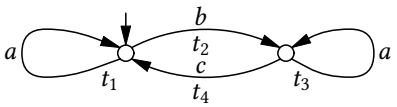

Fig. 1. A classification of progress and fairness

These assumptions hold for the transition systems derived from the fragment of CCS presented in Appendix A. All other arrows have full generality. The validity of the arrows is shown below (Props. 5.2–5.5). The absence of any further arrows follows from Examples 3–12: Example 3 separates weak from strong fairness; it shows that $\mathrm{S}y \not\preceq \mathrm{W}y'$ for all $y, y' \in \{\mathrm{A}, \mathrm{T}, \mathrm{I}, \mathrm{Z}, \mathrm{C}, \mathrm{G}\}$. Example 4 shows that there are no further arrows from SA, and thus none from WA. Example 5 shows that there are no further arrows from ST, and thus none from WT. Together with transitivity, these two conclusions also imply that there are no further arrows from P. Examples 6, 7 and the forthcoming Example 8 show that there are no further arrows from SG, SC and SI, respectively, and thus none from WC. Examples 9, 10 and 12 show that there are no further arrows into ST, WA and WC, respectively, and thus, using transitivity, no further from SZ. Example 11 shows that there are no further arrows from WZ. Further arrows from WI or WG are already ruled out by transitivity of $\preceq$. □

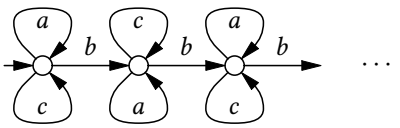
*Example 8.* Consider the process $X|Y$ where $X \stackrel{def}{=} a.X$ and $Y \stackrel{def}{=} \bar{a}.Y$. The infinite path $(a\bar{a})^\infty$ is $x$I-fair and $x$C-fair, but not $xy$-fair for $y \in \{\mathrm{A}, \mathrm{T}, \mathrm{Z}, \mathrm{G}\}$.



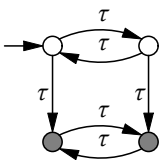

*Example 9.* Consider the process $X|Y$ where $X \stackrel{def}{=} a.X$ and $Y \stackrel{def}{=} b.c.Y$. Under ST-fairness, the path $(abc)^\infty$ is unfair, because task $\{t_3\}$ is infinitely often enabled, yet never taken. However, under $xy$-fairness for $y \in \{\mathrm{A}, \mathrm{I}, \mathrm{Z}, \mathrm{C}, \mathrm{G}\}$ there is no task $\{t_3\}$ and the path is fair.



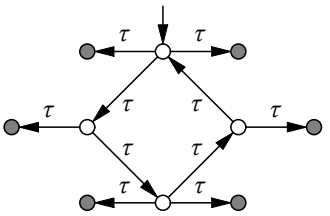

*Example 10.* Consider the process $X$ where $X \stackrel{def}{=} a.X + c.X + b.(X[f])$, using a relabelling operator with $f(a) = c$, $f(c) = a$ and $f(b) = b$. Under $x$A-fairness, the path $(ab)^\infty$ is unfair, because task $c$ is perpetually enabled, yet never taken. However, there are only three instructions (and one component), each of which gets its fair share. So under $xy$-fairness for $Y \in \{\mathrm{I}, \mathrm{Z}, \mathrm{C}, \mathrm{G}\}$ the path is fair. It is also $x$T fair.




*Example 11.* Consider the process $(a|X|Y)\backslash a$ where $X \stackrel{def}{=} \bar{a}.\bar{a}.X$ and $Y \stackrel{def}{=} a.Y$. Under W$y$-fairness with $y \in \{\mathrm{I}, \mathrm{C}, \mathrm{G}\}$ property $\mathscr{G}$ is satisfied, but under WZ- and WT and -fairness it is not, because each synchronisation is enabled merely in every other state, not perpetually. Since there is only a single action $\tau$, under $x$A-fairness property $\mathscr{G}$ is not satisfied.



*Example 12.* Consider the process $((a+c)|X|Y|Z)\backslash a\backslash b\backslash c\backslash d$ where $X \stackrel{def}{=} \bar{a}.\bar{b}.X$, $Y \stackrel{def}{=} \bar{c}.\bar{d}.Y$ and $Z \stackrel{def}{=} a.b.c.d.Z$. Under WC-fairness $\mathscr{G}$ is satisfied, but under W$y$-fairness with $y \in \{\mathrm{A}, \mathrm{T}, \mathrm{I}, \mathrm{Z}, \mathrm{G}\}$ it is not.




Proposition 5.2. $\mathrm{P} \preceq \mathrm{W}y \preceq \mathrm{S}y$ for $Y \in \{\mathrm{A}, \mathrm{T}, \mathrm{I}, \mathrm{Z}, \mathrm{C}, \mathrm{G}\}$.

Proof. That $\mathrm{W}y \preceq \mathrm{S}y$ follows from Definition 4.1. That $\mathrm{P} \preceq \mathrm{W}y$ follows since for each $y$ we have $\bigcup_{T \in \mathscr{T}} T = Tr$. □

Proposition 5.3. $\mathrm{WT} \preceq \mathrm{WZ}$.

Proof. Let $\pi$ be a WZ-fair path on which a transition $t$ is perpetually enabled. Then $T_{instr(t)}$, as defined on Page 7, is a WZ-task that is perpetually enabled on $\pi$. So a transition in $T_{instr(t)}$ must eventually occur, say in state $s$ on $\pi$. By (1) $t$ is the only transition from $T_{instr(t)}$ enabled in $s$. So $t$ will occur. □

Proposition 5.4. $\mathrm{SI} \preceq \mathrm{SZ}$.

Proof. By (2) there are only finitely many SI- and SZ-tasks. Moreover, each SI-task is the union of a collection of SZ-tasks. Let $\pi$ be a SZ-fair path on which a SI-task $T$ is infinitely often enabled (or in the last state). Then an SZ-task $T' \subseteq T$ must be be infinitely often enabled on $\pi$ (or in the last state). So $T'$, and hence $T$, will occur on $\pi$. □

Proposition 5.5. $\mathrm{SC} \preceq \mathrm{SI}$ and $\mathrm{SC} \preceq \mathrm{SG} \preceq \mathrm{SZ}$.

Proof. By (2) and (3) there are only finitely many SI-, SZ-, SC- and SG-tasks. Moreover, each SC-task is the union of a collection of SI-tasks; each SG-task is the union of a collection of SZ-tasks; and each SC-task is the union of a collection of SG-tasks. The rest of the proof proceeds as for Proposition 5.4. □

## 6 CRITERIA FOR EVALUATING NOTIONS OF FAIRNESS

In this section we review four criteria for evaluating fairness properties—the first three taken from [1]—and evaluate the notions of fairness presented so far. We treat the first criterion—feasibility—as prescriptive, in that we discard from further consideration any notion of fairness not meeting this criterion. The other criteria—equivalence robustness, liveness enhancement, and preservation under unfolding—are merely arguments in selecting a particular notion of fairness for an application. In Section 14 we propose a fifth criterion—*preservation under refinement of actions*—and argue that none of the weak notions of fairness satisfies that latter criterion.

### 6.1 Feasibility

The purpose of fairness properties is to reduce the set of infinite potential runs of systems, without altering the set of finite partial runs in any way. Hence a natural requirement is that any finite partial run can be extended to a fair run. This appraisal criteria on fairness properties has been proposed by Apt, Francez & Katz in [1] and called *feasibility*. It also appears in Lamport [37] under the name *machine closure*. We agree with Apt et al. and Lamport that this is a necessity for any sensible notion of fairness. By means of the following theorem we show that this criterion is satisfied by all fairness properties reviewed so far, when applied to the fragment of CCS from Appendix A.

Theorem 6.1. If, in a transition system, only countably many tasks are enabled in each state then the resulting notions of weak and strong fairness are feasible.

Proof. We present an algorithm for extending any given finite path $\pi_0$ into a fair path $\pi$. We build an $\mathbb{N} \times \mathbb{N}$-matrix with a column for the—to be constructed—prefixes $\pi_i$ of $\pi$, for $i \geq 0$. The columns $\pi_i$ will list the tasks enabled in the last state of $\pi_i$, leaving empty most slots if there are only finitely many. An entry in the matrix is either (still) empty, filled in with a task, 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 $\pi_0$ is known, and all columns of the matrix are empty. At each step $i \geq 0$ we proceed as follows:

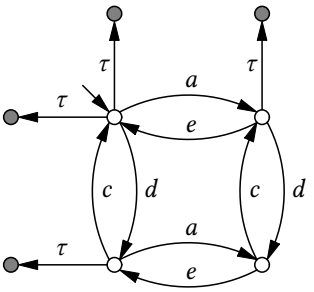
Since $\pi_i$ is known, we fill the $i$-th column by listing all enabled tasks. In case no task is enabled in the last state of $\pi_i$, the algorithm terminates, with output $\pi_i$. Otherwise, we take $j$ to be the smallest value such that entry $f(j) \in \mathbb{N} \times \mathbb{N}$ is already filled in, but not yet crossed out, and the task $T$ listed at $f(j)$ also occurs in column $i$ (i.e., is enabled in the last state of $\pi_i$). We now extend $\pi_i$ into $\pi_{i+1}$ by appending a transition from $T$ to it, while crossing out entry $f(j)$.

Obviously, $\pi_i$ is a prefix of $\pi_{i+1}$, for $i \geq 0$. The desired fair path $\pi$ is the limit of all the $\pi_i$. It is strongly fair (and thus weakly fair), because each task that is even once enabled will appear in the matrix, which acts like a priority queue. Since there are only finitely many tasks with a higher priority, at some point this task will be scheduled as soon as it occurs again. □

Since our fragment of CCS yields only finitely many instructions and components, there are only finitely many tasks according to the fairness notions I, Z, C and G. Moreover, there are only countably many transitions in the semantics of this fragment, and consequently only countably many action labels. Hence all fairness notions reviewed so far, applied to this fragment of CCS, are feasible.

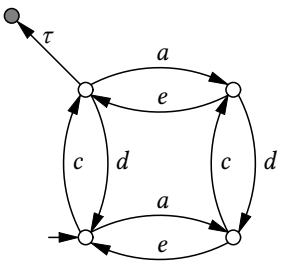
### 6.2 Equivalence robustness

*Example 13.* Consider the process $(X|Y|Z)\backslash b$ where $X \stackrel{def}{=} a.e.X + b$, $Y \stackrel{def}{=} \bar{b}$ and $Z \stackrel{def}{=} d.c.Z + b$. Under fairness of components, the four $\tau$-transitions form one task, as these are the transitions that synchronise with component $Y$. Thus, under WC-fairness, the path $(caed)^\infty$ is unfair, because that task is continuously enabled, yet never taken. However, the path $(cade)^\infty$ is fair, since this task is infinitely often not enabled. In *partial order semantics* [52] these two runs are considered *equivalent*: both model the parallel composition of runs $(ae)^\infty$ of component $X$ and $(cd)^\infty$ of component $Z$, with no causal dependencies between any actions occurring in the latter two runs. For this reason WC-fairness is is not robust under equivalence, as proposed in [1].




In general, Apt, Francez & Katz call a fairness notion *equivalence robust* if "for two infinite sequences which differ by a possibly infinite number of interchanges of independent actions (i.e. equivalent sequences), either both are fair according to the given definition, or both are unfair." [1]

*Example 14.* Consider the process $(X|Y)\backslash b$ where $X \stackrel{def}{=} a.e.X + b$ and $Y \stackrel{def}{=} \bar{b} + d.c.Y$. Under fairness of actions, transitions, instructions, synchronisations or groups of components, the $\tau$-transition forms a separate task. Thus, under Sy-fairness, for $y \in \{\mathrm{A}, \mathrm{T}, \mathrm{I}, \mathrm{Z}, \mathrm{G}\}$, the path $(cade)^\infty$ is unfair, because that task is infinitely often enabled, yet never taken. However, the path $(acde)^\infty$ is fair, since this task is never enabled. Yet these two runs are partial-order equivalent, so also Sy-fairness for $y \in \{\mathrm{A}, \mathrm{T}, \mathrm{I}, \mathrm{Z}, \mathrm{G}\}$ is not equivalence robust.



That WC-, SG- and SZ-fairness are not equivalence robust was established by Apt et al. [1]. There it is further shown that on a fragment of the process algebra CSP [31], featuring only 2-way synchronisation, SC-fairness is equivalence robust. We do not know if this result extends to our fragment of CCS, as it is in some ways less restrictive than the employed fragment of CSP. In the presence of $N$-way synchronisation (with $N>2$) SC-fairness is not equivalence robust [1]; this is shown by a variant of Example 14 that features an extra component $Z$, performing a single action that synchronises with both $b$ and $\bar{b}$. Apt et al. also show that, on their fragment of CSP, WG- and WZ-fairness are equivalence robust. We do not know if these results hold in full generality, or for our fragment of CCS. Example 13 shows that WA- and WI-fairness are not equivalence robust.

### 6.3 Liveness enhancement

A notion of fairness F is *liveness enhancing* if there exists a liveness property that holds for some system when assuming F, but not without assuming F. This criterion for appraising fairness notions stems from [1], focusing on the liveness property of termination; there, F is liveness enhancing iff there exists a system such that all its fair runs terminate, but not all its unfair runs.

When taken literary as worded above, the assumption of progress is liveness enhancing, since the program from Example 1 does satisfy the liveness property $\mathscr{G}$ when assuming progress, but not without assuming progress. Since all fairness properties reviewed here are stronger then progress, it follows that they are liveness enhancing as well.

The notion becomes more interesting in the presence of a background assumption, which is weaker than the fairness assumption being appraised. Assumption F then becomes liveness enhancing if there exist a liveness property that holds for some system when assuming F but not when merely making the background assumption. In [1] *minimal progress* is used as background assumption: 'Every process in a state with enabled *local* actions will eventually execute some action.' Here a process is what we call a component, and an action of that component is *local* if it it is not a synchronisation. This is a weaker form of the assumption of *justness*, to be introduced in Section 13.

In [1] it is established that WC-, WG- and WZ-fairness are not liveness enhancing on the employed fragment of CSP, whereas SC, SG- and SZ-fairness are. The negative results are due to the limited expressive power of that fragment of CSP; in our setting all notions of fairness reviewed so far are liveness enhancing w.r.t. the background assumption of minimal progress. This follows from Examples 6 and 11.

### 6.4 Preservation under unfolding

When using fairness of transitions in the first transition system of Example 2 the task $\mathscr{T}$ consists of two elements, and $\mathscr{G}$ is ensured, whereas the second transition system yields an infinite set $\mathscr{T}$, and $\mathscr{G}$ is not guaranteed, even though the second transition system is simply an unfolding of the first. This could be regarded as a drawback of fairness of transitions.

A possible criterion on fairness notions is that it ought to be possible to unfold a transition system $G$ into a tree—which yields a bijective correspondence between the paths in $G$ and those in its unfolding—without changing the set of fair paths.

This criterion holds for all notions of fairness considered so far, except for fairness of transitions. The reason is that any transition in the unfolding is inherited from a transition $t$ in $G$, and thereby inherits properties like $\ell(t)$, $\mathit{instr}(t)$ and $\mathit{comp}(t)$.

In [53], where fairness of transitions is employed, care is taken to select a particular transition system for any given program. The one chosen has exactly one state for each combination of control locations, and values of variables. So in terms of Example 2 it is the variant of the second transition system in which all final states are collapsed.

## 7 STRONG WEAK FAIRNESS

We present an example of a system for which none of the above fairness notions is appropriate: weak fairness is too weak and strong fairness too strong. We consider this example not to be a corner case, but rather an illustration of the typical way fairness is applied to real systems. Consequently, this example places serious doubts on the notions of fairness reviewed so far. Failing to find a concept of fairness in the literature that completely fits the requirements of this example in the literature, we propose the concept of *strong weak fairness*, which is the logical combination of two of the original fairness concepts contemplated in the literature.

*Example 15.* Let $X$ be a clerk that has to serve customers that queue at three windows of an office. Serving a customer is a multi-step process. During such a process the clerk is not able to serve any other customer. A CCS specification of the clerk's behaviour is $X \stackrel{def}{=} c_1.e.X + c_2.e.X + c_3.e.X$. Here $c_i$, for $i = 1, 2, 3$, is the action of starting to serve a customer waiting at window $i$, and $e$ the remainder of the serve effort. If this were a section on reactive systems we could continue the example by evaluating this clerk in any possible context. But since we promised (at the end of the introduction) to deal, for the time being, with closed systems only, we close the system by specifying the three rows of customers. Let $Y_1 \stackrel{def}{=} \bar{c}_1.Y_1$ model an never ending sequence of customers queuing at window 1, with $\bar{c}_i$ being the action of being served at window $i$. Likewise $Y_2 \stackrel{def}{=} \bar{c}_2.Y_2$, but at window 3 we place only two customers that may take the action $g$ of going home when not being served for a long time, and the action $r$ of returning to window 3 later on:

$$Y_3^{2,2} \stackrel{def}{=} \bar{c}_3.Y_3^{1,1} + g.Y_3^{2,1} \qquad Y_3^{2,1} \stackrel{def}{=} \bar{c}_3.Y_3^{1,0} + g.Y_3^{2,0} + r.Y^{2,2} \qquad Y_3^{2,0} \stackrel{def}{=} r.Y^{2,1}$$
$$Y_3^{1,1} \stackrel{def}{=} \bar{c}_3.Y_3^{0,0} + g.Y_3^{1,0} \qquad Y_3^{1,0} \stackrel{def}{=} r.Y_3^{1,1} \qquad Y_3^{0,0} \stackrel{def}{=} \mathbf{0}$$

Here $Y_3^{i,j}$ is the state in which there are $i$ potential customers for window 3, of which $j$ are currently queueing. Our progress requirement says that if both customers have gone home, eventually one of them will return to window 3. The overall system is $(X|Y_1|Y_2|Y_3^{2,2})\backslash c_1\backslash c_2\backslash c_3$. An interesting liveness property $\mathscr{G}$ is that any customer, including the ones waiting at window 3, eventually gets served. Two weaker properties are $\mathscr{G}_2$: eventually a customer waiting at window 2 will get served, and $\mathscr{G}_3$: eventually either a customer waiting at window 3 will get served, or both customers that were waiting at window 3 are at home.

As manager of the office, we stipulate that the clerk should treat all windows fairly, without imposing a particular scheduling strategy. Which fairness property do we really want to impose?

We can exclude fairness of actions here, because, due to synchronisation into $\tau$, no action differentiates between serving different customers: neither WA nor SA does ensure $\mathscr{G}_2$.

None of the weak fairness properties presented so far is strong enough. For suppose that the clerk would only ever serve window 1. Then the customers waiting at windows 2 and 3 represent tasks that will never be scheduled. However, these tasks are not perpetually enabled, because each time the clerk is between actions $c_1$ and $e$, they are not enabled. For this reason, no weak fairness property is violated in this scenario. In short: Wy for $y \in \{\mathrm{T}, \mathrm{I}, \mathrm{Z}, \mathrm{C}, \mathrm{G}\}$ does not even ensure $\mathscr{G}_2$.

Thus it appears that as manager we have to ask the clerk to be strongly fair: each component (window) that is relentlessly enabled (by having a customer waiting there), should eventually be served. In particular, this fairness requirement (FR) does not allow skipping window 3 altogether, even if its last customer goes home and returns infinitely often. Technically, this amounts to imposing Sy for some $y \in \{\mathrm{T}, \mathrm{I}, \mathrm{Z}, \mathrm{G}\}$. SC is too weak; it does not even ensure $\mathscr{G}_3$, because when the queue-length at window 3 alternates between 2 and 1, that component is deemed to get its fair share.

Suppose that the clerk implements FR by a *round robin* scheduling strategy. The clerk first tries to serve a customer from window 1. If there is no customer at window 1, or after interacting with a client from window 1, she serves a customer from window 2 if there is one, and so on, returning to window 1 after dealing with window 3.

At first sight, this appears to be an entirely fair scheduling strategy. However, it does not satisfy the strong fairness requirement FR: suppose that the customers from window 3 spend most of their time at home, with only one of them showing up at window 3 for a short time every day, and always timed as unfortunate as possible, i.e., arriving right after the clerk started to serve window 1, and departing right before she finishes serving window 2. In that case these customers will never be served, and hence FR is not met. Waiting at each window for the next customer to arrive does not work either as the clerk will starve after 2 rounds at window 3.

One possible reaction to this conclusion is to reject round robin scheduling; an alternative is the "queueing" algorithm of Park [47] (Section 3.3.2): "at each stage, the earliest clause [window] with true guard [a waiting customer] is obeyed [served], and moved to the end of the queue." However, another quite reasonable reaction is that round-robin scheduling is good enough, and that the waiting/going home scheduling of customer 3 contemplated above is so restrictive that as manager we will not be concerned about customer 3 never being scheduled. This customer has only himself to blame. This reaction calls for a fairness requirement that is stronger than weak fairness but weaker than FR, in that it ensures liveness properties $\mathscr{G}_2$ and $\mathscr{G}_3$, but not $\mathscr{G}$.

Park [47] essentially rejects strong fairness because of the complexity of its implementation: "The problem of implementing strong fairness is disquieting. It is not clear that there is any algorithm which is essentially more efficient than the queueing algorithm of 3.3.2. [...] If the problem is essentially as complex as this, then strong fairness in this form would seem an undesirable ingredient of language specification". In [23], Fischer& Paterson confirm the state of affairs as feared by Park: any scheduling algorithm for the clerk above requires at least $n!$ storage states—where $n$ is the number of customers—whereas a weakly fair scheduling algorithm requires only $n$ storage states.

### 7.1 Response to insistence and persistence

One of the very first formalisations of fairness found in the literature occur in [25], where two notions of fairness were proposed. *Response to insistence* "states that a permanent holding of a condition or a request $p$ will eventually force a response $q$." *Response to persistence* states "that the infinitely repeating occurrence of the condition $p$ will eventually cause $q$."

The later notions of weak and strong fairness [2, 30, 39], reviewed in Sections 4 and 5, can be seen as instantiations of the notions from [25], namely by taking the condition $p$ to mean that a task is enabled, and $q$ that it actually occurs. However, the fairness notions of [25] also allow different instantiations, more compatible with Example 15. We can take $p_i$ to be the condition that a customer is waiting at window $i$, and $q_i$ the response that such a customer is served. The resulting notion of response to insistence is much stronger than weak fairness, for it disallows the scenario were the clerk only serves window 1. In fact, it ensures liveness properties $\mathscr{G}_2$ and $\mathscr{G}_3$. Yet, it is correctly implemented by a round robin scheduling strategy, and does not guarantee the unrealistic liveness property $\mathscr{G}$. Qua complexity of its implementation it sides with weak fairness, as there is no need for more than $n$ storage states; see the last paragraph of the previous section.

In [22] we use fairness properties for the formalisation of temporal properties of routing protocols. The situation there is fairly similar to Example 15. The fairness properties $(P_1)$, $(P_2)$ and $(P_3)$ that we propose there are local fairness properties rather than global ones (see Section 5) and instances of response to insistence.

Response to insistence may fail the criterion of feasibility. Without loss of generality, we can identify the response $q$ of response to insistence with the task $T$ of Definition 4.1. Response to insistence says that the permanent holding of the request $p$ is enough to ensure $q$, even if $q$ is, from some point onwards, never enabled. This can happen in a variant of Example 15 where the clerk is given the possibility of serving a single customer for an unbounded amount of time. Such a scenario clearly runs contrary to the intentions of [25]. A more useful fairness notion, probably closer to the intentions of [25], is

> *if a condition or request $p$ holds perpetually from some point onwards, and a response $q$ is infinitely often enabled, then $q$ must occur.*

This is the logical combination of strong fairness and response to insistence. We propose to call it *strong weak fairness*.

Footnote 60 in [22] shows that the instances of response to insistence used in that paper do meet the criterion of feasibility, and hence amount to instances of strong weak fairness.

For the same reason as above, response to persistence does not meet the criterion of feasibility either. To make it more useful, one can combine it with strong fairness in the same fashion as above. However, the resulting notion does not offer much that is not offered by strong fairness alone.

### 7.2 Strong weak fairness of instructions

We finish this section by isolating a particular form for strong weak fairness by choosing the relevant conditions $p$ and responses $q$. This is similar to the instantiation of the notions of strong and weak fairness from Section 4 by choosing the relevant set of tasks in Section 5. Here we only deal with *strong weak fairness of instructions* (SWI), by following the choices made in Section 5 for strong and weak fairness of instructions.

To provide the right setting, we make the following two assumptions on our transition system, all of which are satisfied on the studied fragment of CCS. By property (3) in Section 5, each instruction $I$ belongs to a component $cp(I)$. Given a state $P$ of the overall system, a given component $C \in \mathscr{C}$, if active at all, is in a state corresponding to a subexpression of $P$, denoted by $P_C$.

We say that an instruction $I$ is *requested* in state $P$ if a transition $t$ with $I \in comp(t)$ is enabled in $P_{cp(I)}$. The instruction is *enabled* in $P$ if a transition $t$ with $I \in comp(t)$ is enabled in $P$. The instruction *occurs* in a path $\pi$ if $\pi$ contains a transition $t$ with $I \in comp(t)$. We recall that a path $\pi$ is strongly fair according to fairness of instructions if, for every suffix $\pi'$ of $\pi$, each task that is relentlessly enabled on $\pi'$, occurs in $\pi'$.

*Definition 7.1.* A path $\pi$ in a transition system is *strongly weakly fair* if, for every suffix $\pi'$ of $\pi$, each instruction that is perpetually requested and relentlessly enabled on $\pi'$, occurs in $\pi'$.

Applied to Example 15, instruction $\bar{c}_2$ is requested in each state of the system, and enabled in each state where none of the instructions $e$ is enabled. Instruction $\bar{c}_2$ being requested signifies that a customer is waiting at window 2. Thus, on any rooted path $\pi$ instruction $\bar{c}_2$ is perpetually requested and relentlessly enabled, even though it is not perpetually enabled. Consequently, by strong weak fairness of instructions, but not by weak fairness of instructions, a synchronisation involving $\bar{c}_2$ will eventually occur, meaning that window 2 is served. So $\mathscr{G}_2$ is ensured. Likewise, $\mathscr{G}_3$ is ensured, but $\mathscr{G}$ is not, for the moment that both customers of window 3 are at home, the condition of $\bar{c}_2$ being requested is interrupted.

### 7.3 Adding strong weak fairness of instructions to our taxonomy

Whenever an instruction $I$ is enabled in a state $P$, it is also requested, at least on our fragment of CCS. For other transition systems we take this as an assumption necessary only for the classification

of SWI-fairness. Additionally, we assume that if $I$ is requested in state $P$ and $u$ is a transition from $P$ to $Q$ such that $cp(I) \notin comp(u)$, then $I$ is still requested in $Q$. This assumption also holds on our fragment of CCS.

By the first assumption above, SWI is a stronger notion of fairness then WI. Strictness follows by Example 15, using $\mathscr{G}_2$. By definition, SWI is weaker than SI. Strictness follows by Example 15, using $\mathscr{G}$. Interestingly, SWI is stronger than SC, even though SWI has the characteristics of a weak fairness notion.

Proposition 7.2. SC $\preceq$ SWI.

Proof. Suppose $T_C$, which was given as $\{t \in Tr \mid C \in comp(t)\}$ for a component $C$, is relentlessly enabled on an SWI-fair path $\pi$, yet never occurs in $\pi$. Since each SC-task is the union of finitely many SI-tasks (using (2) and (3); see Proposition 5.5), there must be an instruction $I$ with $cp(I) = C$ that is relentlessly enabled on $\pi$. Let $P$ be a state on $\pi$ in which $I$ is enabled. Then, by the first assumption above, $I$ is also requested in state $P$. But since no transition $t$ with $C \in comp(t)$ is ever scheduled, by the second assumption above, $I$ remains requested for that component in all subsequent states, and hence is perpetually requested. By SWI-fairness, $I$, and thereby $T_C$, will occur in $\pi$, contradicting the assumption. □

Hence the position of SWI in our hierarchy of fairness notions is as indicated in Figure 2. There can not be any further arrows into or out of SWI, because this would give rise to new arrows between the other notions.

## 8 LINEAR-TIME TEMPORAL LOGIC

Fairness properties can be expressed concisely in Linear-time Temporal Logic (LTL). LTL formulas [50] are build from a set AP of *atomic propositions* $p, q, \ldots$ by means of unary operators **F** and **G**[8] and the connectives of propositional logic. They are interpreted on total transition systems $G = (S, Tr, source, target, I)$ as in Definition 2.1 that are equipped with a validity relation $\models \subseteq S \times \mathrm{AP}$ that tells which atomic propositions hold in which states. Here a transition system is *total* iff each state has an outgoing transition. It is inductively defined when an infinite path $\pi$ in $G$ *satisfies* an LTL formula $\varphi$—notation $\pi \models \varphi$:

- $\pi = s_0 t_1 s_1 t_2 s_2 \ldots \models p$ iff $s_0 \models p$;
- $\pi \models \varphi \wedge \psi$ iff $\pi \models \varphi$ and $\pi \models \psi$;
- $\pi \models \neg\varphi$ iff $\pi \not\models \varphi$;
- $\pi \models \mathbf{F}\psi$ iff there is a suffix $\pi'$ of $\pi$ with $\pi' \models \phi$;
- $\pi \models \mathbf{G}\psi$ iff for each suffix $\pi'$ of $\pi$ one has $\pi' \models \phi$.

The transition system $G$ satisfies $\varphi$—notation $G \models \varphi$—if $\pi \models \varphi$ for each infinite rooted path in $G$.

This definition of satisfaction has a progress assumption built-in, for totality implies that a path is infinite iff it is progressing. LTL can be applied to transition systems that are not necessarily total by simply replacing "infinite" by "progressing" in the above definition (cf. Section 3).

We apply LTL to transition systems $G$ were some relevant propositions $q$ are defined on the *transitions* rather then the states of $G$. This requires a conversion of $G$ into a related transition system $G'$ where those propositions are shifted to the states, and hence can be used as the atomic propositions of LTL. Many suitable conversions appear in the literature [16, 29]. For our purposes, the following partial unfolding suffices:

*Definition 8.1.* Given an augmented transition system $G = (S, Tr, source, target, I, \mathrm{AP}_S, \models_S, \mathrm{AP}_{Tr}, \models_{Tr})$ where $\models_S \subseteq S \times \mathrm{AP}_S$ supplies the validity of state-based atomic propositions, and $\models_{Tr} \subseteq Tr \times \mathrm{AP}_{Tr}$ supplies the validity of transition-based atomic propositions.

[8] In later work operators **X** and **U** are added to the syntax of LTL; these are not needed here.

Let $G' := (S', Tr', source', target', I, \mathrm{AP}, \models)$, where $S' := I \cup Tr$, $Tr' := \{(source(t), t) \mid t \in Tr \wedge source(t) \in I\} \cup \{(t,u) \in Tr \times Tr \mid target(t) = source(u)\}$ $source'(t,u) = t$, $target'(t,u) = u$, $\mathrm{AP} = \mathrm{AP}_S \uplus \mathrm{AP}_{Tr}$ and $t \models p$ iff $p \in \mathrm{AP}_S \wedge ((t \in I \wedge t \models_S p) \vee (t \in Tr \wedge target(t) \models_S p))$, or $p \in \mathrm{AP}_{Tr} \wedge t \in Tr \wedge t \models p$.

This construction simply makes a copy of each state for each transition that enters it, so that a proposition pertaining to a transition can be shifted to the target of that transition, without running into ambiguity when a state is the target of multiple transitions. Instead of calling a state $(t, target(t))$, we simply denote it by the transition $t$. We say that $G \models \varphi$ iff $G' \models \varphi$, for $\varphi$ an LTL formula that may use state-based as well as transition-based atomic propositions.

In [25], response to insistence was expressed in LTL as $\mathbf{G}(\mathbf{G}p \Rightarrow \mathbf{F}q)$. Here $\mathbf{G}p \Rightarrow \mathbf{F}q$ "states that a permanent holding of a condition or a request $p$ will eventually force a response $q$." "Sometimes, the response $q$ frees the requester from being frozen at the requesting state. In this case once $q$ becomes true, $p$ ceases to hold, apparently falsifying the hypothesis $\mathbf{G}p$. This difficulty is only interpretational and we can write instead the logically equivalent condition $\neg\mathbf{G}(p \wedge \neg q)$" [25]. The outermost $\mathbf{G}$ requires this condition to hold for all future behaviours.

Using that $\neg\mathbf{G}(p \wedge \neg q)$ is equivalent to $\mathbf{F}\neg p \vee \mathbf{F}q$, and $\mathbf{G}(\mathbf{F}\neg p \vee \mathbf{F}q)$ is equivalent to $\mathbf{GF}\neg p \vee \mathbf{GF}q$, it follows that response to insistence can equivalently be expressed as $\mathbf{FG}p \Rightarrow \mathbf{GF}q$, saying that if from some point onwards condition $p$ holds perpetually, then response $q$ will occur infinitely often.

Weak fairness, as formalised in Section 4, can be expressed by the same LTL formula, but taking $q$ to be the occurrence of a task $T$, a condition that holds when performing any transition from $T$, and $p$ the condition that $T$ is enabled, which holds for any state with an outgoing transition from $T$. The whole notion of weak fairness (when given a collection $\mathscr{T}$ of tasks) is then the conjunction, for all $T \in \mathscr{T}$, of the formulas $\mathbf{G}(\mathbf{G}(enabled(T)) \Rightarrow \mathbf{F}(occurs(T)))$.

Likewise, *Response to Persistence* is expressed as $\mathbf{G}(\mathbf{GF}p \Rightarrow \mathbf{F}q)$, where $\mathbf{GF}p \Rightarrow \mathbf{F}q$ is logically equivalent to $\neg\mathbf{G}(\mathbf{F}p \wedge \neg q)$ [25]. *Response to Persistence* can equivalently be expressed as $\mathbf{GF}p \Rightarrow \mathbf{GF}q$, saying that if condition $p$ holds infinitely often then response $q$ occurs infinitely often. Again, strong fairness is expressed in the same way.

## 9 FULL FAIRNESS

In [53, Def. 3] a path $\pi = s_0\, t_1\, s_1\, t_2\, s_2 \ldots$ is regarded as unfair if there exists a predicate $\mathscr{H}$ on the set of states such that a state in $\mathscr{H}$ is reachable from each state $s_i$, yet $s_i \in \mathscr{H}$ for only finitely many $i$. It is not precisely defined which sets of states $\mathscr{H}$ can be seen as predicates; if we allow any set $\mathscr{H}$ then the resulting notion of fairness is not feasible.

*Example 16.* Let $\pi$ be any infinite rooted path of the following program.

$$\textbf{initialise } (x,y) \textbf{ to } (0,0)$$
$$\textbf{while } (true) \textbf{ do } (x,y) := (x{+}1, 0) \textbf{ od} \parallel \textbf{while } (true) \textbf{ do } (x,y) := (x{+}1, 1) \textbf{ od}$$

Then $\pi$ can be encoded as a function $f_\pi : \mathbb{N} \to \{0,1\}$, expressing $y$ in terms of the current value of $x$. Let $\mathscr{H} := \{(x, 1{-}f_\pi(x)) \mid x \in \mathbb{N}\}$. $\mathscr{H}$ is reachable from each state of $\pi$, but never reached. Consequently $\pi$ is unfair. This holds for all infinite rooted paths. So no path is fair.

Since we impose feasibility as a necessary requirement for a notion of fairness, this rules out the form of fairness contemplated above. Nevertheless, this idea can be used to support liveness properties $\mathscr{G}$, and in this form we propose to call it *full fairness*.

*Definition 9.1.* [29] A liveness property $\mathscr{G}$, modelled as a set of states in a transition system $G$, is an *AGEF property* iff (a state of) $\mathscr{G}$ is reachable from every state $s$ that is reachable from an initial state of $G$.[9]

*Definition 9.2.* A liveness property $\mathscr{G}$ holds under the assumption of *full fairness* (Fu) iff $\mathscr{G}$ is an AGEF property.

Full fairness is not a fairness assumption in the sense of Section 4. It does not eliminate a particular set of paths (the unfair ones) from consideration. Nevertheless, like fairness assumptions, it increases the set of liveness properties that hold, and as such can be compared with fairness assumptions.

Obviously no feasible fairness property can be strong enough to ensure a liveness property that is not AGEF, for it is always possible to follow a finite path to a state from which it is hopeless to still satisfy $\mathscr{G}$. Hence full fairness, as a tool for validating liveness properties, is the strongest notion of fairness conceivable.

## 10 STRONG FAIRNESS OF TRANSITIONS

This section provides an exact characterisation of the class of liveness properties that hold under strong fairness of transitions (ST). It follows that on finite-state transition systems, strong fairness of transitions is as strong as full fairness.

Theorem 10.1. A liveness property $\mathscr{G}$, modelled as a set of states, holds in a transition system under the assumption of strong fairness of transitions, iff $\mathscr{G}$ is an AGEF property and each infinite rooted path that does not visit (a state of) $\mathscr{G}$ has a loop.

Proof. As pointed out above, if $\mathscr{G}$ is not an AGEF-property, it holds under no fairness assumption. Let $\pi$ be a loop-free infinite rooted path that does not visit (a state of) $\mathscr{G}$. Since each state of $\pi$ is visited only finitely often, there is no transition that is enabled relentlessly on $\pi$. Thus, $\pi$ is ST-fair, and consequently $\mathscr{G}$ does not hold under strong fairness of transitions.

Now let $\mathscr{G}$ be an AGEF-property and each infinite rooted path that does not visit $\mathscr{G}$ has a loop. Let $\pi$ be an ST-fair rooted path. It remains to be shown that $\pi$ visits $\mathscr{G}$. Suppose it does not.

Define the *distance* (from $\mathscr{G}$) of a state $s$ to be the length of the shortest path from $s$ to a state of $\mathscr{G}$. Since $\mathscr{G}$ is AGEF, any state $s$ has a finite distance. For each $n \geq 0$ let $S_n$ be set of states with distance $n$. Each state in $S_n$, with $n > 0$, must have an outgoing transition to a state in $S_{n-1}$. If $\pi$ is finite, its last state is in $S_n$ for some $n > 0$, and a transition (to a state in $S_{n-1}$) must be enabled in its last state, contradicting the ST-fairness of $\pi$. So assume $\pi$ is infinite. Then, there is a state on $\pi$ that is visited infinitely often. Let $n \in \mathbb{N}$ be the smallest number such that a state $s$ in $S_n$ is visited infinitely often. Since $\pi$ does not visit $\mathscr{G}$, $n > 0$. Consider a transition $t$ from $s$ to a state in $S_{n-1}$. This transition is relentlessly enabled on $\pi$, and thus must occur in $\pi$ infinitely often. It follows that a state in $S_{n-1}$ is visited infinitely often, yielding a contradiction. □

Corollary 10.2. A liveness property $\mathscr{G}$ holds in a finite-state transition system under the assumption strong fairness of transitions iff $\mathscr{G}$ is an AGEF property.

Thus, on finite-state systems strong fairness of transitions is the strongest notion of fairness conceivable. That it is strictly stronger than the other notions of Section 5 is illustrated below.

*Example 17.* Consider the process $X|c.X$, where $X \stackrel{def}{=} a.b.c.X$. Let $\mathscr{G}$ consist of those states where both processes are in the same local state. If we consider the finite-state transition system

[9] The name stems from the corresponding CTL-formula.

that lists all 9 possible configurations of the system then $\mathscr{G}$ is an AGEF property, and thus will be reached under ST-fairness. Clearly, $\mathscr{G}$ is not guaranteed under SZ- and SA-fairness.

Example 5 shows that on infinite-state systems strong fairness of transitions is strictly weaker than full fairness.

## 11 PROBABILISTIC FAIRNESS

When regarding nondeterministic choices as probabilistic choices with unknown probability distributions, one could say that a liveness property *holds under the assumption of probabilistic fairness* if it holds with probability 1, regardless of these probability distributions. Formally, a *probabilistic interpretation* of a transition system $G$[10] is a function $p : Tr \to (0, 1]$ assigning a positive probability to each transition, such that for each state $s$ that has an outgoing transition the probabilities of all transitions with source $s$ sum up to 1. Given a probabilistic interpretation, the probability of (following) a finite path is the product of the probabilities of the transitions in that path. The probability of reaching a set of states $\mathscr{G}$ is the sum over all finite paths from the initial state to a state in $\mathscr{G}$, not passing through any states of $\mathscr{G}$ on the way, of the probability of that path. Now $\mathscr{G}$ holds under probabilistic fairness (Pr) iff this probability is 1 for *each* probabilistic interpretation. Probabilistic fairness is a meaningful concept only for countably-branching transition systems, since it is impossible to assign each transition a positive such that their sum is 1 in case a state has uncountably many outgoing transitions.

A form of probabilistic fairness was contemplated in [51]. Similar to full fairness, probabilistic fairness is not a fairness assumption in the sense of Section 4, as it does not eliminate a particular set of paths. We show that it is equally strong as strong fairness of transitions.

Theorem 11.1. On countably-branching transition systems a liveness property holds under probabilistic fairness iff it holds under strong fairness of transitions.

Proof. We use the characterisation of strong fairness of transitions from Theorem 10.1. If a liveness property $\mathscr{G}$ is not AGEF, it surely does not hold under probabilistic fairness, since there exists a state, reachable with probability $> 0$, from which $\mathscr{G}$ cannot be reached. Furthermore, if there exists an infinite loop-free rooted path that does not visit $\mathscr{G}$, then the probabilities of the transitions leaving that path can be chosen in such a way that the probability of remaining on this path forever, and thus not reaching $\mathscr{G}$, is positive.

Now let $\mathscr{G}$ be an AGEF-property that does not hold under probabilistic fairness. It remains to be shown that there exists an infinite loop-free path that does not visit $\mathscr{G}$.

Allocate probabilities to the transitions in such a way that the probability of reaching $\mathscr{G}$ from the initial state is less than 1. For each state $s$, let $p_s$ be the probability of reaching $\mathscr{G}$ from that state. Since $p_s$ is the weighted average of the values $p_{s'}$ for all states $s'$ reachable from $s$ in one step, we have that if $s$ has a successor state $s'$ with $p_{s'} > p_s$, then it also has a successor state $s''$ with $p_{s''} < p_s$.

Claim: If, for a reachable state $s$, $p_s < 1$ then there is a path $st_1s_1t_2s_2 \ldots s_nt_{n+1}s_{n+1}$ with $n \geq 0$ such that $p_{s_i} = p_s$ for $i = 1, \ldots, n$ and $p_{s_{n+1}} < p_s$.

Proof of claim: Since $\mathscr{G}$ is an AGEF property, there is a path $\pi = st_1s_1t_2s_2 \ldots s_k$ from $s$ to a state $s_k \in \mathscr{G}$. Clearly $p_{s_k} = 1$. Let $n \in \mathbb{N}$ be the smallest index for which $p_{s_{n+1}} \neq p_{s_n}$. In case $p_{s_{n+1}} < p_{s_n}$ we are done. Otherwise, there must be a successor state $s''$ of $s_n$ with $p_{s''} < p_{s_n}$, and we are also done. ∎

[10] In this section we restrict attention to transition systems with exactly one initial state. An arbitrary transition system can be transformed into one with this property by adding a fresh initial state with an outgoing transition to each of the old initial states. This restriction saves us the effort of also declaring a probability distribution over the initial states.

Application of the claim: By assumption, $p_{s_0} < 1$, for $s_0$ the initial state. Using the claim, there exists an infinite path $\pi = s_0 t_1 s_1 t_2 s_2 \ldots$ such that $p_{s_i} \leq p_s$ for all $i \in \mathbb{N}$, and $\forall i \in \mathbb{N}.\ \exists j > i.\ p_{s_j} < p_{s_i}$. Clearly, this path does not visit a state of $\mathscr{G}$, and no state $s$ can occur infinitely often in $\pi$. After cutting out loops, $\pi$ is still infinite, and moreover loop-free, which finishes the proof. □

## 12 EXTREME FAIRNESS

In [51], Pnueli proposed the strongest possible notion of fairness that fits in the format of Definition 4.1. A first idea (not Pnueli's) to define the strongest notion might be to admit *any* set of transitions as a task. However, the resulting notion of fairness would fail the criterion of feasibility.

*Example 18.* The transition system for the program of Example 16 can be depicted as an infinite binary tree. For any path $\pi$ in this tree, let $T_\pi$ be the set of all transitions that do not occur in $\pi$. On $\pi$, this task is perpetually enabled, yet does not occur. It follows that $\pi$ is unfair. As this holds for any path $\pi$, no path is fair, and the resulting notion of fairness is infeasible.

Avoiding this type of example, Pnueli defined the notion of *extreme fairness* by admitting any task (i.e. any predicate on the set of transitions) that is definable in first-order logic. This makes sense when we presuppose any formal language for defining predicates on transitions. Pnueli is not particularly explicit about the syntax and semantics of such a language, which is justified because one only needs to know that such formalisms can only generate only countably many formulas, and thus only countably many tasks. With Theorem 6.1 it follows that extreme fairness is feasible.

Assuming a sufficiently expressive language for specifying tasks, any task according to fairness of actions, transitions, instructions, etc., is also a task according to extreme fairness. Hence the strong version of extreme fairness (Ext)—the one considered in [51]—sits at the top of the hierarchy of Figure 1, but still below full fairness.

*Example 19.* Under full fairness the following program will surely terminate.

$$\textbf{initialise } y \textbf{ to } 1$$
$$\textbf{while } (y > 0) \textbf{ do } y := y - 1 \textbf{ od} \;\|\; \textbf{while } (y > 0) \textbf{ do } y := y + 1 \textbf{ od}$$

Yet, under any other notion of fairness, including probabilistic fairness, it is possible that $y$ slowly but surely increases (in the sense that no finite value is repeated infinitely often). Fairness (even extreme fairness) can cause a decrease of $y$ once in a while, but is not strong enough to counter the upwards trend.

Pnueli invented extreme fairness as a true fairness notion that closely approximates a variant of probabilistic fairness [51]. His claim that for finite-state programs extreme fairness and probabilistic fairness coincides, now follows from Corollary 10.2 and Theorem 11.1.

## 13 JUSTNESS

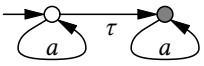
Fairness assumptions can be a crucial ingredient in the verification of liveness properties of real systems. A typical example is the verification of a communication protocol that ensures that a stream of messages is relayed correctly, without loss or reordering, from a sender to a receiver, while using an unreliable communication channel. The *alternating bit protocol* [6, 40], an example of such a communication protocol, works by means of acknowledgements, and resending of messages for which no acknowledgement is received.

To prove correctness of such protocols one has to make the indispensable fairness assumption that attempts to transmit a message over the unreliable channel will not fail in perpetuity. Without a fairness assumption, no such protocol can be found correct, and one misses a chance to check

the protocol logic against possible flaws that have nothing to do with a perpetual failure of the communication channel. For this reason, fairness assumptions are made in many process-algebraic verification platforms, and are deeply ingrained in their methodology [3, 8]. The same can be said for other techniques, such as automata-based approaches, or temporal logic.

Fairness assumptions, however, need to be chosen with care, since they can lead to false conclusions.

*Example 20.* Consider the process $P := (X|\bar{b})\backslash b$ where $X \stackrel{def}{=} a.X + b.X$. The process $X$ models the behaviour of relentless Alice, who keeps calling her friends, either Bob ($b$) or any other ($a$). In parallel, $\bar{b}$ models Bob, impatiently waiting for a call by Alice. Our liveness property $\mathscr{G}$ is to achieve a connection between Alice and Bob.

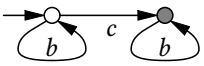



Under each of the notions of fairness of Sections 5–12, $P$ satisfies $\mathscr{G}$. Yet, it is perfectly reasonable that the connection is never established: Alice could never be in the mood of calling Bob; maybe she totally broke up with him.

The above example is not an anomaly; it describes the default situation. A fairness assumption says, in essence, that if one tries something often enough, one will eventually succeed. There is nothing in our understanding of the physical universe that support such a belief. Fairness assumptions are justified in exceptional situations, the verification of the alternating bit protocol being a good example. However, by default they are unwarranted.

In the remainder of this section we investigate what is needed to verify realistic liveness properties when not making any fairness assumptions.

*Example 21.* Consider the process $P := X|c$ where $X \stackrel{def}{=} b.X$. Here $X$ is an (endless) series of calls between Alice and Bob in London (who finally speak to each other again), while $c$ is the action by Cateline of eating a croissant in Paris. Clearly, there is no interaction between Cateline's desire and Alice's and Bob's chats. The question arises whether she is guaranteed to succeed in eating her breakfast. Using progress, but not fairness, the answer is negative, for there is a progressing rooted path consisting of $b$-transitions only. Nevertheless, as nothing stops Cateline from making progress, in reality $c$ will occur.




We therefore propose a strong progress assumption, called *justness*:

> *Once a transition is enabled that stems from a set of parallel components, one (or more) of these components will eventually partake in a transition.* (J)

In Example 21, justness would guarantee that Cateline will have breakfast.

To formalise justness, we employ a binary relation $\frown$ between transitions, with $t \frown u$ meaning that the transition $u$ interferes with $t$, in the sense that it affects a resource that is needed by $t$. In particular, $t$ cannot run concurrently with $u$.

*Definition 13.1.* A path $\pi$ in a transition system representing a closed system is *just* if for each transition $t$ with $s := source(t) \in \pi$, a transition $u$ occurs in $\pi$ past the occurrence of $s$, such that $t \frown u$.

When thinking of the resources alluded to in the above explanation of $\frown$ as components, $t \frown u$ means that a component affected by the execution of $u$ is a necessary participant in the execution of $t$. The relation $\frown$ can then be expressed in terms of two functions $npc, afc : Tr \to \mathscr{P}(\mathscr{C})$ telling for each transition $t$ which components are necessary participants in the execution of $t$, and which

are affected by the execution of $t$, respectively. Then

$$t \smile\!\!\bullet\, u \quad \text{iff} \quad npc(t) \cap afc(u) = \emptyset\,.$$

We assume the following crucial property for our transition systems:

> If $t, u \in Tr$ with $source(t) = source(u)$ and $npc(t) \cap afc(u) = \emptyset$ then there is a transition $v \in Tr$ with $source(v) = target(u)$ and $npc(v) = npc(t)$. (#)

The underlying idea is that if a transition $u$ occurs that has no influence on components in $npc(t)$, then the internal state of those components is unchanged, so any synchronisation between these components that was possible before $u$ occurred, is still possible afterwards.

We also assume that $npc(t) \cap afc(t) \neq \emptyset$ for all $t \in Tr$, that is, $\not\smile\!\!\bullet$ is reflexive. Intuitively, this means that a least one component that is required by $t$ is also affected when taking the transition $t$. Thus, one way to satisfy the justness requirement is by executing $t$.

Justness is thus precisely formalised for any transition system arising from a process algebra, Petri net, or other specification formalism for which functions *afc* and *npc*, satisfying (#) and reflexivity of $\not\smile\!\!\bullet$, are defined. Many specification formalisms satisfy $afc(t) = npc(t)$ for all $t \in Tr$. In such cases we write *comp* for *afc* and *npc*, and $\smile$ for $\smile\!\!\bullet$, this relation then being symmetric.

This applies in particular to the fragment of CCS studied in this paper; in Appendix A we present a definition of *comp* which satisfies (#) and reflexivity of $\not\smile$. For a 'local' transition *comp* returns a singleton set, whereas for a CCS-synchronisation yielding a $\tau$-transition two components will be present. We only include minimal components (cf. Footnote 5). The transition $(P|a.Q)\backslash c\,|\bar{a}.R \xrightarrow{\tau} (P|Q)\backslash c\,|R$ for example stems from a synchronisation of $a$ and $\bar{a}$ in the components $a.Q$ and $\bar{a}.R$. Since $a.Q$ is a subcomponent of $(P|a.Q)\backslash c$, it can be argued that the component $(P|a.Q)\backslash c$ also partakes in this transition $t$. Yet, we do not include this component in $comp(t)$.

The justness assumption is fundamentally different from a fairness assumption: rather than assuming that some condition holds perpetually, or infinitely often, we merely assume it to hold once. In this regard, justness is similar to progress. Furthermore, justness (J) implies progress: we have P $\preceq$ J.

Assuming justness ensures $\mathscr{G}$ in Examples 1, 4, 5 and 21, but not in Examples 6, 7, 11, 12 and 20.

In Examples 2 and 3 there are three components: L(eft), R(ight) and an implicit memory (M), where the value of $y$ is stored. The analysis of Example 2 depends on the underlying memory model. Clearly, any write to $y$ affects the memory component, i.e. $\text{M} \in afc(y := 0)$ and $\text{M} \in afc(r)$, where $r$ is the atomic instruction "**while** $(y > 0)$ **do** $y := y + 1$ **od**". Moreover, since $r$ involves reading from memory, M is a necessary participant in this transition: $\text{M} \in afc(r)$. If we would assume that a write always happens immediately, regardless of the state of the memory, one could argue that the memory is not a necessary participant in the transition $y := 0$. This would make justness a sufficient assumption to ensure $\mathscr{G}$. A more plausible memory model, however, might be that no two components can successfully write to the same memory at the same time. For one component to write, it needs to "get hold of" the memory, and while writing, no other component can get hold of it. Under this model $\text{M} \in npc(y := 0)$, so that justness is not a sufficient assumption to ensure $\mathscr{G}$.

In Example 3, by the above reasoning, we have $afc(\ell_1) = npc(\ell_1) = afc(m_1) = npc(m_1) = \{\text{L}, \text{M}\}$. Since $npc(\ell_1) \cap afc(m_1) \neq \emptyset$, we have $\ell_1 \not\smile\!\!\bullet\, m_1$, i.e. the occurrence of $m_1$ can disable $\ell_1$. Justness does not guarantee that $\ell_1$ will ever occur, so assuming justness is insufficient to ensure $\mathscr{G}$.

The difference between *npc* and *afc* shows up in process algebras featuring broadcast communication, such as the one of [28]. If $t$ models a broadcast transition, the only necessary participant is the component that performs the transmitting part of this synchronisation. However, all components that receive the broadcast are in $afc(t)$. In this setting, one may expect that necessary participants in a synchronisation are always affected ($npc(t) \subseteq afc(t)$). However, in a process algebra with signals

[7, 18] we find transitions $t$ with $npc(t) \not\subseteq afc(t)$. Let $t$ model the action of a driver seeing a red traffic light. This transition has two necessary participants: the driver and the traffic light. However, only the driver is affected by this transition; the traffic light is not.

In applications where $npc \neq afc$, two variants of fairness of components and fairness of groups of components can be distinguished, namely by taking either $npc$ or $afc$ to be the function $comp : Tr \to \mathscr{P}(\mathscr{C})$ assumed in Section 5. Since the specification formalisms dealing with fairness found in the literature, with the exception of our own work [18, 22, 28], did not give rise to a distinction between necessary and affected participants in a synchronisation, in our treatment of fairness we assume that $npc = afc$ ($= comp$), leaving a treatment of the general case for future work.

Special cases of the justness assumption abound in the literature. Kuiper & de Roever [33] for instance assume 'fundamental liveness', attributed to Owicki & Lamport [46], 'ensuring that if a process is continuously able to proceed, it eventually will.' Here 'process' is what we call 'component'. This is formalised for specific commands of their process specification language, and in each case appears to be an instance of what we call justness. However, there is no formalisation of justness as one coherent concept. Likewise, the various liveness axioms in [46], starting from Section 5, can all be seen as special cases of justness. Apt, Francez & Katz [1] assume 'minimal progress', also attributed to [46]: 'Every process in a state with enabled *local* actions will eventually execute some action.' This is the special case of Definition 13.1 where $npc(t)$ is a singleton.

In the setting of Petri nets the (necessary and affected) participants in a transition $t$ could be taken to be the preplaces of $t$. Hence $t \not\frown u$ means that the transition system transitions $t$ and $u$ stem from Petri net transitions that have a common preplace. Here justness says that if a Petri net transition $t$ is enabled, eventually a transition will fire that takes away a token from one of the preplaces of $t$. In Petri-net theory this is in fact a common assumption [54], which is often made without giving it a specific name.

In [22, 28] we introduced justness in the setting of two specific process algebras. Both have features that resist a correct translation into Petri nets, and hence our justness assumptions cannot be explained in terms of traditional assumptions on Petri nets. The treatment above is more general, not tied to any specific process algebra.

In [18, 27] we argue that a justness assumption is essential for the validity of any mutual exclusion protocol. It appears that in the correctness arguments of famous mutual exclusion protocols appearing in the literature [35, 49] the relevant justness assumption is taken for granted, without explicit acknowledgement that the correctness of the protocol rests on the validity of this assumption. We expect that such use of justness is widespread.

## 14 UNINTERRUPTED JUSTICE

In Example 20 weak fairness (of any kind) is enough to ensure $\mathscr{G}$. Now, let the action $a$ of calling anybody except Bob consist of two subactions $a_1.a_2$, yielding the specification $X' \stackrel{def}{=} a_1.a_2.X' + b.X'$. Here $a_1$ could for instance be the act of looking up a telephone number in the list of contacts, and $a_2$ the actual calling attempt. Substituting $a_1.a_2$ for $a$ is a case of *action refinement* [26], and represents a chance in the level of abstraction at which actions are modelled. After this refinement, the system no longer satisfies $\mathscr{G}$ when assuming weak fairness (only). Hence weak fairness is not preserved under refinement of actions.

When the above is seen as an argument against using weak fairness, two alternatives come to mind. One is to use strong fairness instead; the other is to tighten the definition of weak fairness in such a way that that a rooted path is deemed fair only if it remains weakly fair after any action refinement.

To this end we revisit the meaning of "perpetually enabled" in Definition 4.1. This way of defining weak fairness stems from Lehmann, Pnueli & Stavi [39], who used the words "justice" for weak fairness and "continuously" for perpetual. Instead of merely requiring that a task is enabled in every state, towards a more literal interpretation of "continuously" we additionally require that it remains enabled between each pair of successive states, e.g. during the execution of the transitions.

Formalising this requires us to define enabledness during a transition. If $source(t) = source(u)$ then $t \frown u$ tells us that the possible occurrence of $t$ is unaffected by the possible occurrence of $u$. In other words, $t \not\frown u$ indicates that the occurrence of $u$ ends or interrupts the enabledness of $t$, whereas $t \frown u$ indicates that $t$ remains enabled during the execution of $u$.

*Definition 14.1.* For a transition system $G = (S, Tr, source, target, I, \mathscr{T})$, equipped with a relation $\frown \subseteq Tr \times Tr$, a task $T \in \mathscr{T}$ is *enabled* during a transition $u \in Tr$ if $t \frown u$ for some $t \in T$ with $source(t) = source(u)$. It is said to be *continuously enabled* on a path $\pi$ in $G$, if it is enabled in every state *and transition* of $\pi$. A path $\pi$ in $G$ is J*-fair* if, for every suffix $\pi'$ of $\pi$, each task that is continuously enabled on $\pi'$, occurs in $\pi'$.

As before, we can reformulate the property to avoid the quantification over suffixes: A path $\pi$ in $G$ is J-fair if each task that from some state onwards is continuously enabled on $\pi$, occurs infinitely often in $\pi$.

Letting $x$ range over $\{J, W, S\}$ extends the taxonomy of Section 5 by six new entries, some of which coincide (JZ, JG and J). Figure 2 shows the full hierarchy, including the entries Ext, Pr and Fu from Sections 9–12 and J from Section 13. The arrows are valid under the assumptions (1)–(3) from

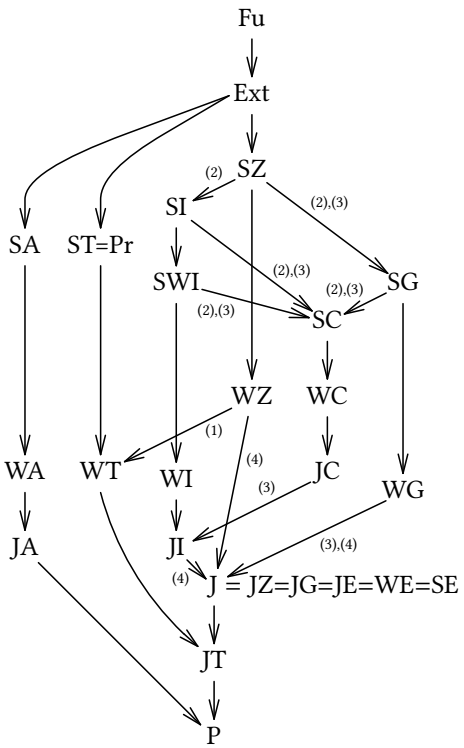



Fig. 2. A classification of progress, justness and fairness assumptions

Section 5, and the assumption that $npc = afc = comp$, together with

(4) if $t \smile u$ with $source(t) = source(u)$, then $\exists v \in Tr$ with $source(v) = target(u)$ and $instr(v) = instr(t)$.

This assumption says that if a transition $t$ is enabled that does not depend on any component that is affected by $u$, and $u$ occurs, then afterwards a variant of $t$, stemming from the same instructions, is still possible. This assumption strengthens (#) and holds for the transition systems derived from the fragment of CCS presented in Appendix A.

Clearly, $\mathrm{P} \preceq \mathrm{J}y \preceq \mathrm{W}y \preceq \mathrm{S}y$ for $y \in \{\mathrm{A}, \mathrm{T}, \mathrm{I}, \mathrm{Z}, \mathrm{C}, \mathrm{G}\}$.

Proposition 14.2. $\mathrm{JI} \preceq \mathrm{JC}$.

Proof. W.l.o.g. let $\pi$ be a JC-fair path on which a task $T_I$, for a given instruction $I \in \mathcal{I}$, is continuously enabled. Then, using assumption (3), on $\pi$ component $cp(I)$ is continuously enabled. Hence, the task $T_{cp(I)}$ will occur in $\pi$. Let $t \in T_{cp(I)}$ be a transition in $\pi$. Then $cp(I) \in comp(t) \cap comp(u)$ for any $u \in T_I$, so that $t \not\frown u$. Thus, no transition from $T_I$ is enabled during the execution of $t$, contradicting the assumption that $T_I$ is continuously enabled on $\pi$. Hence $\pi$ is JI-fair. □

Proposition 14.3. $\mathrm{J} \preceq \mathrm{J}y$ for $y \in \{\mathrm{I}, \mathrm{Z}, \mathrm{C}, \mathrm{G}\}$.

Proof. W.l.o.g. let $\pi$ be a JI-fair path such that a transition $t$ is enabled in its first state. Let $I \in instr(t)$. Assume that $\pi$ contains no transition $u$ with $comp(t) \cap comp(u) \neq \emptyset$. Then, using (4), for each transition $u$ occurring in $\pi$, a transition from $T_I$ is enabled during and right after $u$. So, by JI-fairness, the task $T_I$ must occur in $\pi$, but this contradicts the assumption. It follows that $\pi$ is just. The proof for $y \in \{\mathrm{Z}, \mathrm{C}, \mathrm{G}\}$ goes likewise, for $y \in \{\mathrm{C}, \mathrm{G}\}$ also using (3). □

Proposition 14.4. $\mathrm{J}y \preceq \mathrm{J}$ for $y \in \{\mathrm{T}, \mathrm{Z}, \mathrm{G}\}$.

Proof. W.l.o.g. let $\pi$ be a just path on which a task $T_G$, for a given $G \subseteq \mathscr{C}$, is continuously enabled. Let $t \in T_G$ be enabled in the first state of $\pi$.

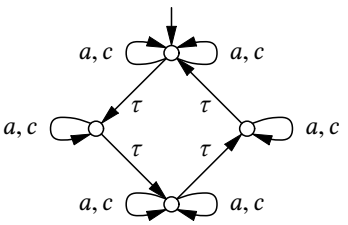
Then, by justness, $\pi$ contains a transition $u$ with $\emptyset \neq comp(u) \cap comp(t) = comp(u) \cap G$. It follows that no transition $v \in T_G$ is enabled during $u$, contradicting the assumption.

Now assume a task $T_Z$, for a given $Z \subseteq \mathcal{I}$, is continuously enabled on $\pi$. Using (3), let $G = \{cp(I) \mid I \in Z\}$. Then also $T_G$ is continuously enabled on $\pi$, which led to a contradiction above.

Finally assume a transition $t$ is continuously enabled on $\pi$. Let $G = comp(t)$. Then also $T_G$ is continuously enabled on $\pi$, which leads to a contradiction. □

The absence of any further arrows follows from Examples 3–20: Example 20 separates J-fairness from weak fairness; it shows that $\mathrm{W}y \not\preceq \mathrm{J}y'$ (and hence $\mathrm{S}y \not\preceq \mathrm{J}y'$) for all $y, y' \in \{\mathrm{A}, \mathrm{T}, \mathrm{I}, \mathrm{Z}, \mathrm{C}, \mathrm{G}\}$. Example 4, in which JT-fairness suffices to ensure $\mathscr{G}$, shows that there are no further arrows from SA, and thus none from WA and JA. Example 5, in which justness or JA-fairness suffices to ensure $\mathscr{G}$, shows that there are no further arrows from ST, and thus none from WT, JT and P. Example 12, in which JC-fairness suffices to ensure $\mathscr{G}$, shows that there are no further arrows into JC. So, using transitivity, and remembering the results from Sections 5–9, it suffices to show that $\mathrm{JA} \not\preceq \mathrm{SZ}$, $\mathrm{JI} \not\preceq \mathrm{WZ}$ and $\mathrm{JI} \not\preceq \mathrm{WG}$. This will be demonstrated by the following two examples.

*Example 22.* Consider the process $(X|Y|Z)\backslash b\backslash d$ where $X \stackrel{def}{=} \bar{b}.\bar{b}.X[f]+a.X+c.X$, $Y \stackrel{def}{=} \bar{d}.\bar{d}.Y[f]+a.X+c.X$ and $Z \stackrel{def}{=} b.b.d.d.Z$, using a relabelling operator with $f(a) = c$ and $f(c) = a$. Under $x$A-fairness, for $x \in \{\mathrm{J}, \mathrm{W}, \mathrm{S}\}$, the path $(a\tau)^\infty$ is unfair, because task $c$ is perpetually enabled, yet never taken. However, there are only twelve instructions (and three components), each of which gets its fair share. Also, each possible synchronisation will occur. So under $xy$-fairness for $y \in \{\mathrm{I}, \mathrm{Z}, \mathrm{C}, \mathrm{G}\}$ the path is fair.



*Example 23.* Consider the process $(e|X|Y|Z)\backslash a\backslash b\backslash c\backslash d\backslash e$ where $X \stackrel{def}{=} \bar{a}.\bar{b}.X + \bar{e}$, $Y \stackrel{def}{=} \bar{c}.\bar{d}.Y + \bar{e}$ and $Z \stackrel{def}{=} a.b.c.d.Z$. Its transition system is the same as for Example 12. Under $x$I- and under $x$C-fairness $\mathscr{G}$ is satisfied, but under W$y$-fairness with $y \in \{\mathrm{A}, \mathrm{T}, \mathrm{Z}, \mathrm{G}\}$ it is not.

## 15 FAIRNESS OF EVENTS

An *event* in a transition system can be defined as an equivalence class of transitions stemming from the same synchronisation of instructions, except that each visit of an instruction gives rise to a different event. Fairness of events can best be explained in terms of examples.

In Examples 4 and 5 $\mathscr{G}$ does hold: on the infinite path that violates $\mathscr{G}$ there is a single event corresponding with the action $a$, which is perpetually enabled.

In Example 6 $\mathscr{G}$ does not hold, for each round through the recursive equation counts as a separate visit to the $b$-instruction, so during the run $a^\infty$ no $b$-event is enabled more than once. Similar reasoning applies to Examples 7–12, where $\mathscr{G}$ is not ensured and/or the indicated path is SE-fair.

Fairness of events is defined in Costa & Stirling [14]—although on a restriction-free subset of CCS where it coincides with fairness of components—and in Corradini, Di Berardini & Vogler [12], named fairness of actions.

Capturing fairness of events in terms of tasks, as in Section 5, with the events as tasks, requires a semantics of CCS different from the standard one given in Appendix A, yielding a partially unfolded transition system. Example 6, for instance, needs an infinite transition system to express that after each $a$-transition a different $b$-event is enabled, and likewise for Examples 4, 7, 11 and 12. Such a semantics, involving an extra layer of labelling, appears in [12, 14, 15]. What we need to know here about this semantics is that

(5) the events form a partition of the set of transitions;
(6) if an event $e$ is enabled in two states $s$ and $s'$ on a path $\pi$, then $e$ is enabled also in each state and during each transition that occurs between $s$ and $s'$;
(7) if $t \smile u$ with $source(t) = source(u)$ and $t \in e$, then $\exists v \in Tr$ with $source(v) = target(u)$ and $v \in e$,
(8) if $t, t' \in e$ then $comp(t) = comp(t')$. So let $comp(e) := comp(t)$ when $t \in e$.

Property (6) implies immediately that strong fairness of events (SE), weak fairness of events (WE) and J-fairness of events (JE) coincide.

Theorem 15.1. A path is just iff it is JE-fair.

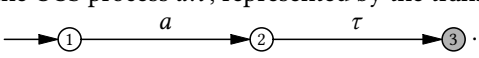
Proof. '$\Rightarrow$': Let $\pi$ be a just path on which an event $e$ is continuously enabled. By justness, a transition $u$ occurs in $\pi$ such that $comp(e) \cap comp(u) \neq \emptyset$. By assumption, event $e$ must be enabled during $u$. Hence there is a $t \in e$ with $t \smile u$, i.e. $comp(t) \cap comp(u) = \emptyset$, in contradiction with (8). Hence $\pi$ is JE-fair.

'$\Leftarrow$': W.l.o.g. let $\pi$ be a JE-fair path such that a transition $t$ is enabled in its first state. Using (5), let $e$ be the event with $t \in e$. Assume that $\pi$ contains no transition $u$ with $comp(t) \cap comp(u) \neq \emptyset$. Then, using (7), $e$ is continuously enabled on $\pi$. So, by JE-fairness, the task $e$ must occur in $\pi$, i.e., $\pi$ contains a transition $v \in e$. By (8) $comp(v) = comp(t)$, but this contradicts the assumption. So $\pi$ is just. □

## 16 REACTIVE SYSTEMS

Sections 2–15 dealt with closed systems, having no run-time interactions with the environment. We now generalise almost all definitions and results to reactive systems, interacting with their environments through synchronisation of actions.

*Example 24.* Consider the CCS process $a.\tau$, represented by the transition system



Here $a$ is the action of receiving the signal $\bar{a}$ from the environment. Will this process satisfy the liveness property $\mathscr{G}$, by reaching state 3? When assuming the progress property for closed systems proposed in Section 3, the answer is positive. However, when taking the behaviour of the environment into account, $\mathscr{G}$ is not guaranteed at all. For the environment may fail to send the signal $\bar{a}$ that is received as $a$, in which case the system will be stuck in its initial state.

To properly model reactive systems, we distinguish *blocking* and *non-blocking* transitions. A blocking transition—$a$ in our example—requires participation of the environment in which the system will be running, whereas a non-blocking transition (e.g. $\tau$) does not. In [54], blocking and non-blocking transitions are called *cold* and *hot*. In many process algebras transitions are labelled with actions, and whether a transition is blocking is entirely determined by its label. Accordingly, we speak of blocking and non-blocking actions. In CCS, the only non-blocking action is $\tau$. However, for certain applications it makes sense to declare some other actions to be non-blocking [27]; this constitutes a promise that we will never put the system in an environment that can block these actions. In [28] we use a process algebra with broadcast communication: a broadcast counts as a non-blocking action, because it will happen regardless whether anyone receives it.

In the setting of reactive systems the progress assumption of Section 3 needs to be reformulated:

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

*Definition 16.1.* A path in a transition system is *progressing* if either it is infinite or its last state is the source of no non-blocking transition.

Our justness assumption is adapted in the same vein:

*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.*

*Definition 16.2.* A path $\pi$ in a transition system is *just* if for each non-blocking transition $t$ with $s := source(t) \in \pi$, a transition $u$ occurs in $\pi$ past the occurrence of $s$, such that $t \not\leadsto u$.

The treatment of fairness is adapted to reactive systems by defining a task $T$ to be *enabled* in a state $s$ iff there exists a *non-blocking* transition $t \in T$ with $source(t) = s$ (cf. Definition 4.1). In Definition 14.1 $t$ is also required to be non-blocking. The reason is that if sufficiently many states on a path merely have an outgoing transition from $T$ that is blocking, it might very well be that the environment blocks all those transitions, so that the system is unable to perform a transition from $T$. On the other hand, performing a blocking transition from $T$ is a valid way to satisfy the promise of fairness. To obtain a notion of fairness where an arbitrary non-blocking transition from $T$ is required to occur in $\pi$, one can simply define a new task that only has the non-blocking transitions from $T$.

With these amendments to the treatment of progress and fairness, the observation remains that progress is exactly what weak or strong fairness prescribes for finite paths, provided that each non-blocking transition occurs in a task.

Applied to CCS, where only $\tau$-actions are non-blocking, fairness of actions looses most of its power. All other relations in the taxonomy remain unchanged. Theorem 6.1 remains valid as well. Definition 9.1 needs to be reformulated as follows.

*Definition 16.3.* A liveness property $\mathscr{G}$, modelled as a set of states in a reactive transition system $G$, is an *AGEF property* iff (a state of) $\mathscr{G}$ is reachable along a non-blocking path[11] from every state $s$ that is reachable (by any means) from an initial state of $G$.

[11] A path containing non-blocking transitions only.

With this adaptation, a non-AGEF property is never ensured, no matter which feasible fairness property is assumed. Yet, Theorem 10.1 still holds, with the *distance*, in the proof, being the length of the shortest non-blocking path to $\mathscr{G}$. We do not generalise probabilistic fairness to reactive systems.

Full fairness for reactive systems is what corresponds to *Koomen's fair abstraction rule* (KFAR), a widely-used proof principle in process algebra, introduced in [8]; see also [3, 4]. Any process-algebraic verification that shows a liveness property of a system $P$ by proving $P$ weakly bisimilar to a system $Q$ where this liveness property obviously holds, implicitly employs full fairness.

## 17 EVALUATING NOTIONS OF FAIRNESS

The following table evaluates the notions of fairness reviewed in this paper against the criteria for appraising fairness properties discussed in Sections 6, 14, 7 and 13.

| | Fu | Ext | SA | ST | SZ | SG | SI | SWI | SC | WA | WT | WZ | WG | WI | WC | JA | JT | JI | JC | J | Pr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| feasibility | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| equivalence robustness | (+) | − | − | − | − | − | − | ± | ± | − | ? | ? | ? | − | − | − | ? | − | − | + | + |
| liveness enhancement | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| pres. under unfolding | + | + | + | − | + | + | + | + | + | + | − | + | + | + | + | + | − | + | + | + | + |
| pres. under action ref. | + | + | + | + | + | + | + | + | + | − | − | − | − | − | − | + | + | + | + | + | + |
| useful for queues | − | − | − | − | − | − | − | + | − | − | − | − | − | − | − | − | − | − | − | − | − |
| typically warranted | − | − | − | − | − | − | − | − | − | − | − | − | − | − | − | − | + | − | − | + | + |

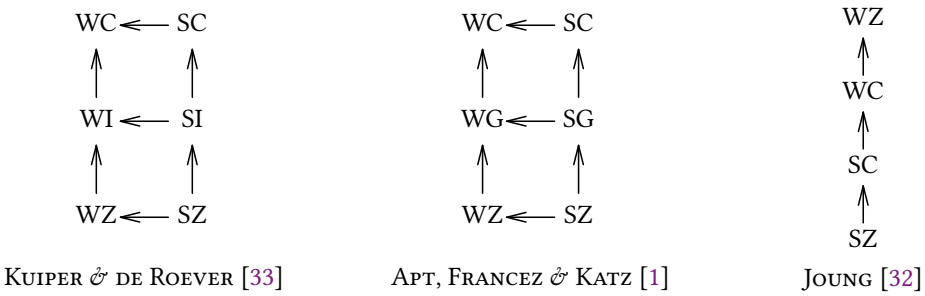
All notions reviewed satisfy the criterion of feasibility, from Section 6.1; this is no wonder, since we disregarded any notion not satisfying this property. The counterexamples given in Section 6.2 against the equivalence robustness of WC- and Sy-fairness, for $y \in \{A,T,Z,G,I\}$, can easily be adapted to cover the other notions adorned with a − in the table above; for SWI and SC this involves the use of $N$-way synchronisation, thereby leaving our fragment of CCS. The notion is meaningless for full fairness (Fu), and the positive results for progress and justness are fairly straightforward. It is trivial to find examples showing that JA and JT-fairness, and thus also all stronger notions, are liveness enhancing w.r.t. the background assumption of 'minimal progress' (cf. Section 6.3). As observed in Section 6.4, only fairness of transitions fails to be preserved under unfolding. Being not preserved under refinement of actions, as discussed in Section 14, is a problem that befalls only the notions of weak fairness. The next entry refers to the typical application of fairness illustrated in Section 7; it is adequately handled only by strong weak fairness. The last line reflects our observation that most notions of fairness are not warranted in many realistic situations. This is illustrated by Example 6.4 in Section 13 for weak and strong fairness, Example 22 for JA-fairness, and Example 23 for JI- and JC-fairness.

## 18 CONCLUSION, RELATED AND FUTURE WORK

We compared and classified many notions of fairness found in the literature. Our classification is by no means exhaustive. We skipped, for instance, *k-fairness* [9] and *hyperfairness* [37]—both laying between strong fairness and full fairness. We skipped *unconditional fairness* [34], introduced in [39] as *impartiality*, since that notion does not satisfy the essential criterion of feasibility. We characterised full fairness as the strongest possible fairness assumption that satisfies this important requirement.

Comparisons of fairness notions appeared before in [1, 32, 33]. Kuiper & de Roever [33] considered weak and strong fairness of components, instructions and synchronisations, there called processes, guards and channels, respectively. Their findings, depicted in Figure 3, agree mostly with ours, see Figure 1. They differ in the inclusions between WZ, WI and WC, which are presented

without proof and deemed "easy". It appears that our Counterexamples 11 and 12 can be translated into the version of CSP used in [33].




Kuiper & de Roever [33] Apt, Francez & Katz [1] Joung [32]

Fig. 3. Earlier classifications of fairness assumptions

Apt, Francez & Katz [1] consider weak and strong fairness of components, groups of components and synchronisations, there called processes, channels or groups, and communications, respectively. The authors work with a restricted subset of CSP, on which the difference between the weak notions of fairness and the corresponding notions of J-fairness largely disappears. Most of their strict inclusions (see Figure 3) agree with ours, except for the ones between WZ, WG and WC. Our counterexamples 11 and 12 cannot be translated into their subset of CSP. In fact, it follows from [1, Prop. 5] that in their setting WZ, WG and WC are equally powerful.

Joung [32] considers weak and strong fairness of components and synchronisations, there called processes and interactions. He also considers many variants of these notions, not covered here. Restricting his hierarchy to the four notions in common with ours yields the third lattice of Figure 3. Joung makes restrictions on the class of modelled systems comparable with those of [1]; as a consequence weak fairness and J-fairness coincide. By this, his lattice agrees with ours on the common elements.

In Section 4 we proposed a general template for strong and weak fairness assumptions, which covers most of the notions reviewed here (but not probabilistic and full fairness). We believe this template also covers interesting notions of fairness not reviewed here. In [10], for instance, two transitions belong to the same task in our sense iff they are both $\tau$-transitions resulting from a $\pi$-calculus synchronisation of actions $\bar{a}y$ and $a(z)$ that both lay within the scope of the same restriction operator $(\nu a)$. A more general definition of what counts as a fairness property, given in the form of a language-theoretic, a game-theoretic, and a topological characterisation, appears in [56].

Using fairness assumptions in the verification of liveness properties amounts to saying that if one tries something often enough, surely it will eventually succeed. Although for some applications this is useful, in many cases it is unwarranted. Progress assumptions, on the other hand, are warranted in many situations, and are moreover indispensable in the verification of liveness properties. Here we introduced the concept of *justness*, a stronger version of progress, that we believe is equally warranted, and, moreover, indispensable for proving many important liveness properties. Special cases of justness are prevalent in the literature on fairness. However, they often occur as underlying assumptions when studying fairness, rather than as alternatives. Moreover, we have not seen them occur in the general form we advocate here.

We showed that as a fairness assumption justness coincides with fairness of events. This could be interpreted as saying that justness is not a new concept. However, properly defining fairness of events requires a much more sophisticated machinery than defining justness. More importantly,

this machinery casts it as a fairness assumption, thus making it equally implausible as other forms of fairness. By recognising the same concept as justness, the precondition of an infinite series of attempts is removed, and the similarity with progress is stressed. This makes justness an appealing notion.

As future work we plan to extend the presented framework—in particular justness—to formalisms providing some form of asymmetric communication such as a broadcast mechanism. Examples for such formalisms are AWN [21] a (process-)algebra for wireless networks, and Petri nets with read arcs [55].

## REFERENCES


[1] Krzysztof R. Apt, Nissim Francez, and Shmuel Katz. 1988. Appraising Fairness in Languages for Distributed Programming. *Distr. Comput.* 2, 4 (1988), 226–241. https://doi.org/10.1007/BF01872848

[2] Krzysztof R. Apt and Ernst-Rüdiger Olderog. 1983. Proof Rules and Transformations Dealing with Fairness. *Science of Computer Programming* 3, 1 (1983), 65–100. https://doi.org/10.1016/0167-6423(83)90004-7 A technical report appeared in 1981.

[3] Jos C. M. Baeten, Jan A. Bergstra, and Jan Willem Klop. 1987. On the Consistency of Koomen's Fair Abstraction Rule. *TCS* 51 (1987), 129–176. https://doi.org/10.1016/0304-3975(87)90052-1

[4] Jos C. M. Baeten and W. Peter Weijland. 1990. *Process Algebra.* Cambridge University Press. https://doi.org/10.1017/CBO9780511624193

[5] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking.* MIT Press.

[6] Keith A. Bartlett, Roger A. Scantlebury, and Peter T. Wilkinson. 1969. A Note on Reliable Full-duplex Transmission over Half-duplex Links. *Comm. ACM* 12, 5 (1969), 260–261. https://doi.org/10.1145/362946.362970

[7] Jan A. Bergstra. 1988. ACP with Signals. In Proc. Int. Workshop on *Algebraic and Logic Programming (LNCS)*, J. Grabowski, P. Lescanne, and W. Wechler (Eds.), Vol. 343. Springer, 11–20. https://doi.org/10.1007/3-540-50667-5_53

[8] Jan A. Bergstra and Jan Willem Klop. 1986. Verification of an Alternating Bit Protocol by Means of Process Algebra. In Proc. MMSSSS'85 *(LNCS)*, Wolfgang Bibel and Klaus P. Jantke (Eds.), Vol. 215. Springer, 9–23. https://doi.org/10.1007/3-540-16444-8_1

[9] Eike Best. 1984. Fairness and Conspiracies. *Inform. Process. Lett.* 18, 4 (1984), 215–220. https://doi.org/10.1016/0020-0190(84)90114-5

[10] Diletta Cacciagrano, Flavio Corradini, and Catuscia Palamidessi. 2009. Explicit Fairness in Testing Semantics. *LMCS* 5, 2 (2009). https://doi.org/10.2168/LMCS-5(2:15)2009

[11] Mario Coppo, Mariangiola Dezani-Ciancaglini, Luca Padovani, and Nobuko Yoshida. 2013. Inference of Global Progress Properties for Dynamically Interleaved Multiparty Sessions. In Proc. Coordination'13 *(LNCS)*, Vol. 7890. Springer, 45–59. https://doi.org/10.1007/978-3-642-38493-6_4

[12] Flavio Corradini, Maria Rita Di Berardini, and Walter Vogler. 2006. Fairness of Actions in System Computations. *Acta Inf.* 43, 2 (2006), 73–130. https://doi.org/10.1007/s00236-006-0011-2

[13] Flavio Corradini, Maria Rita Di Berardini, and Walter Vogler. 2006. Fairness of Components in System Computations. *TCS* 356, 3 (2006), 291–324. https://doi.org/10.1016/j.tcs.2006.02.011

[14] Gerardo Costa and Colin Stirling. 1984. A Fair Calculus of Communicating Systems. *Acta Inf.* 21 (1984), 417–441. https://doi.org/10.1007/BF00271640

[15] Gerardo Costa and Colin Stirling. 1987. Weak and Strong Fairness in CCS. *Information and Computation* 73, 3 (1987), 207–244. https://doi.org/10.1016/0890-5401(87)90013-7

[16] Rocco De Nicola and Frits W. Vaandrager. 1995. Three Logics for Branching Bisimulation. *J. ACM* 42, 2 (1995), 458–487. https://doi.org/10.1145/201019.201032

[17] Edsger W. Dijkstra. 1965. Cooperating Sequential Processes. Reprinted in: P. Brinch Hansen: *The Origin of Concurrent Programming*, Springer, 2002, pp. 65–138. (1965). https://doi.org/10.1007/978-1-4757-3472-0_2

[18] Victor Dyseryn, Robert J. van Glabbeek, and Peter Höfner. 2017. Analysing Mutual Exclusion using Process Algebra with Signals. In Proc. EXPRESS/SOS *(EPTCS)*, Kirstin Peters and Simone Tini (Eds.), Vol. 255. Open Publishing Association, 18–34. https://doi.org/10.4204/EPTCS.255.2

[19] E. Allen Emerson and Edmund M. Clarke. 1982. Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons. *Science of Computer Programming* 2, 3 (1982), 241–266. https://doi.org/10.1016/0167-6423(83)90017-5

[20] E. Allen Emerson and Joseph Y. Halpern. 1986. 'Sometimes' and 'Not Never' Revisited: On Branching Time versus Linear Time Temporal Logic. *J. ACM* 33, 1 (1986), 151–178. https://doi.org/10.1145/4904.4999

[21] Ansgar Fehnker, Robert J. van Glabbeek, Peter Höfner, Annabelle K. McIver, Marius Portmann, and Wee Lum Tan. 2012. A Process Algebra for Wireless Mesh Networks. In Proc. ESOP'12 *(LNCS)*, Helmut Seidl (Ed.), Vol. 7211. Springer,

295–315. https://doi.org/10.1007/978-3-642-28869-2_15
[22] Ansgar Fehnker, Robert J. van Glabbeek, Peter Höfner, Annabelle K. McIver, Marius Portmann, and Wee Lum Tan. 2013. *A Process Algebra for Wireless Mesh Networks used for Modelling, Verifying and Analysing AODV.* Technical Report 5513. NICTA. http://arxiv.org/abs/1312.7645
[23] Michael J. Fischer and Micheal S. Paterson. 1983. Storage Requirements for Fair Scheduling. *Inf. Process. Lett.* 17, 5 (1983), 249–250. https://doi.org/10.1016/0020-0190(83)90107-2
[24] Nissim Francez. 1986. *Fairness.* Springer. https://doi.org/10.1007/978-1-4612-4886-6
[25] Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. 1980. On the Temporal Analysis of Fairness. In Proc. POPL '80, Paul W. Abrahams, Richard J. Lipton, and Stephen R. Bourne (Eds.). ACM Press, 163–173. https://doi.org/10.1145/567446.567462
[26] Robert J. van Glabbeek and Ursula Goltz. 2001. Refinement of Actions and Equivalence Notions for Concurrent Systems. *Acta Inf.* 37 (2001), 229–327. https://doi.org/10.1007/s002360000041
[27] Robert J. van Glabbeek and Peter Höfner. 2015. CCS: It's not fair! *Acta Inf.* 52, 2-3 (2015), 175–205. https://doi.org/10.1007/s00236-015-0221-6
[28] Robert J. van Glabbeek and Peter Höfner. 2015. *Progress, Fairness and Justness in Process Algebra.* Technical Report 8501. NICTA. http://arxiv.org/abs/1501.03268
[29] Robert J. van Glabbeek and Marc Voorhoeve. 2006. Liveness, Fairness and Impossible Futures. In Proc. CONCUR'06 *(LNCS)*, Christel Baier and Joost-Pieter Katoen (Eds.), Vol. 4137. Springer, 126–141. https://doi.org/10.1007/11817949_9
[30] Orna Grumberg, Nissim Francez, Johann A. Makowsky, and Willem-Paul de Roever. 1985. A Proof Rule for Fair Termination of Guarded Commands. *Information and Control* 66, 1/2 (1985), 83–102. https://doi.org/10.1016/S0019-9958(85)80014-0 IFIP conference version 1981.
[31] C.A.R. Hoare. 1978. Communicating Sequential Processes. *Comm. ACM* 21, 8 (1978), 666–677. https://doi.org/10.1145/359576.359585
[32] Yuh-Jzer Joung. 2001. On Fairness Notions in Distributed Systems II. Equivalence-Completions and Their Hierarchies. *Information and Computation* 166, 1 (2001), 35–60. https://doi.org/10.1006/inco.2000.3015
[33] Ruurd Kuiper and Willem-Paul de Roever. 1983. Fairness Assumptions for CSP in a Temporal Logic Framework. In *Formal Description of Programming Concepts II*, Dines Bjørner (Ed.). North-Holland, 159–170.
[34] Marta Z. Kwiatkowska. 1989. Survey of Fairness Notions. *Information and Software Technology* 31, 7 (1989), 371–386. https://doi.org/10.1016/0950-5849(89)90159-6
[35] Leslie Lamport. 1974. A New Solution of Dijkstra's Concurrent Programming Problem. *Commun. ACM* 17, 8 (1974), 453–455. https://doi.org/10.1145/361082.361093
[36] Leslie Lamport. 1977. Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Engineering* 3, 2 (1977), 125–143. https://doi.org/10.1109/TSE.1977.229904
[37] Leslie Lamport. 2000. Fairness and Hyperfairness. *Distr. Comput.* 13, 4 (2000), 239–245. https://doi.org/10.1007/PL00008921
[38] Leslie Lamport. 2002. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers.* Addison-Wesley.
[39] Daniel J. Lehmann, Amir Pnueli, and Jonathan Stavi. 1981. Impartiality, Justice and Fairness: The Ethics of Concurrent Termination. In Proc. ICALP'81 *(*LNCS), Shimon Even and Oded Kariv (Eds.), Vol. 115. Springer, 264–277. https://doi.org/10.1007/3-540-10843-2_22
[40] William C. Lynch. 1968. Reliable Full-duplex File Transmission over Half-duplex Telephone Line. *Comm. ACM* 11, 6 (1968), 407–410. https://doi.org/10.1145/363347.363366
[41] Zohar Manna and Amir Pnueli. 1989. Completing the Temporal Picture. In Proc. ICALP'89 *(LNCS)*, Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca (Eds.), Vol. 372. Springer, 534–558. https://doi.org/10.1007/BFb0035782
[42] Zohar Manna and Amir Pnueli. 1995. *Temporal Verification of Reactive Systems – Safety.* Springer. https://doi.org/10.1007/978-1-4612-4222-2
[43] Robin Milner. 1980. *A Calculus of Communicating Systems.* LNCS, Vol. 92. Springer. https://doi.org/10.1007/3-540-10235-3
[44] Jayadev Misra. 1988. A Rebuttal of Dijkstra's Position on Fairness. (1988). http://www.cs.utexas.edu/users/misra/Notes.dir/fairness.pdf
[45] Jayadev Misra. 2001. *A Discipline of Multiprogramming — Programming Theory for Distributed Applications.* Springer. https://doi.org/10.1007/978-1-4419-8528-6
[46] Susan S. Owicki and Leslie Lamport. 1982. Proving Liveness Properties of Concurrent Programs. *ACM TOPLAS* 4, 3 (1982), 455–495. https://doi.org/10.1145/357172.357178
[47] David M. R. Park. 1981. A Predicate Transformer for Weak Fair Iteration. In Proc. 6$^{th}$ IBM Symposium on *Mathematical Foundations of Computer Science,* Hakone, Japan. 211–228. http://hdl.handle.net/2433/103001

[48] Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir Das. 2003. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental), Network Working Group. (2003). https://doi.org/10.17487/RFC3561
[49] Gary L. Peterson. 1981. Myths About the Mutual Exclusion Problem. *Inform. Process. Lett.* 12, 3 (1981), 115–116. https://doi.org/10.1016/0020-0190(81)90106-X
[50] Amir Pnueli. 1977. The Temporal Logic of Programs. In Proc. FOCS'77. IEEE, 46–57. https://doi.org/10.1109/SFCS.1977.32
[51] Amir Pnueli. 1983. On the Extremely Fair Treatment of Probabilistic Algorithms. In Proc. STOC'83. ACM, 278–290. https://doi.org/10.1145/800061.808757
[52] Vaughan R. Pratt. 1986. Modeling Concurrency with Partial Orders. *International Journal of Parallel Programming* 15, 1 (1986), 33–71. https://doi.org/10.1007/BF01379149
[53] Jean-Pierre Queille and Joseph Sifakis. 1983. Fairness and Related Properties in Transition Systems. *Acta Inf.* 19 (1983), 195–220. https://doi.org/10.1007/BF00265555
[54] Wolfgang Reisig. 2013. *Understanding Petri Nets — Modeling Techniques, Analysis Methods, Case Studies.* Springer. https://doi.org/10.1007/978-3-642-33278-4
[55] Walter Vogler. 2002. Efficiency of Asynchronous Systems, Read Arcs, and the MUTEX-problem. *TCS* 275, 1-2 (2002), 589–631. https://doi.org/10.1016/S0304-3975(01)00300-0
[56] Hagen Völzer and Daniele Varacca. 2012. Defining Fairness in Reactive and Concurrent Systems. *J. ACM* 59, 3 (2012), 13. https://doi.org/10.1145/2220357.2220360

Table 1. Structural operational semantics of CCS

$$\alpha.E \xrightarrow{\alpha} E \qquad \frac{E \xrightarrow{\alpha} E'}{E+F \xrightarrow{\alpha} E'} \qquad \frac{F \xrightarrow{\alpha} F'}{E+F \xrightarrow{\alpha} F'}$$

$$\frac{E \xrightarrow{\alpha} E'}{E|F \xrightarrow{\alpha} E'|F} \qquad \frac{E \xrightarrow{a} E',\ F \xrightarrow{\bar{a}} F'}{E|F \xrightarrow{\tau} E'|F'} \qquad \frac{F \xrightarrow{\alpha} F'}{E|F \xrightarrow{\alpha} E|F'}$$

$$\frac{E \xrightarrow{\alpha} E'}{E\backslash a \xrightarrow{\alpha} E'\backslash a}\ (a{\neq}\alpha{\neq}\bar{a}) \qquad \frac{E \xrightarrow{\alpha} E'}{E[f] \xrightarrow{f(\alpha)} E'[f]} \qquad \frac{S(X)[\mathbf{fix}_Y S/Y]_{Y\in dom(S)} \xrightarrow{\alpha} E}{\mathbf{fix}_X S \xrightarrow{\alpha} E}$$

## A CCS

CCS [43] is parametrised with a set $\mathscr{A}$ of *names*. The set $\bar{\mathscr{A}}$ of *co-names* is $\bar{\mathscr{A}} := \{\bar{a} \mid a \in \mathscr{A}\}$, and $\mathscr{L} := \mathscr{A} \cup \bar{\mathscr{A}}$ is the set of *labels*. Finally, $Act := \mathscr{L} \dot\cup \{\tau\}$ is the set of *actions*. Below, $a$, $b$, $c$, ... range over $\mathscr{L}$ and $\alpha$, $\beta$ over $Act$. A *relabelling* is a function $f : \mathscr{L} \to \mathscr{L}$ satisfying $f(\bar{a}) = \overline{f(a)}$; it extends to $Act$ by $f(\tau) := \tau$. Let $\mathscr{X}$ be a set $X$, $Y$, ... of *process variables*. The set $\mathbb{T}_{CCS}$ of CCS expressions is the smallest set including:

| | | |
|---|---|---|
| $\mathbf{0}$ | | *inaction* |
| $\alpha.E$ | for $\alpha \in Act$ and $E \in \mathbb{T}_{CCS}$ | *action prefixing* |
| $E + F$ | for $E, F \in \mathbb{T}_{CCS}$ | *choice* |
| $E\|F$ | for $E, F \in \mathbb{T}_{CCS}$ | *parallel composition* |
| $E\backslash a$ | for $a \in \mathscr{A}$ and $E \in \mathbb{T}_{CCS}$ | *restriction* |
| $E[f]$ | for $f$ a relabelling and $E \in \mathbb{T}_{CCS}$ | *relabelling* |
| $X$ | for $X \in \mathscr{X}$ | *process variable* |
| $\mathbf{fix}_X S$ | for $S : \mathscr{X} \rightharpoonup \mathbb{T}_{CCS}$ and $X \in dom(S)$ | *recursion.* |

A partial function $S : \mathscr{X} \rightharpoonup \mathbb{T}_{CCS}$ is called a *recursive specification*, and traditionally written as $\{Y \stackrel{def}{=} S(Y) \mid Y \in dom(S)\}$. We often abbreviate $\alpha.\mathbf{0}$ by $\alpha$, and $\mathbf{fix}_X S$ by "$X$ where $S$". A CCS expression $P$ is *closed* if each occurrence of a process variable $Y$ in $P$ lays within a subexpression $\mathbf{fix}_X S$ of $P$ with $Y \in dom(S)$; $\mathrm{T}_{CCS}$ denotes the set of closed CCS expressions, or *processes*.

The traditional semantics of CCS is given by the labelled transition relation $\to \subseteq \mathrm{T}_{CCS} \times Act \times \mathrm{T}_{CCS}$ between closed CCS expressions. The transitions $p \xrightarrow{\alpha} q$ with $p, q \in \mathrm{T}_{CCS}$ and $\alpha \in Act$ are derived from the rules of Table 1.

To extract from each CCS process $P$ a transition system $(S, Tr, source, target, I)$ as in Definition 2.1, take $S$ to be the set of closed CCS expressions reachable from $P$, $I = \{P\}$, and $Tr$ the set of *proofs* $\pi$ of transitions $p \xrightarrow{\alpha} q$ with $p \in S$. Here a *proof* of $p \xrightarrow{\alpha} q$ is a well-founded tree with the nodes labelled by elements of $\mathrm{T}_{CCS} \times Act \times \mathrm{T}_{CCS}$, such that the root has label $p \xrightarrow{\alpha} q$, and if $\mu$ 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 Table 1. Of course $source(\pi) = p$, $target(\pi) = q$ and $\ell(\pi) = \alpha$.

In Section 5 we restrict our attention to the fragment of CCS where $dom(S)$ is finite for all recursive specifications $S$ and parallel composition does not occur in the expressions $S(Y)$. Moreover, each occurrence of a process variable $X$ as well as each occurrence of a parallel composition $H_1|H_2$ in an expression $E + F$ is *guarded*, meaning that it lays in a subexpression of the form $\alpha.G$. Given a process $P$, let $\mathcal{I}$ be the set of all occurrences of action prefix operators $\alpha.\_$ in $P$. To define the function *instr*, give each such action a different name $n$, and carry these names through the proofs of transitions, by using

$$\alpha_n.E \xrightarrow{\alpha}_n E \qquad \frac{E \xrightarrow{a}_n E',\ F \xrightarrow{\bar{a}}_m F'}{E|F \xrightarrow{\tau}_{n,m} E'|F'}$$

and keeping the same list of names in each of the other rules of Table 1. If $\pi \in Tr$ is a proof of a transition $p \xrightarrow{\alpha}_Z q$ with $Z$ a list of names (seen as a set), then $instr(\pi) = Z$. In Appendix B we show that property (1) holds for this fragment of CCS. As CCS expressions are finite objects, property (2) holds as well.

The set $\mathscr{C}$ of parallel components of a closed expression $P$ in our fragment of CCS is defined as the arguments (LEFT or RIGHT) of parallel composition operators occurring in $P$ (or the entire process $P$). A component $C \in \mathscr{C}$ can be named by a prefix-closed subset of the regular language $\{\text{L}, \text{R}\}^*$. Each action occurrence can be associated to exactly one of those components. This yields a function $cp : \mathcal{I} \to \mathscr{C}$.

*Example 25.* The CCS expression $a.(P|b.Q)|U$ has at least five components—more if $P$, $Q$ or $U$ have parallel subcomponents—namely $a.(P|b.Q)$ (named L), $P$ (named LL), $b.Q$ (named LR), and $U$ (named R), and the entire expression (named $\varepsilon$). Let $a_1$ and $b_1$ be the indicated occurrences of $a$ and $b$. Then $cp(a_1) = \text{L}$ and $cp(b_1) = \text{LR}$.

Now the function $comp : Tr \to \mathscr{P}(\mathscr{C})$ is given by $comp(t) = \{cp(I) \mid I \in instr(t)\}$, so that (3) is satisfied.

## B PROOF OF THE UNIQUE SYNCHRONISATION PROPERTY

An expression $F$ is called a *subexpression* of a CCS expression $E$ iff $F \preceq E$, where $\preceq$ is the smallest reflexive and transitive relation on CCS expressions satisfying $E \preceq \alpha.E$, $E \preceq E + F$, $F \preceq E + F$, $E \preceq E|F$, $F \preceq E|F$, $E \preceq E\backslash a$, $E \preceq E[f]$ and $S(Y) \preceq \mathbf{fix}_X S$ for each $Y \in dom(S)$. An *extended subexpression* is defined likewise, but with an extra clause $\mathbf{fix}_Y S \preceq \mathbf{fix}_X S$ for each $X, Y \in dom(S)$. We say that $F$ has an *unguarded occurrence* in $E$ iff $F \preceq E$, where $\preceq$ is defined as $\preceq$, except that the clause $E \preceq \alpha.E$ is skipped, and the clauses for $\mathbf{fix}$ are replaced by $S(X) \preceq \mathbf{fix}_X S$ and

$$\text{if } Y \preceq \mathbf{fix}_X S \text{ and } Y \in dom(S) \text{ then } S(Y) \preceq \mathbf{fix}_X S.$$

If $\alpha.F \preceq E$ then that occurrence of $\alpha$ in $E$ is called *unguarded*.

A *named* CCS expression is an expression $E$ in our fragment of CCS in which each action occurrence is equipped with a name. It is *well-named* if for each extended subexpression $F$ of $E$, all unguarded action occurrences in $F$ have a different name, and moreover, every subexpression $F|G$ of $E$ satisfies $n(F) \cap n(G) = \emptyset$, where $n(E)$ is the set containing all names of action occurrences in $E$. Clearly, if all its action occurrences have different names, $E$ is well-named.

CLAIM 1: If $E$ is well-named, then so is any subexpression $F$ of $E$, and $n(F) \subseteq n(E)$.

PROOF OF CLAIM 1: Directly from the definitions. ∎

CLAIM 2: If $E := \mathbf{fix}_X S$ is well-named then so is $H := S(X)[\mathbf{fix}_Y S/Y]_{Y \in dom(S)}$, and $n(H) \subseteq n(E)$.

PROOF OF CLAIM 2: Clearly $n(H) \subseteq n(S(X)) \cup \bigcup_{Y \in dom(S)} n(\mathbf{fix}_Y S) \subseteq n(E)$.

By the restrictions imposed on our fragment of CCS, $H$ has no subexpressions of the form $F|G$. Let $F$ be an extended subexpression of $H$. We have to show that all unguarded action occurrences in $F$ have different names. Since $F$ is an extended subexpression of $H$, it either is an extended subexpression of $\mathbf{fix}_Y S$ for some $Y \in dom(S)$, or of the form $F'[\mathbf{fix}_Y S/Y]_{Y \in dom(S)}$ for an extended subexpression $F'$ of $S(X)$.

In the first case $F$ is also an extended subexpression of $\mathbf{fix}_X S$. Since the latter expression is well-named, all unguarded action occurrences in $F$ have a different name.

In the second case we consider two subcases. First suppose that no variable $Y \in dom(S)$ has an unguarded occurrence in $F'$. Then all unguarded action occurrences in $F$ are in fact unguarded action occurrences in $F'$. Since $F'$ is an extended subexpression of $E$, they all have a different name.

Now suppose that a variable $Y \in dom(S)$ has an unguarded occurrence in $F'$. Then, by the restrictions imposed on our fragment of CCS, $F'$ has no occurrences of either choice, parallel composition or action prefixing, and $Y$ is the only process variable occurring unguarded in $F'$. So the unguarded action occurrences in $F$ are in fact unguarded action occurrences in $\mathbf{fix}_Y S$. Since the latter is an extended subexpression of $E$, they all have different names. ∎

Claim 3: If $E$ is well-named and $E \xrightarrow{\alpha}_Z F$, then so is $F$.

Proof of Claim 3: Using and Claims 1 and 2, a trivial structural induction on the proof of transitions shows that if $E$ is well-named and $E \xrightarrow{\alpha}_Z F$, then so is $F$, and $n(F) \subseteq n(E)$. ∎

We write $\pi : E \rightarrow_n$ if $\pi$ is a proof of $E \xrightarrow{\alpha}_Z F$ for some $\alpha$, $F$ and $Z$ such that $n \in Z$.

Claim 4: If $\pi : E \rightarrow_n$ then $E$ has an unguarded action occurrence named $n$.

Proof of Claim 4: A straightforward induction on $\pi$. ∎

Claim 5: If $E$ is well-named and $Z \subseteq \mathcal{I}$ then there is at most one proof of a transition $E \xrightarrow{\alpha}_Z F$.

Proof of Claim 5: A straightforward induction on the proof of $E \xrightarrow{\alpha}_Z F$, using Claims 1,2 and 4. ∎

That property (1) holds for the fragment of CCS of Appendix A follows from Claims 3 and 5.