Logical Methods in Computer Science
Vol. 5 (2:15) 2009, pp. 1–27
www.lmcs-online.org

Submitted Jul. 10, 2008
Published Jun. 22, 2009

EXPLICIT FAIRNESS IN TESTING SEMANTICS

DILETTA CACCIAGRANO [a], FLAVIO CORRADINI [b], AND CATUSCIA PALAMIDESSI [c]

[a] Dipartimento di Matematica e Informatica, Università degli Studi di Camerino, Camerino, Italy
e-mail address: {diletta.cacciagrano,flavio.corradini}@unicam.it

[c] INRIA Futurs and LIX, École Polytechnique, France
e-mail address: catuscia@lix.polytechnique.fr

Abstract. In this paper we investigate fair computations in the π-calculus [25]. Following Costa and Stirling's approach for CCS-like languages [10, 11], we consider a method to label process actions in order to filter out unfair computations. We contrast the existing fair-testing notion [35, 26] with those that naturally arise by imposing weak and strong fairness. This comparison provides insight about the expressiveness of the various 'fair' testing semantics and about their discriminating power.

1. Introduction

One of the typical problems of concurrency is to ensure that all the tasks that are supposed to be executed do not get postponed indefinitely in favor of other activities. This property, which is called *fairness*, can be implemented by using a particular scheduling policy that excludes unfair behavior. For instance, in Pict [33], (weak) fairness is obtained by using FIFO channel queues and a round-robin policy for process scheduling. A stronger property (strong fairness) is obtained by using priority queues.

Of course in practice it is not feasible to impose that all implementations adopt a certain scheduler. One reason is that, depending on the underlying machine, one scheduling policy may be much more efficient than another one. Hence fairness has been studied, since the beginning of the research on Concurrency, as an abstract property and independently from the implementation.

1998 ACM Subject Classification: D.2.4, F.1.2.
Key words and phrases: Pi-Calculus, Testing Semantics, Strong Fairness, Weak Fairness.
[a,b] The work of Diletta Cacciagrano and Flavio Corradini has been supported by the Investment Funds for Basic Research (MIUR-FIRB) project Laboratory of Interdisciplinary Technologies in Bioinformatics (LITBIO) and by Halley Informatica.
[c] The work of Catuscia Palamidessi has been partially supported by the INRIA DREI Équipe Associée PRINTEMPS and by the INRIA ARC project ProNoBiS.

© D. Cacciagrano, F. Corradini, and C. Palamidessi
Creative Commons

1.1. **Fairness in literature.** Most of the common notions of fairness share the same general form: "Every *entity* that is enabled *sufficiently often* will eventually make progress." Varying the interpretations of '*entity*' and '*sufficiently often*' leads to different notions of fairness.

Kuiper and de Roever [18] identified a wide hierarchy of fairness notions for the CSP language (channel fairness, process fairness, guard fairness, and communication fairness), according to the entity taken into account (respectively channel, process, guard and communication). Each of these fairness notions have a weak and a strong variant, which differ in the interpretation of *sufficiently often*: weak forms of fairness are concerned with continuously enabled entities, whereas strong forms of fairness are concerned with the infinitely enabled entities.

Independently, Costa and Stirling investigated (weak and strong) fairness of actions for a CCS-like language without restriction in [10], and fairness of components for the full CCS in [11]. An important result of their investigation was the characterization of fair executions in terms of the concatenation of certain finite sequences, called LP-steps. This result allowed expressing fairness as a local property instead than a property of complete maximal executions.

Although [18] and [10, 11] seem to define different fairness varieties, there is a correspondence between some notions in the two approaches (up to the language on which the study is based): guard fairness corresponds to fairness of actions, while process fairness corresponds to fairness of components. However, the communication mechanism of the languages chosen for the study - CSP in [18] and CCS in [10, 11] - modifies the interrelationships among notions. In fact, in CSP processes communicate by name, each channel corresponds precisely to a pair of processes, i.e only two processes communicate along any given channel and only one channel is used between any two processes; on the other hand, in CCS any number of processes may communicate along a given channel, and two processes may communicate along any number of channels. This implies that some fairness notions are related in CSP while they are not related in CCS. For example, while every channel-fair computation is also process-fair in CSP ([15]), in CCS it is possible for a particular channel to be used sufficiently often and yet for another process to become blocked while trying to use that same channel[1].

Hennessy [16] introduced the concept of fairness in his *acceptance trees* model, by adding limit points indicating which infinite paths are fair. The notion of fairness incorporated into this semantics is a form of unconditional fairness: an infinite execution is considered fair if every process makes infinitely many transitions along that computation.

Francez [15] characterized the notions of fairness in [18] in terms of a so-called *machine closure* property and by means of a topological model.

Fairness has also been investigated in the context of probabilistic systems. Koomen [21] explained fairness with probabilistic arguments: the *Fair Abstraction Rule* establishes that no matter how small the probability of success is, if one tries often enough one will eventually succeed. Pnueli introduced in [32] the notion of *extreme fairness* and *α-fairness*, to abstract from the precise values of probabilities.

[1]It suffices to consider the term $\bar{a} \mid !a.\bar{a} \mid \bar{a}$, where a and $\bar{a}$ denote actions of input and output on channel a, respectively, and $!a.\bar{a}$ denotes a process which can perform infinitely often an input on channel a, followed by an output on the same channel. Although channel a must be used infinitely often along any infinite computation, it is possible under *channel fairness* that the leftmost $\bar{a}$ is ignored, while the right-most $\bar{a}$ synchronizes continually with the process $!a.\bar{a}$. This is not the case under process fairness.

1.2. **Fairness in bisimulation equivalences and testing semantics.** Observational equivalences and preorders can have different bearings with respect to fairness. In particular, this is the case of testing preorders [12] and bisimulation equivalences [24, 31].

The first framework was presented by De Nicola and Hennessy in their seminal work [12], where they proposed the concept of testing and defined the *must*- and the *may*-testing semantics, as well as their induced preorders. Given a process P and a test (observer) o,

- P *may* o means that there exists a successful computation from $P \mid o$ (where $\mid$ is the parallel operator, and successful means that there is a state where the special action ω is enabled);
- P *must* o means that every maximal computation from $P \mid o$ is successful;
- The preorder $P \leq_{sat} Q$ means that for any test o, P *sat* o implies Q *sat* o, where *sat* denotes *may* or *must*;
- The equivalence $P \approx_{sat} Q$ means $P \leq_{sat} Q$ and $Q \leq_{sat} P$.

The second framework [24, 31] arises from the principle of (mutual) simulation of systems. The prime representatives of this family are bisimilarity and observation congruence [24]. In particular, weak bisimulation incorporates a particular notion of fairness: it abstracts from the τ-loops (i.e infinite sequences of τ - or internal - actions) in which the "normal" behavior can be resumed each time after a finite sequence of τ-actions. Such a property can be useful in practice - for instance for communication protocols in systems with lossy communication media, which retransmit lost messages. There is a fairness principle implicitly associated with such systems, based on the assumption that the path which stays in the loop forever is not a possible behavior of the system. Interesting proofs of protocol correctness based on this principle are given in [4, 22].

Bisimulation equivalences are usually rather strict, since they depend on the whole branching structure of processes, which in some cases may be not relevant. On the other hand, most of the standard testing preorders interpret τ-loops as divergences, making them quasi-observable. In fact, the *must*-predicate on $P \mid o$ immediately fails if P is able to do a τ-loop that never reaches a successful state. Hence, while the standard testing equivalences are coarser than weak bisimulation in the case of divergence-free processes, they are not comparable with the latter in general.

In [35] and in [26] a new testing semantics was proposed to incorporate the fairness notion: the *fair*-testing (aka *should*-testing) semantics. In contrast to the classical *must*-testing (semantics), *fair*-testing abstracts from certain τ-loops. This is achieved by stating that the test o is satisfied if success always remains within reach in the system under test. In other words, P *fair* o holds if in every maximal computation from $P \mid o$ every state can lead to success after finitely many interactions. The characterizing semantics for *fair*-testing and a similar testing scenario can already be found in [38].

The relation between bisimulations and *fair*-testing was investigated in [13], in the context of name-passing process calculi like the asynchronous π-calculus [19] and the join-calculus [14]. The authors of [13] presented a hierarchy of equivalences obtained as variations of Milner and Sangiorgi's weak barbed bisimulation. In particular, they proved that the coupled barbed equivalence strictly implies the *fair*-testing equivalence. They also showed that those relations coincide in the join-calculus and on a restricted version of the asynchronous π-calculus, called *local* π-calculus, where reception occurs only on names bound by a restriction (not on free and received names).

Another relation motivated by the aim of incorporating in *must*-testing the fairness property of observation congruence is the *acceptance*-testing, which was defined and studied in [5]. This relation is captured by the failures model but, in contrast to *must*-testing, it does not yield a precongruence with respect to abstraction (or hiding), a construction which internalizes visible actions and may thereby introduce new divergences.

The probabilistic intuitions motivating the Koomen's rule inspired another approach to incorporate fairness in a testing semantics [29]. The authors of [29] defined a probabilistic *must*-semantics in which a (probabilistic) process *must*-satisfy a test if and only if the probability with which the process satisfies the test equals 1, and proved that two non-probabilistic processes are *fair*-equivalent if and only if their probabilistic versions are equivalent in the probabilistic testing semantics.

1.3. **The goal of this work: A study of testing semantics with implicit and explicit fairness.** *Fair*-testing is an appealing equivalence. Some of its advantages are that it detects deadlocks and implements fairness. It has also been used in various works. For example, [6] uses the *fair*-testing preorder as an implementation relation for distributed communication protocols.

The purpose of our study is to try to make operationally explicit the fairness assumption which is implicit in the *fair*-testing semantics. The advantages of the formulation in operational terms is to have a better understanding of this notion. Also, it can help eliminating some of the known drawbacks: for example, *fair*-testing abstract fairness is not enforced by practical scheduling policies, and direct proofs of equivalence are very difficult because they involve nested inductions for all quantifiers in the definition of *fair*-testing and all evaluation contexts.

In contrast to [29] we want to keep invariant the original testing scenario and try to characterize (or approximate) *fair*-testing semantics - which does not involve any probability assumption - in term of a non-probabilistic testing semantics equipped with some explicit fairness notion.

We proceed as follows:

- We consider the choiceless π-calculus [25] and we develop for it an approach to fairness (of actions) similar to that which has been proposed in [10, 11] for CCS-like languages [24]. More precisely, we define (i) a *labeling method* for π-calculus terms that ensures that no label occurs more than once in a labeled term (*unicity*), that a label disappears only when the corresponding action is performed (*disappearance*), and that, once it has disappeared, it will not appear in the computation anymore (*persistence*), (ii) the notion of *live action*, which refers to the fact that the action can currently be performed, and (iii) *weak* and *strong fairness* of actions.
- We then contrast the existing *fair*-testing semantics [35, 26] with those that naturally arise by imposing weak and strong fairness [10, 11] on a *must*-testing semantics.

In the following we justify our choices, and describe in detail our setting and results.

1.4. **The choiceless π-calculus.** The choiceless π-calculus is essentially the π-calculus without the choice operator (+). This seems a rather appealing framework to study fairness. In fact, the choice operator is a bit controversial with respect to fairness, because it is not clear what fairness should mean in the case of a repeated execution of a choice construct. In [11] the continuous selection of the same branch of a choice construct turns out to be

fair, while other researcher would not agree to consider fair this kind of computation. The reason why it is fair in [11] is that when the action that has not been selected comes back in the recursive call, it is considered a new action, and it is relabeled. On the contrary, in other approaches, like for instance [18], the guards that come back are precisely the object of weak fairness.

On the other hand, thanks to the fact that the restriction operator "ν" allows the creation of new names and the scope extrusion, the π-calculus is more expressive than CCS, and it is possible to represent in it various types of choices in a compositional way by means of the parallel operator (see [27, 28, 30]). In particular, the internal choice and the input-guarded choice. For example, the term $(\nu a)(\bar{a} \mid a.b.0 \mid a.c.0)$ represents the internal choice between b and c. If we want to repeat the execution of this choice, we use the replication operator "!" which creates an arbitrary number of copies of the argument. The issue of fairness depends on where we place "!" in the term: $!(\nu a)(\bar{a} \mid a.b.0 \mid a.c.0)$ can produce an infinite sequence of "b"'s, and the corresponding computation is considered fair because the subterms $a.b.0$, $a.c.0$ have only one copy of $\bar{a}$ in the same scope, so if such copy synchronizes with $a.b.0$, then $a.c.0$ will be disabled forever. In a sense, the term represents a new choice each time. On the contrary, $(\nu a)!(\bar{a} \mid a.b.0 \mid a.c.0)$ can also produce an infinite sequence of "b"'s, but the corresponding computation is not fair because all the copies of $\bar{a}$ are in the same scope and therefore $a.c.0$ is always enabled. In a sense, here we repeat always the same choice.

We find that the reduction of choice to the parallel operator brings some insight to the relation between repeated choice and fairness, in the sense that the definition of fairness for the various kinds of combination of choice and repetition stems naturally from the definition of fairness for the parallel operator.

1.5. **The labeling method.** In [10, 11], labels are flat sequences of 1's and 2's and are assigned to operators according to the syntactic structure of the term, without distinguishing between static and dynamic operators. In our approach, labels are pairs $\langle s, n\rangle$ in $(\{0,1\}^* \times \mathbb{N})$ and are associated to prefix and replication operators; restriction and parallel operators do not get a label on their own. In contrast to [10, 11], the aim is to keep separated the information about static and dynamic operators and avoid labels which (at least for our purpose) are superfluous, thus making more intuitive their role in the notion of fairness.

The first component of a pair, s, represents the position of the process (whose top-level operator is associated to that label) in the term structure, and it depends only on the (static) parallel operator. This component ensures the unicity of a label. The second component, n, provides information about the dynamics of the process in the term structure. More precisely, it indicates how many actions that process has already executed since the beginning of the computation, and it depends only on the (dynamic) prefix operator. This second component serves to ensure the persistence property of a label.

Informally, a label $\langle s, n\rangle$ denotes unambiguously a parallel process - the one associated to s - and a precise action of it - the one nested at level n in the original term. Note that: (i) all the actions of a parallel process share the first label component s and they only differ from the second component n; (ii) actions of different parallel processes at the same level share the second label component n and are distinguished by the first component s.

We give now an example to illustrate the difference with the labeling method of [10, 11]. We recall that in [10, 11] the labels are assigned essentially by using the tree representing

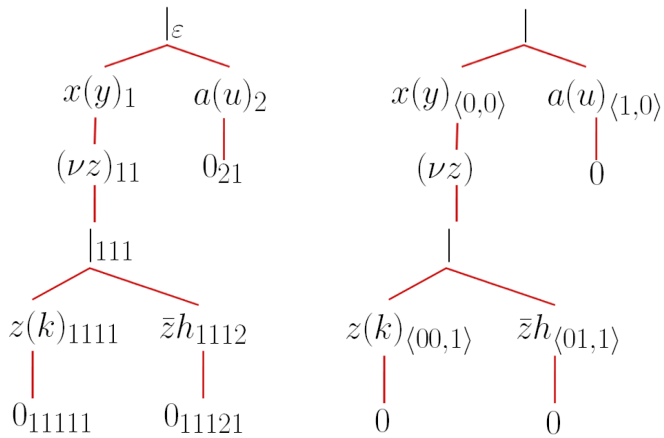

Figure 1: Tree-representation of labeled terms.

the abstract syntax of the term: we add 1 to the string representing the label on the left branch, and 2 on the right branch.

Example 1.1. Consider the term $S = x(y).((\nu z)(z(k).0 \mid \bar{z}h.0)) \mid a(u).0$. The left-most tree in Figure 1 is the the labeling of S in the approach of [10, 11], while the right-most one is the the labeling of S in our approach. The representation of both labeled terms in the usual linear syntax is given in Example 4.7.

1.6. **Testing with explicit fairness vs. fair-testing.** The labeling method allows defining *weak-* and *strong-fair* computations. Using these notions, we adapt *must*-testing semantics [2] to obtain what we call *weak-fair must*-testing semantics and *strong-fair must*-testing semantics. Then we compare these two '*fair*'-testing semantics with the *fair*-testing [35, 26], that does not need any labeling of actions, and with the standard *must*-testing. This comparison reveals the expressiveness of the various testing semantics we consider. In particular:

- we show that *weak-fair must* testing is strictly stronger than *strong-fair must* testing,
- we show that *must*-testing is strictly stronger than *weak-fair must* testing,
- we prove that *strong-fair must* testing is strictly stronger than *fair*-testing,
- we prove that *strong-fair* and *weak-fair must*-testing cannot be characterized by a notion based on the transition tree, like *fair*-testing.

1.7. **Roadmap of the paper.** The rest of the paper is organized as follows. Section 2 recalls the definition of the π-calculus. Section 3 recalls the definition of the *must*-testing and the *fair*-testing semantics. Section 4 shows the labeling method and its main properties. *Weak–fair must-* and *strong-fair must*-testing semantics are defined in Section 5 and compared in Section 6. Finally, in Section 7 we investigate why strong and weak fairness notions are not enough to characterize *fair*-testing semantics. Section 8 contains some concluding remarks and plans for future work. All the proofs omitted in the body of the paper are in the appendixes.

2. The π-calculus

We briefly recall here the basic notions about the (choiceless) π-calculus. Let $\mathcal{N}$ (ranged over by $x, y, z, \dots$) be a set of names. The set $\mathcal{P}$ of processes (ranged over by $P, Q, R, \dots$) is generated by the following grammar:

$$P \quad ::= \quad 0 \;\Big|\; x(y).P \;\Big|\; \bar{x}y.P \;\Big|\; P \mid P \;\Big|\; (\nu x)P \;\Big|\; !P$$

The input prefix $y(x).P$, and the restriction $(\nu x)P$, act as name binders for the name x in P. The free names $fn(P)$ and the bound names $bn(P)$ of P are defined as usual. The set of names of P is defined as $n(P) = fn(P) \cup bn(P)$.

The operational semantics of processes is given via a labeled transition system, whose states are the process themselves. The labels (ranged over by $\mu, \gamma, \dots$) "correspond" to prefixes, input xy and output $\bar{x}y$, and to the bound output $\bar{x}(y)$ (which models scope extrusion). If $\mu = xy$ or $\mu = \bar{x}y$ or $\mu = \bar{x}(y)$ we define $sub(\mu) = x$ and $obj(\mu) = y$. The functions $fn(\cdot)$, $bn(\cdot)$ and $n(\cdot)$ are extended to cope with labels as follows:

$$\begin{array}{llll} bn(xy) = \emptyset & bn(\bar{x}(y)) = \{y\} & bn(\bar{x}y) = \emptyset & bn(\tau) = \emptyset \\ fn(xy) = \{x, y\} & fn(\bar{x}(y)) = \{x\} & fn(\bar{x}y) = \{x, y\} & fn(\tau) = \emptyset \end{array}$$

We take into account the early operational semantics for $\mathcal{P}$ in [37], as shown in Table 1. We only omit symmetric rules of Par, Com and Close for simplicity, and we assume alpha-conversion to avoid collision of free and bound names.

$$\text{Input } x(y).P \xrightarrow{xz} P\{z/y\}$$

$$\text{Output } \bar{x}y.P \xrightarrow{\bar{x}y} P$$

$$\text{Open } \frac{P \xrightarrow{\bar{x}y} P'}{(\nu y)P \xrightarrow{\bar{x}(y)} P'}\, x \neq y \qquad \text{Res } \frac{P \xrightarrow{\mu} P'}{(\nu y)P \xrightarrow{\mu} (\nu y)P'}\, y \notin n(\mu)$$

$$\text{Par } \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\; bn(\mu) \cap fn(Q) = \emptyset$$

$$\text{Com } \frac{P \xrightarrow{xy} P',\ Q \xrightarrow{\bar{x}y} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \text{Close } \frac{P \xrightarrow{xy} P',\ Q \xrightarrow{\bar{x}(y)} Q'}{P \mid Q \xrightarrow{\tau} (\nu y)(P' \mid Q')}\, y \notin fn(P)$$

$$\text{Rep } \frac{P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P' \mid !P}$$

Table 1: Early operational semantics for $\mathcal{P}$ terms.

Definition 2.1. (*Weak transitions*) Let P and Q be $\mathcal{P}$ processes. Then:
– $P \stackrel{\varepsilon}{\Longrightarrow} Q$ iff $\exists\, P_0, ..., P_n \in \mathcal{P}$, $n \geq 0$, s.t. $P = P_0 \stackrel{\tau}{\longrightarrow} ... \stackrel{\tau}{\longrightarrow} P_n = Q$;
– $P \stackrel{\mu}{\Longrightarrow} Q$ iff $\exists\, P_1, P_2 \in \mathcal{P}$ s.t. $P \stackrel{\varepsilon}{\Longrightarrow} P_1 \stackrel{\mu}{\longrightarrow} P_2 \stackrel{\varepsilon}{\Longrightarrow} Q$.

Notation 2.2. For convenience, we write $x(y)$ and $\bar{x}y$ instead of $x(y).0$ and $\bar{x}y.0$, respectively. Furthermore, we write $P \stackrel{\mu}{\longrightarrow}$ (respectively $P \stackrel{\mu}{\Longrightarrow}$) to mean that there exists P' such that $P \stackrel{\mu}{\longrightarrow} P'$ (respectively $P \stackrel{\mu}{\Longrightarrow} P'$) and we write $P \stackrel{\varepsilon}{\Longrightarrow}\stackrel{\mu}{\longrightarrow}$ to mean that there are P' and Q such that $P \stackrel{\varepsilon}{\Longrightarrow} P'$ and $P' \stackrel{\mu}{\longrightarrow} Q$.

3. Testing semantics

In this section we briefly summarize the basic definitions behind the testing machinery for the π-calculus.

Definition 3.1. (*Observers*)
– Let $\omega \notin \mathcal{N}$. ω denotes a special action used to report success. By convention $fn(\omega) = bn(\omega) = \emptyset$.
– The set $\mathcal{O}$ (ranged over by $o, o', o'', \ldots$) of observers is defined like $\mathcal{P}$, where the grammar is extended with the production $P ::= \omega.P$.
– The operational semantics of $\mathcal{P}$ is extended to $\mathcal{O}$ by adding $\omega.P \stackrel{\omega}{\longrightarrow} P$.

Definition 3.2. (*Experiments*) The set of experiments over $\mathcal{P}$ is defined as

$$\mathcal{E} = \{\, (P \mid o) \mid\ P \in \mathcal{P},\, o \in \mathcal{O}\}$$

Definition 3.3. (*Maximal Computations*) Given $P \in \mathcal{P}$ and $o \in \mathcal{O}$, a maximal computation from $P \mid o$ is either an infinite sequence of the form

$$P \mid o = T_0 \stackrel{\tau}{\longrightarrow} T_1 \stackrel{\tau}{\longrightarrow} T_2 \stackrel{\tau}{\longrightarrow} \ldots$$

or a finite sequence of the form

$$P \mid o = T_0 \stackrel{\tau}{\longrightarrow} T_1 \stackrel{\tau}{\longrightarrow} \ldots \stackrel{\tau}{\longrightarrow} T_n \stackrel{\tau}{\not\longrightarrow} .$$

We are now ready to define *must*- and *fair*-testing semantics.

Definition 3.4. (*Must- and Fair-Testing Semantics*) Given a process $P \in \mathcal{P}$ and an observer $o \in \mathcal{O}$, define:
– P *must* o if and only if *for every* maximal computation from $P \mid o$

$$P \mid o = T_0 \stackrel{\tau}{\longrightarrow} T_1 \stackrel{\tau}{\longrightarrow} \ldots \stackrel{\tau}{\longrightarrow} T_i\, [\stackrel{\tau}{\longrightarrow} \ldots]$$

there exists $i \geq 0$ such that $T_i \stackrel{\omega}{\longrightarrow}$;
– P *fair* o if and only if *for every* maximal computation from $P \mid o$

$$P \mid o = T_0 \stackrel{\tau}{\longrightarrow} T_1 \stackrel{\tau}{\longrightarrow} \ldots \stackrel{\tau}{\longrightarrow} T_i\, [\stackrel{\tau}{\longrightarrow} \ldots]$$

we have $T_i \stackrel{\omega}{\Longrightarrow}$, for every $i \geq 0$.

4. A labeled version of the π-calculus

In order to deal with the notion of fairness of actions [10], we first need to introduce a labeling method. Consider the following term:

$$P = \bar{a} \mid !a.\bar{a} \mid \bar{a}.$$

Notice that every maximal computation from P is always of the form

$$P \xrightarrow{\tau} P \xrightarrow{\tau} P \xrightarrow{\tau} \dots$$

However, without labels we would not be able to distinguish fair computations from unfair ones, since we do not know which $\bar{a}$ synchronizes with $!a.\bar{a}$ and makes progress at each step. So, we need to be able to refer unambiguously to individual actions and to monitor them along any computation.

4.1. **The idea behind the labeling method.** A 'reasonable' labeling method, independently from the choice of the labels domain, has to provide *unicity* (e.g. no label occurs more than once in a labeled term), *disappearance* (e.g. a label disappears only when the corresponding action is performed) and *persistence* (e.g. once a label disappears, it does not appear in the computation anymore).

The labeling method can be more or less informative, in the sense that the degree of information about the structure of terms (static information) and about the computation history (dynamic information) can vary. For our purpose we find useful to adopt a labeling method which is rather informative and keeps separate the static and dynamic aspects.

Definition 4.1. (*Ground Labeled $\mathcal{P}$*) We define $\mathcal{P}^e_{gr}$ as the language generated by the following grammar:

$$E ::= 0 \;\Big|\; \mu_{\langle s,n\rangle}.E \;\Big|\; (\nu x)E \;\Big|\; E \mid E \;\Big|\; !_{\langle s,n\rangle}P$$

where $s \in \{0,1\}^*$, $n \in \mathbb{N}$, $P \in \mathcal{P}$ and the prefix μ is of the form $x(y)$ or $\bar{x}y$.

Obviously, $\mathcal{P}^e_{gr}$ also contains labeled terms in which the labels do not respect the structure and/or the execution order. To avoid this problem, we restrict the labeled language to those terms which are *well-formed*. The *well-formedness* predicate $wf(\cdot)$ (Table 4), allows us to obtain a well-defined labeling method; it is defined by using a binary relation $\Re$ over sets of labels, which checks the absence of label conflicts in the parallel composition, and a labeling function $L_{\langle s,n\rangle}(\cdot)$, where $s \in \{0,1\}^*$ and $n \in \mathbb{N}$, which allows us to avoid label conflicts in the prefix composition.

First, we define $\Re$: if L_0 and L_1 are sets of labels, $L_0 \,\Re\, L_1$ holds if and only if for every $\langle s_0, n_0\rangle \in L_0$ and $\langle s_1, n_1\rangle \in L_1$, the first elements of the labels, s_0 and s_1, are not related w.r.t. the usual prefix relation between strings. Formally:

Definition 4.2.

1. Given two strings $s_0,\ s_1 \in \{0,1\}^*$, we write $s_0 \sqsubseteq s_1$ if and only if s_0 is a prefix of s_1, i.e. $s_1 = s_0\alpha$ for some $\alpha \in \{0,1\}^*$;
2. Given $L_0,\ L_1 \subseteq (\{0,1\}^* \times \mathbb{N})$, we write $L_0 \,\Re\, L_1$ if and only if $\forall\langle s_0, n_0\rangle \in L_0.\quad \forall\langle s_1, n_1\rangle \in L_1.\ s_0 \not\sqsubseteq s_1$ and $s_1 \not\sqsubseteq s_0$.

Remark 4.3. From Definition 4.2, it follows immediately that

$$L_0 \,\Re\, L_1 \;\text{ implies }\; \forall\langle s_0, n_0\rangle \in L_0.\, \forall\langle s_1, n_1\rangle \in L_1.\, \langle s_0, n_0\rangle \neq \langle s_1, n_1\rangle.$$

Then, the labeling function $L_{\langle s,n\rangle}(\cdot)$ is defined following inductively the $\mathcal{P}$ terms operational structure.

Definition 4.4. Let $P \in \mathcal{P}$. Define $L_{\langle s,n\rangle}(P)$, where $\langle s,n\rangle \in (\{0,1\}^* \times \mathbb{N})$, as in Table 2.

$L_{\langle s,n\rangle}(0)$	$=$	0
$L_{\langle s,n\rangle}(\mu.P)$	$=$	$\mu_{\langle s,n\rangle}.L_{\langle s,n+1\rangle}(P)$
$L_{\langle s,n\rangle}(P_0 \mid P_1)$	$=$	$L_{\langle s0,n\rangle}(P_0) \mid L_{\langle s1,n\rangle}(P_1)$
$L_{\langle s,n\rangle}((\nu x)P)$	$=$	$(\nu x)L_{\langle s,n\rangle}(P)$
$L_{\langle s,n\rangle}(!P)$	$=$	$!_{\langle s,n\rangle}P$

Table 2: Labeling function $L_{\langle s,n\rangle}(.)$.

We will use the relation $\Re$ in combination with the function $top(\cdot)$, defined in Table 3, which gives the top-level label set of a labeled term. In the same table we define also the function $lab(\cdot)$, which returns the whole set of labels, and which will be useful later.

$E = 0$:	$top(E) = \emptyset$	$lab(E) = \emptyset$
$E = \mu_{\langle s,n\rangle}.E'$:	$top(E) = \{\langle s,n\rangle\}$	$lab(E) = \{\langle s,n\rangle\} \cup lab(E')$
$E = (\nu x)E'$:	$top(E) = top(E')$	$lab(E) = lab(E')$
$E = E_0 \mid E_1$:	$top(E) = top(E_0) \cup top(E_1)$	$lab(E) = lab(E_0) \cup lab(E_1)$
$E = !_{\langle s,n\rangle}P$:	$top(E) = \{\langle s,n\rangle\}$	$lab(E) = \{\langle s,n\rangle\}$

Table 3: Function $top(\cdot)$ and $lab(\cdot)$.

Remark 4.5. From the definitions in Table 3, we have that

$$\forall E \in \mathcal{P}^e_{gr}.\ top(E) \subseteq lab(E)$$

Finally, Table 4 defines formally the well-formedness predicate $wf(\cdot)$. Note that we use $\Re$ to check the lack of conflict, between labels in parallel components, at the top-level only. This constraint will turn out to be sufficient. In fact, in Lemma A.6 in the appendix it is proved that

$$top(E_0)\ \Re\ top(E_1) \ \text{ implies } \ lab(E_0)\ \Re\ lab(E_1).$$

Now we are ready to define the set of labeled $\mathcal{P}$-calculus terms, denoting it by $\mathcal{P}^e$.

$$\text{Nil}\ \frac{}{wf(0)} \qquad \text{Pref}\ \frac{\mu.P \in \mathcal{P}}{wf(L_{\langle s,n\rangle}(\mu.P))}$$

$$\text{Par}\ \frac{wf(E_0), \quad wf(E_1), \quad top(E_0)\ \Re\ top(E_1)}{wf(E_0 \mid E_1)}$$

$$\text{Res}\ \frac{wf(E)}{wf((\nu x)E)} \qquad \text{Rep}\ \frac{P \in \mathcal{P}}{wf(!_{\langle s,n\rangle}P)}$$

Table 4: Well formed terms.

Definition 4.6. The labeled $\mathcal{P}$-calculus, denoted by $\mathcal{P}^e$, is the set

$$\{E \in \mathcal{P}^e_{gr} \mid wf(E)\}$$

It would be possible to defined well-formed terms without explicitly relying on the labeling function: for example, defining an ordering relation between labels to characterize well-formedness of prefixing. However, our aim is to keep separated static and dynamic informations. More in detail, $\mathcal{P}^e$ contains all the well-formed processes of the form '$L_{\langle s,n\rangle}(P)$' (Lemma A.4). However, the operational semantics of $\mathcal{P}^e$, introduced in the following, does not preserve the '$L_{\langle s,n\rangle}(P)$' format: for this reason, the $wf(.)$ predicate is defined in order to ensure the closure of $\mathcal{P}^e$ w.r.t $\xrightarrow{\tau}$.

Example 4.7. Consider again the term $S = x(y).((\nu z)(z(k).0 \mid \bar{z}h.0)) \mid a(u).0$ of Example 1.1. In the approach of [10, 11], the labeling of S would give the term

$$x(y)_1.((\nu z)_{11}(z(k)_{1111}.0_{11111} \mid_{111} \bar{z}h_{1112}.0_{11121})) \mid_\varepsilon a(u)_2.0_{21}.$$

In our approach, the labeling of S is the term

$$x(y)_{\langle 0,0\rangle}.((\nu z)(z(k)_{\langle 00,1\rangle}.0 \ \mid\ \bar{z}h_{\langle 01,1\rangle}.0)) \ \mid\ a(u)_{\langle 1,0\rangle}.0.$$

4.2. **Some properties of the labeled π-calculus.** The operational semantics of $\mathcal{P}^e$ is similar to the one in Table 1; we simply ignore labels in order to derive a transition. The only rule that needs attention is the one for processes in the scope of the replication operator, since the unfolding generates new parallel processes and we must ensure unicity, disappearance and persistence of labels. We use the dynamic labeling described in Table 5.

$\mathcal{P}^e$ is trivially closed w.r.t. renaming, since a renaming does not change labels. It follows that the language is closed w.r.t. $\xrightarrow{\tau}$.

Next result states the main properties which make our labeling method 'reasonable':

Theorem 4.8. *Let $E \in \mathcal{P}^e$.*

1. *(Unicity) No label $\langle s,n\rangle$ occurs more than once in E;*
2. *(Disappearance) If $E \xrightarrow{\mu} E'$ then $\exists\langle s,n\rangle \in lab(E).\ \langle s,n\rangle \notin lab(E')$;*

$$\text{Rep} \quad \frac{P \xrightarrow{\mu} P'}{!_{\langle s,n\rangle}P \xrightarrow{\mu} L_{\langle s0,n+1\rangle}(P') \mid !_{\langle s1,n+1\rangle}P}$$

Table 5: Replication Rule in $\mathcal{P}^e$.

3. *(Persistence)* $\forall k \geq 1.\ E_0 \xrightarrow{\mu_0} E_1 \xrightarrow{\mu_1} E_2 \xrightarrow{\mu_2} \ldots \xrightarrow{\mu_{k-1}} E_k$, *if* $\langle s,n\rangle \in lab(E_0) \cap lab(E_k)$ *then* $\langle s,n\rangle \in lab(E_i)$ *for any* $i \in [1..(k-1)]$.

Proof.

(1) By induction on the structure of E.
- $E = 0$: then $lab(0) = \emptyset$.
- $E = L_{\langle s,n\rangle}(\mu.P')$: then $lab(E) = \{\langle s,n\rangle\} \cup lab(L_{\langle s,n+1\rangle}(P'))$. By Lemma A.4, $wf(L_{\langle s,n+1\rangle}(P'))$, i.e. $L_{\langle s,n+1\rangle}(P') \in \mathcal{P}^e$ and, by induction hypothesis, for every $\langle s',n'\rangle \in lab(L_{\langle s,n+1\rangle}(P'))$, $\langle s',n'\rangle$ does not occur more than once in $lab(L_{\langle s,n+1\rangle}(P'))$. By Lemma A.3, $\forall\langle s',n'\rangle \in lab(L_{\langle s,n+1\rangle}(P'))$. $s \sqsubseteq s'$ and $n+1 \leq n'$. Hence $\langle s,n\rangle \notin lab(L_{\langle s,n+1\rangle}(P'))$.
- $E = (E_0|E_1)$: by definition, $\forall i \in \{0,1\}.wf(E_i)$ holds, implying $E_i \in \mathcal{P}^e$, and $lab(E) = \bigcup_i lab(E_i)$. By induction hypothesis, for every $i \in \{0,1\}$ and every $\langle s_i,n_i\rangle \in lab(E_i)$, $\langle s_i,n_i\rangle$ does not occur more than once in $lab(E_i)$. By Lemma A.6, $top(E_0)\ \Re\ top(E_1)$ implies $lab(E_0)\ \Re\ lab(E_1)$, i.e. $\forall\langle s_0,n_0\rangle \in lab(E_0).\ \forall\langle s_1,n_1\rangle \in lab(E_1).\ \langle s_0,n_0\rangle \neq \langle s_1,n_1\rangle$. Hence, for every $i \in \{0,1\}$ and every $\langle s_i,n_i\rangle \in lab(E_i)$, $\langle s_i,n_i\rangle$ does not occur more than once in $lab(E)$.
- Cases $E = (\nu x)E'$ and $E = !_{\langle s,n\rangle}P$ can be proved similarly.

(2) By Remark 4.5 and Lemma A.8, it suffices to prove that

$$E \xrightarrow{\mu} E' \text{ implies } \exists\langle s,n\rangle \in top(E).\ \langle s,n\rangle \notin top(E').$$

In fact, $\langle s,n\rangle \in top(E)$ implies $\langle s,n\rangle \in lab(E)$ (by Remark 4.5), and $\langle s,n\rangle \notin top(E')$ implies $\langle s,n\rangle \notin lab(E')$ (by Lemma A.8).

By induction on the depth of $E \xrightarrow{\mu} E'$.
- *Rule Input/Output*: $E = L_{\langle s,n\rangle}(\mu.P) \xrightarrow{\mu} E' = L_{\langle s,n+1\rangle}(P')$ (either $P' = P$ or $P' = P\{z/y\}$). $top(L_{\langle s,n\rangle}(\mu.P)) = \{\langle s,n\rangle\}$ and, by Lemma A.3 on $L_{\langle s,n+1\rangle}(P')$, $\forall\langle r',m'\rangle \in top(L_{\langle s,n+1\rangle}(P')) \subseteq lab(L_{\langle s,n+1\rangle}(P'))$. $s \sqsubseteq r'$ and $n+1 \leq m'$. Hence $\langle s,n\rangle \in top(E)$ and $\langle s,n\rangle \notin top(L_{\langle s,n+1\rangle}(P'))$.
- *Rule Par*: $E = (E_0 \mid E_1) \xrightarrow{\mu} (E_0' \mid E_1)$, where $bn(\mu) \cap fn(E_1) = \emptyset$ and $E_0 \xrightarrow{\mu} E_0'$. By induction hypothesis, $\exists\langle r_0,m_0\rangle \in top(E_0).\langle r_0,m_0\rangle \notin top(E_0')$. Since $top(E_0)\Re top(E_1)$ holds, then $\{\langle r_0,m_0\rangle\}\Re top(E_1)$, i.e. $\langle r_0,m_0\rangle \notin top(E_1)$. We conclude that $\langle r_0,m_0\rangle \notin top(E_0' \mid E_1)$.
- *Rule Com*: $E = (E_0 \mid E_1) \xrightarrow{\tau} (E_0' \mid E_1')$, where $E_0 \xrightarrow{xy} E_0'$ and $E_1 \xrightarrow{\bar{x}y} E_1'$. By induction hypothesis, $\exists\langle r_0,m_0\rangle \in top(E_0).\ \langle r_0,m_0\rangle \notin top(E_0')$ and $\exists\langle r_1,m_1\rangle \in top(E_1).\ \langle r_1,m_1\rangle \notin top(E_1')$.
  Consider $\langle r_0,m_0\rangle$ (case $\langle r_1,m_1\rangle$ is symmetric). Since $wf(E_0 \mid E_1)$, then we have $top(E_0)\ \Re\ top(E_1)$. This implies $\{\langle r_0,m_0\rangle\}\Re top(E_1)$.

By item (3) of Lemma A.5 on $E_1 \xrightarrow{\bar{x}y} E_1'$, $\forall \langle r_1', m_1' \rangle \in top(E_1')$. $\exists \langle r', m' \rangle \in top(E_1)$. $r' \sqsubseteq r_1'$ and $m' \leq m_1'$. By Lemma A.2, it follows that $\{\langle r_0, m_0 \rangle\} \Re\, top(E_1')$, and therefore $\langle r_0, m_0 \rangle \notin top(E_1')$. We can conclude that $\langle r_0, m_0 \rangle \in top(E_0 \mid E_1)$ and $\langle r_0, m_0 \rangle \notin top(E_0') \cup top(E_1') = top(E_0' \mid E_1')$.

– *Rule Open/Res/Close/Rep*: These cases can be proved similarly.

(3) In [11] (Lemma 8.8), the analogous property is only proved for $k = 2$. However, the general case cannot be obtained by induction, since the reasoning for the case $k = 2$ does not contain the essential elements to prove the inductive step. Differently from [11], we prove the property in the general case. We proceed as follows.

By contradiction, let $i \in [1..(k-1)]$ be the least index such that $\langle s, n \rangle \notin lab(E_i)$ and let $j \in [(i+1)..k]$ be the least index such that $\langle s, n \rangle \in lab(E_j)$. By the minimality of i, we can apply Lemma A.7 and we obtain that $\langle s, n \rangle \in top(E_{i-1})$. By item (2) of Lemma A.5 on E_j, $\exists \langle r_j, m_j \rangle \in top(E_j)$. $r_j \sqsubseteq s$ and $m_j \leq n$. By item (3) of Lemma A.5 on $E_c \xrightarrow{\mu_c} E_{c+1}$ for any $c \in [(i-1)..(j-1)]$, $\exists \langle r_c, m_c \rangle \in top(E_c)$. $r_c \sqsubseteq r_{c+1}$ and $m_c \leq m_{c+1}$. It follows that $\exists \langle r_{i-1}, m_{i-1} \rangle \in top(E_{i-1})$. $r_{i-1} \sqsubseteq s$ and $m_{i-1} \leq n$.

– In the case $\langle r_{i-1}, m_{i-1} \rangle$ and $\langle s, n \rangle$ are distinct labels: we contradict item (1) of Lemma A.5.

– In the case $\langle r_{i-1}, m_{i-1} \rangle = \langle s, n \rangle$: it follows that $\forall c \in ((i-1)..(j-1)]$. $\exists \langle r_c, m_c \rangle \in top(E_c)$. $s \sqsubseteq r_c \sqsubseteq s$ and $n \leq m_c \leq n$, i.e. $s = r_c$ and $n = m_c$, contradicting that $\langle s, n \rangle \notin lab(E_c)$. □

Remark 4.9. The disappearance property states that a label *disappears* when the corresponding action is performed. On the other hand, the persistence ensures a *complete disappearance* of a label, once the corresponding action is performed. In fact, it is clear that for $E_0 \xrightarrow{\mu_0} E_1 \xrightarrow{\mu_1} \ldots \xrightarrow{\mu_{k-1}} E_k$ with $\langle s, n \rangle \in lab(E_0) \cap lab(E_k)$ the existance of some $h \in [1..(k-1)]$ satisfying $\langle s, n \rangle \notin lab(E_h)$ would contradict item (3).

As expected, the labeled language is a *conservative extension* of the unlabeled one. To prove the statement, we have to formally define the $\mathcal{P}$ process that is obtained by deleting all the labels appearing within a labeled term.

Definition 4.10. Let $E \in \mathcal{P}^e$. Define $Unl(E)$ as the $\mathcal{P}$ process obtained by removing all the labels in E. It can be defined by induction as in Table 6.

$Unl(0)$	$=$	0
$Unl(\mu_{\langle s,n \rangle}.E)$	$=$	$\mu.Unl(E)$
$Unl(E_0 \mid E_1)$	$=$	$Unl(E_0) \mid Unl(E_1)$
$Unl((\nu x)E)$	$=$	$(\nu x)Unl(E)$
$Unl(!_{\langle s,n \rangle}P)$	$=$	$!P$

Table 6: Function $Unl(\cdot)$.

The conservative property of the labeled extension is expressed by the following lemma, which can be proved by induction on the depth of $E \xrightarrow{\mu} E'$ (item (1)) and $Unl(E) \xrightarrow{\mu} P'$ (item (2)).

Proposition 4.11. *Let $E \in \mathcal{P}^e$.*

1. $E \xrightarrow{\mu} E'$ *implies* $Unl(E) \xrightarrow{\mu} Unl(E')$*;*
2. $Unl(E) \xrightarrow{\mu} P'$ *implies* $\exists E' \in \mathcal{P}^e.\ E \xrightarrow{\mu} E'$ *and* $Unl(E') = P'$*.* □

5. Strong and weak fairness of actions

The labeling method proposed in the previous section can be extended in a natural way over the observers, adding $B ::= \omega.B$ in the grammar of $\mathcal{P}^e_{gr}$, $\omega.o \xrightarrow{\omega} o$ in the operational semantics and extending the functions $L_{\langle s,n\rangle}$, $top(\cdot)$, $lab(\cdot)$, $Unl(\cdot)$ and the predicate $wf(\cdot)$ as shown in Table 7. No label is associated to ω since we do not need to distinguish ω occurrences[2].

$(L_{\langle s,n\rangle}/Unl)$	$L_{\langle s,n\rangle}(\omega.o)$	$=$	$Unl(L_{\langle s,n\rangle}(\omega.o))$	$=$	$\omega.o$
(top/lab)	$top(\omega.o)$	$=$	$lab(\omega.o)$	$=$	$\emptyset$
(wf)	$\dfrac{\omega.o \in \mathcal{O}}{wf(\omega.o)}$				

Table 7: Labeling method extension over observers.

In the following, $\mathcal{O}^e$ (ranged over by $\rho, \rho', ..$) denotes the set of labeled observers and $\mathcal{E}^e$ denotes the set of labeled experiments over $\mathcal{P}^e$, as expected.

The definition of *live label* is crucial in the notion of fairness. Given a labeled experiment $S \in \mathcal{E}^e$, a *live label* is a label associated to a top-level action which can immediately be performed, i.e. an input/output prefix able to synchronize. Table 8 defines the live labels of a labeled experiment $S \in \mathcal{E}^e$, according to the labeling method proposed in Section 4. Informally, Table 8 is a rephrasing of operational rules: even if live labels cannot be directly defined in term of transitions, deductions of live predicate mime the proof for a derivation. As a consequence, ω is not live, since a complementary action ($\overline{\omega}$) does not exist. Given a labeled experiment S, the set of S live labels is denoted by $Ll(S)$.

Definition 5.1. Let $S \in \mathcal{E}^e$, let $\langle s, n\rangle \in (\{0,1\}^* \times \mathbb{N})$.

$$Ll(S) = \{\langle s, n\rangle \in (\{0,1\}^* \times \mathbb{N}) \mid live(\langle s, n\rangle, \tau, S)\}$$

is the set of live labels associated to initial $\xrightarrow{\tau}$ from S.

If $S \not\xrightarrow{\tau}$, then $Ll(S) = \emptyset$. Since $top(S)$ is defined as the set of labels appearing at the top of S, it follows immediately by the definition of live actions that $Ll(S) \subseteq top(S)$. For simplicity, labels will be denoted in the following by $v, v_1, v_2, \ldots \in (\{0,1\}^* \times \mathbb{N})$.

[2] $E \xrightarrow{\omega}$ whenever an *arbitrary* occurrence of ω is at the top level in E.

$$\text{Input}\quad \frac{x, y, z \in \mathcal{N}}{\mathit{live}(\langle s, n\rangle, xz, x(y)_{\langle s,n\rangle}.S)}$$

$$\text{Output}\quad \frac{x, z \in \mathcal{N}}{\mathit{live}(\langle s, n\rangle, \bar{x}z, \bar{x}z_{\langle s,n\rangle}.S)} \qquad \text{Res}\quad \frac{\mathit{live}(\langle s, n\rangle, \mu, S) \quad y \notin n(\mu)}{\mathit{live}(\langle s, n\rangle, \mu, (\nu y)S)}$$

$$\text{Open}\quad \frac{\mathit{live}(\langle s, n\rangle, \bar{x}y, S) \quad x \neq y}{\mathit{live}(\langle s, n\rangle, \bar{x}(y), (\nu y)S)} \qquad \text{Rep}\quad \frac{S \xrightarrow{\mu} S'}{\mathit{live}(\langle s, n\rangle, \mu, !_{\langle s,n\rangle}S)}$$

$$\text{Par}\quad \frac{\mathit{live}(\langle s, n\rangle, \mu, S_0) \quad bn(\mu) \cap fn(S_1) = \emptyset}{\mathit{live}(\langle s, n\rangle, \mu, (S_0 \mid S_1))}$$

$$\text{Com}\,\frac{\mathit{live}(\langle s, n\rangle, xy, S_0), \quad \mathit{live}(\langle r, m\rangle, \bar{x}y, S_1)}{\mathit{live}(\langle s, n\rangle\,, \tau, S_0 \mid S_1),\ \mathit{live}(\langle r, m\rangle\,, \tau, S_0 \mid S_1)}$$

$$\text{Close}\,\frac{\mathit{live}(\langle s, n\rangle, xy, S_0), \quad \mathit{live}(\langle r, m\rangle, \bar{x}(y), S_1), \quad y \notin fn(S_0)}{\mathit{live}(\langle s, n\rangle\,, \tau, (\nu y)(S_0 \mid S_1)),\ \mathit{live}(\langle r, m\rangle\,, \tau, (\nu y)(S_0 \mid S_1))}$$

Table 8: Live labels.

We can now formally define the strong and weak notions of fairness. Intuitively, a *weak-fair* computation is a maximal computation such that no label becomes live and then stays live forever.

Definition 5.2. (*Weak-fair Computations*) Given $S \in \mathcal{E}^e$, a *weak-fair computation* from S is a maximal computation,

$$S = S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} S_2 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S_i\ [\xrightarrow{\tau} \ldots]$$

where $\forall v \in (\{0,1\}^* \times \mathbb{N}).\ \forall i \geq 0.\ \exists j \geq i.\ v \notin Ll(S_j)$.

A *strong-fair* computation is a maximal computation such that no label is live infinitely often. Formally, strong fairness imposes that for every label there is some point beyond which it is never live.

Definition 5.3. (*Strong-fair Computations*) Given $S \in \mathcal{E}^e$, a *strong-fair computation* from S is a maximal computation,

$$S = S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} S_2 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S_i\ [\xrightarrow{\tau} \ldots]$$

where $\forall v \in (\{0,1\}^* \times \mathbb{N}).\ \exists i \geq 0.\ \forall j \geq i.\ v \notin Ll(S_j)$.

Note that every finite computation is *strong-fair* (resp. *weak-fair*), because there is no transition $\xrightarrow{\tau}$ from the end state, which implies that there are no live labels.

Some useful results follow:

Theorem 5.4. $\forall S \in \mathcal{E}^e$.

1. *there is always a strong-fair computation from S, and*
2. *every strong-fair computation from S is weak-fair, but not vice versa.*

Proof.

(1) We apply items of Lemma C.1. If $S \not\xrightarrow{\tau}$, then the empty computation is *strong-fair*, since $Ll(S) = \emptyset$. Otherwise, there is a maximal computation $\mathcal{C}$

$$S = S_0 \xrightarrow{\tau} S_0^1 \xrightarrow{\tau} .. \xrightarrow{\tau} S_0^{n_0} \xrightarrow{\tau} S_1 \, [\xrightarrow{\tau} S_1^1 \xrightarrow{\tau} .. \xrightarrow{\tau} S_1^{n_1} \xrightarrow{\tau} S_2 \xrightarrow{\tau} \ldots]$$

where $\forall i \geq 0.\ Ll(S_i) \cap Ll(S_{i+1}) = \emptyset$ and $\forall j \geq i.\ Ll(S_i) \cap Ll(S_j) = \emptyset$. Suppose, by contradiction, that $\mathcal{C}$ is not *strong-fair*: then there exists a label v such that $\forall i \geq 0.$ $\exists j \geq i.\ v \in Ll(\tilde{S})$, where either $\tilde{S} = S_j$ or $\tilde{S} = S_j^k$, contradicting the hypothesis on $\mathcal{C}$.

(2) The positive result is trivial: by definition, a *strong-fair* computation is a special case of *weak-fair* computation. To prove the negative result, let $S = E \mid \rho$, where $E = !_{v_1^0} a \mid (\nu b)(\bar{b}_{v_2^0} \mid !_{v_3^0} b.(\bar{a} \mid \bar{b}))$ and $\rho = a_{v_4}.\omega$: it is not difficult to check that there exists a maximal computation from S, along which a_{v_4} is never performed. The maximal computation $\mathcal{C}$ we consider is the following one (we omit 0 term by convenience):

$$E \mid \rho = S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} S_2 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S_i \xrightarrow{\tau}$$

where $\forall j \geq 0.\ Q(v_2^j, v_3^j) = (\nu b)(\bar{b}_{v_2^j} \mid !_{v_3^j} b.(\bar{a} \mid \bar{b}))$ and

$$\begin{array}{ll}
S_0 = !_{v_1^0} a \mid Q(v_2^0, v_3^0) \mid a_{v_4}.\omega & \ldots \\
S_1 = !_{v_1^0} a \mid \bar{a}_{v_5^1} \mid Q(v_2^1, v_3^1) \mid a_{v_4}.\omega & S_i = !_{v_1^i} a \mid Q(v_2^{i-1}, v_3^{i-1}) \mid a_{v_4}.\omega \\
S_2 = !_{v_1^2} a \mid Q(v_2^1, v_3^1) \mid a_{v_4}.\omega & S_{i+1} = !_{v_1^i} a \mid \bar{a}_{v_5^{i+1}} \mid Q(v_2^{i+1}, v_3^{i+1}) \mid a_{v_4}.\omega \\
S_3 = !_{v_1^2} a \mid \bar{a}_{v_5^3} \mid Q_2(v_2^3, v_3^3) \mid a_{v_4}.\omega & S_{i+2} = !_{v_1^{i+2}} a \mid Q(v_2^{i+1}, v_3^{i+1}) \mid a_{v_4}.\omega \\
S_4 = !_{v_1^4} a \mid Q(v_2^3, v_3^3) \mid a_{v_4}.\omega & \ldots
\end{array}$$

Notice that, in $\mathcal{C}$, we have $v_4 \notin Ll(S_0), v_4 \in Ll(S_1), v_4 \notin Ll(S_2), v_4 \in Ll(S_3), \ldots, v_4 \notin Ll(S_i), v_4 \in Ll(S_{i+1}), v_4 \notin Ll(S_{i+2}), \ldots$ and so on. Moreover for every $v \in Ll(S_j)$, where $v \neq v_4$, there exists $k > j$ such that $v \notin Ll(S_k)$. I.e., $\mathcal{C}$ is *weak-fair* but it is not *strong-fair*. □

6. Comparing 'fair'-testing semantics

In this section we consider the addition of the requirement of fairness in the definition of the *must*-testing and investigate the resulting semantic relations. In particular, we compare the different notions of fairness (the notions we introduce and the existing notion of *fair*-testing semantics), and the *must*-testing semantics.

Let us start by observing that P *must* o implies P *fair* o, but not vice versa: it suffices to consider the process $P = (\nu b)(\bar{b} \mid !b.\bar{b}) \mid \bar{a}$ and the observer $o = a.\omega$.

Now, we define our notions of 'fair' *must*-testing.

Definition 6.1. (*Strong/Weak-fair Must Semantics*) Let $E \in \mathcal{P}^e$ and $\rho \in \mathcal{O}^e$. Define E *sfmust* ρ (E *wfmust* ρ) if and only if *for every strong-fair* (respectively, *weak-fair*) computation from $(E \mid \rho)$

$$E \mid \rho = S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S_i \, [\xrightarrow{\tau} \ldots]$$

there exists some $i \geq 0$ such that $S_i \xrightarrow{\omega}$.

The following result states the relation between *weak-fair must*-testing and *strong-fair must*-testing. It is the case that *weak-fair must*-testing implies *strong-fair must*-testing, but not vice versa. In fact, any *strong-fair* computation is also *weak-fair*. To prove the negative result, we consider an experiment with *weak-fair* computation in which the label prefixing ω becomes live, loses its liveness, becomes live again, etc., without being performed: this computation is *weak-fair* by definition and unsuccessful. Notice that this label should be always performed in a *strong-fair* computation, determining the success of it.

Proposition 6.2. $\forall E \in \mathcal{P}^e$. $\forall \rho \in \mathcal{O}^e$.

E wfmust ρ implies E sfmust ρ, but not vice versa.

Proof. For the positive part, suppose, by contradiction, that there exists a *strong-fair* computation $\mathcal{C}$

$$E \mid \rho = S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S_i \, [\xrightarrow{\tau} \ldots]$$

such that $\forall i \geq 0$. $S_i \not\xrightarrow{\omega}$. Since a *strong-fair* computation is *weak-fair* too, then $\mathcal{C}$ is *weak-fair*. It follows that E *wf~~m~~ust* ρ, thus contradicting the hypothesis.

We now prove the negative result. Consider again $E = !_{v_1^0} a \mid Q(v_2^0, v_3^0)$ and $\rho = a_{v_4}.\omega$, where $Q(v_2^j, v_3^j) = (\nu b)(\bar{b}_{v_2^j} \mid !_{v_3^j} b.(\bar{a} \mid \bar{b}))$.

Notice that the computation proposed in the proof of item (2) of Theorem 5.4, where $v_4 \notin Ll(S_0), v_4 \in Ll(S_1), v_4 \notin Ll(S_2)$, $v_4 \in Ll(S_3), .., v_4 \notin Ll(S_j), v_4 \in Ll(S_{j+1})$, $v_4 \notin Ll(S_{j+2})$ etc., is unsuccessful: in fact, v_4 loses its liveness even if a_{v_4} is not performed. In such a case $\forall j \geq 0$. $S_j \not\xrightarrow{\omega}$. It follows that E *wf~~m~~ust* ρ.

To prove that E *sfmust* ρ holds, it suffices to notice that for every $j \geq 0$ and every $v_2^j, v_3^j \in (\{0,1\}^* \times \mathbb{N})$,

(1) $Q(v_2^j, v_3^j) \xrightarrow{\tau} \bar{a}_{v_5^{j+1}} \mid Q_2(v_2^{j+1}, v_3^{j+1})$, i.e. $Q(v_2^j, v_3^j)$ can perform infinite $\xrightarrow{\tau}$ sequences;

(2) for every $T \in \mathcal{E}^e$, every $\xrightarrow{\tau}$ from $(Q(v_2^j, v_3^j) \mid T)$ does not follow from a synchronization (either Rule Com or Close) between $Q(v_2^j, v_3^j)$ and T;

(3) for every maximal computation $\mathcal{C}'$ from $E \mid \rho$

$$E \mid \rho = S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S_i \, [\xrightarrow{\tau} \ldots]$$

there always exists

$$S_1 = !_{v_1^0} a \mid \bar{a}_{v_5^1} \mid Q(v_2^1, v_3^1) \mid a_{v_4}.\omega.$$

(4) $v_4 \notin Ll(S_0)$, $v_4 \in Ll(S_1)$ and $v_4 \in Ll(S_{j+1})$ whenever there exists $k \geq (j+1)$ such that $\bar{a}_{v_5^k}$ is a top-level parallel component of S_{j+1}.

By definition of $Q(v_2^j, v_3^j)$, there exist infinitely many indexes k such that $\bar{a}_{v_5^k}$ is a top-level parallel component of S_{j+1}; it follows that v_4 can be live infinitely often. But this is not possible if $\mathcal{C}'$ is a *strong-fair* computation: in fact, by definition, v_4 will lose its liveness forever, i.e. a_{v_4} will be performed. In such a case there will be $i \geq 2$ in $\mathcal{C}'$ such that $S_i \xrightarrow{\omega}$. $\square$

Proposition 6.3 shows the relation between *strong-fair must-* (respectively, *weak-fair must-*) testing semantics and *must*-testing.

Proposition 6.3. $\forall E \in \mathcal{P}^e.\ \forall \rho \in \mathcal{O}^e.$
1. $Unl(E)\ must\, Unl(\rho)$ *implies* E *wfmust* ρ*, but not vice versa;*
2. $Unl(E)\ must\, Unl(\rho)$ *implies* E *sfmust* ρ*, but not vice versa.*

Proof.
(1) For the positive part, suppose there is a *weak-fair* computation from $E \mid \rho$

$$E \mid \rho = S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} S_i\ [\xrightarrow{\tau} \dots]$$

such that $\forall i \geq 0.\ S_i \not\xrightarrow{\omega}$. Then there exists the following maximal computation

$$Unl(E \mid \rho) = Unl(S_0) \xrightarrow{\tau} Unl(S_1) \xrightarrow{\tau} \dots \xrightarrow{\tau} Unl(S_i)\ [\xrightarrow{\tau} \dots]$$

where $\forall i \geq 0.\ Unl(S_i) \not\xrightarrow{\omega}$, i.e. $Unl(E)\ \not{m}ust\, Unl(\rho)$.

We now prove the negative part. Let $E = (\nu b)(\bar{b}_{v_1^0} \mid !_{v_2^0} b.\bar{b}) \mid \bar{a}_{v_3}$ and $\rho = a_{v_4}.\omega$, we have $Unl(E)\ \not{m}ust\, Unl(\rho)$. However, in every *weak-fair* computation from $E \mid \rho$

$$E \mid \rho = S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} S_i \xrightarrow{\tau} \dots$$

there must exist $j \geq 0$ such that $S_{j+1} = (\nu b)(\bar{b}_{v_1^j} \mid !_{v_2^j} b.\bar{b}) \mid \omega \xrightarrow{\omega}$ and $\forall i \in [0..j].\ S_i = (\nu b)(\bar{b}_{v_1^i} \mid !_{v_2^i} b.\bar{b}) \mid \bar{a}_{v_3} \mid a_{v_4}.\omega$. It follows by the fact that $\forall i \in [0..j].\ v_4 \in Ll(S_i)$ and there must exist $k > i$ $(k = j+1)$ such that $v_4 \notin Ll(S_k)$. It is possible only in the case $a_{v_4}.\omega$ synchronizes with $\bar{a}_{v_3}$ in $S_{k-1} = (\nu b)(\bar{b}_{v_1^k} \mid !_{v_2^k} b.\bar{b}) \mid \bar{a}_{v_3} \mid a_{v_4}.\omega$.

(2) Immediate consequence of item (1) and Proposition 6.2. □

7. Fair-testing and 'fair'-testing semantics

In [35] it is shown that *fair*-testing semantics on finite state systems corresponds to some (strong) notion of fairness. However, this result does not hold in general. We will show that *strong-fair must*-testing (and hence *weak-fair must*-testing) does not suffice to characterize *fair*-testing.

The reason behind the negative result relies on the fact that we can construct a term for which there exist experiments being successful under *fair*-testing and performing maximal unsuccessful computations which are strong fair.

Theorem 7.1. $\forall E \in \mathcal{P}^e.\ \forall \rho \in \mathcal{O}^e.$
1. E *sfmust* ρ *implies* $Unl(E)\ fair\, Unl(\rho)$*, but not vice versa;*
2. E *wfmust* ρ *implies* $Unl(E)\ fair\, Unl(\rho)$*, but not vice versa.*

Proof.
(1) For the positive result, suppose, by contradiction, there exists a maximal computation from $Unl(E) \mid Unl(\rho)$

$$Unl(E) \mid Unl(\rho) = T_0 \xrightarrow{\tau} T_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} T_i\ [\xrightarrow{\tau} \dots]$$

and there exists $i \geq 0$ such that $T_i \not\overset{\omega}{\Longrightarrow}$, i.e. for each T' such that $T_i \overset{\varepsilon}{\Longrightarrow} T'$, we have $T' \not\xrightarrow{\omega}$. It follows that for every maximal computation from T_i of the form

$$T_i = T'_0 \xrightarrow{\tau} T'_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} T'_j\ [\xrightarrow{\tau} \dots]$$

$T'_j \not\xrightarrow{\omega}$ for every j. Since ω cannot synchronize, it does not disappear once it is at the top level of a term. It implies that $\forall j \in [0..(i-1)].\, T_j \not\xrightarrow{\omega}$. Now, consider the computation

$$E \mid \rho = S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S_i \, [\xrightarrow{\tau} \ldots]$$

where for every $k \geq 0$ we have $T_k = Unl(S_k)$. Then there exists $i \geq 0$ such that $S_i \not\overset{\omega}{\Longrightarrow}$, i.e. for each S' such that $S_i \overset{\varepsilon}{\Longrightarrow} S'$, we have $S' \not\xrightarrow{\omega}$. It follows that for any maximal computation from S_i

$$S_i = S'_0 \xrightarrow{\tau} S'_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S'_j \, [\xrightarrow{\tau} \ldots]$$

$S'_j \not\xrightarrow{\omega}$ for every j. Hence for every *strong-fair* computation from S_i (which always exists, by Theorem 5.4)

$$S_i = S'_0 \xrightarrow{\tau} S'_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S'_j \, [\xrightarrow{\tau} \ldots]$$

$S'_j \not\xrightarrow{\omega}$ for every j. It follows that, given a *strong-fair* computation from S_i

$$S_i = S''_0 \xrightarrow{\tau} S''_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S''_j \, [\xrightarrow{\tau} \ldots]$$

where $S''_j \not\xrightarrow{\omega}$ for every j, the following maximal computation

$$E \mid \rho = S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S_i = S'_0 \xrightarrow{\tau} S''_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S''_j \, [\xrightarrow{\tau} \ldots]$$

is *strong-fair* (by Lemma C.2), and $\forall k \in [0..(i-1)].\, S_k \not\xrightarrow{\omega}$, and $\forall j \geq 0.\, S''_j \not\xrightarrow{\omega}$. It follows that $E \not\mathit{sf\text{-}must}\, \rho$, contradicting the hypothesis.

We now prove the negative part. As explained before, it suffices to consider $E = \bar{c}_{v_1^0} \mid !_{v_2^0} c.((\nu b)(\bar{b} \mid b.\bar{c} \mid b.\bar{a}))$ and $\rho = a_{v_3}.\omega$. Clearly, $Unl(E)$ *fair* $Unl(\rho)$, but there exists the following maximal computation

$$\begin{aligned}
E \mid \rho = \bar{c}_{v_1^0} \mid !_{v_2^0} c.((\nu b)(\bar{b} \mid b.\bar{c} \mid b.\bar{a})) \mid a_{v_3}.\omega &\xrightarrow{\tau} \\
(\nu b)(\bar{b}_{v_4^1} \mid b_{v_5^1}.\bar{c}_{v_6^1} \mid b_{v_7^1}.\bar{a}_{v_8^1}) \mid !_{v_2^1} c.((\nu b)(\bar{b} \mid b.\bar{c} \mid b.\bar{a})) \mid a_{v_3}.\omega &\xrightarrow{\tau} \\
(\nu b)(b_{v_7^1}.\bar{a}_{v_8^1}) \mid \bar{c}_{v_6^1} \mid !_{v_2^1} c.((\nu b)(\bar{b} \mid b.\bar{c} \mid b.\bar{a})) \mid a_{v_3}.\omega &\xrightarrow{\tau} \\
\ldots &\xrightarrow{\tau} \\
\prod_{i \in [1..k]} (\nu b)(b_{v_7^i}.\bar{a}_{v_8^i}) \mid \bar{c}_{v_6^k} \mid !_{v_2^k} c.((\nu b)(\bar{b} \mid b.\bar{c} \mid b.\bar{a})) \mid a_{v_3}.\omega &\xrightarrow{\tau} \\
\ldots &\xrightarrow{\tau}
\end{aligned}$$

where no term has ω enabled. Notice that ω is always prefixed in $a_{v_3}.\omega$ and v_3 is always disabled since every occurrence of $\bar{a}_{v_8^i}$ is prefixed in a deadlock term $(\nu b)(b_{v_7^i}.\bar{a}_{v_8^i})$. Hence this computation is *strong-fair*.

(2) The positive part is an immediate consequence of item (1) and Proposition 6.2. As for the negative part, observe that the counterexample in the proof of item (1) is a counterexample here too, because the computation considered is also *weak-fair*. □

Previous result establishes that the notion of *weak*- and *strong-fair must*-testing differ from the notion of *fair*-testing in literature. A natural question is, then, which notion is more suitable than the other in given situations. As shown by the counterexample in the proof of previous theorem, the difference is with respect to computations that are fair but unsuccessful, and they offer at every state the possibility of being successful. These computations are considered acceptable by the notion of *fair*-testing, but not by our notion, and in our opinion, they should not be.

Example 7.2. We illustrate the difference with the well-known example of the dining philosophers. We can specify the system in our language in the following way. The system, DP, is composed by three forks $\bar{f}_0, \bar{f}_1, \bar{f}_2$ and three philosophers P_0, P_1, P_2, in parallel:

$$DP \stackrel{\text{def}}{=} \bar{f}_0 \mid P_0 \mid \bar{f}_1 \mid P_1 \mid \bar{f}_2 \mid P_2$$

Each philosopher replicates the following activity: first, he chooses whether to start with the left fork (if available) or with the right fork (if available). For the choice we use the input-guarded choice construct, represented here by the operator $+$. It is well-known that this kind of choice can be expressed in the asynchronous π-calculus, and therefore also in the language that we consider here, by a translation that preserves must semantics [28].

$$P_i \stackrel{\text{def}}{=} !\,(L_i + R_i)$$

Under the left choice the philosopher takes the left fork, then chooses whether to take the right fork (if available) or to give up. In the first case, he takes the fork, eats, and then releases both forks. In the second case, he releases the left fork. This behavior can be represented as follows (where $\oplus$ denotes summation modulo 3):

$$L_i \stackrel{\text{def}}{=} f_i.(f_{i\oplus 1}.\overline{eat}.(\bar{f}_i \mid \bar{f}_{i+1}) \;+\; \tau.\bar{f}_i)$$

The behavior under the right choice is analogous:

$$R_i \stackrel{\text{def}}{=} f_{i\oplus 1}.(f_i.\overline{eat}.(\bar{f}_{i\oplus 1} \mid \bar{f}_i) \;+\; \tau.\bar{f}_{i\oplus 1})$$

Let us consider the observer which detects whether one philosopher succeeds to eat:

$$o \stackrel{\text{def}}{=} eat\,.\,\omega$$

We can see that

$$DP \;\; fair \;\; o$$

In fact, in every computation either a philosopher succeeds in taking both forks, and in that case he eats and the observer is satisfied, or there is always the possibility that one fork becomes available and can be taken by a philosopher who has already another fork. On the other hand, the computation in which each philosopher in turn takes the right fork, releases it, then take the second fork, releases it, then take the right fork ... etc. is strongly fair, and unsuccessful. Hence we have

$$DP \;\; \not{sf}must \;\; o$$

The answer given by our semantics is consistent with the view in Distributed Computing, where fairness and progress (a generalization of success - in this case, the fact that someone will eventually eat) are distinct concepts, and the Dining Cryptographers are considered an example of the fact that the first (fairness) does not imply the latter (progress).

The difference between *fair*-testing and both *weak*- and *strong-fair must*-testing relies on the fact that the former is based on properties of the transition tree and the latter are based on the notion of fairness.

We will prove in fact that no notion based only on the transition tree can characterize *strong-fair must*- and *weak-fair must*-testing. To this purpose, let us recall the definition of (strong) *bisimulation*.

Definition 7.3. (*Bisimulation*) A *bisimulation* is a binary relation $\mathcal{R}$ satisfying the following: $P \,\mathcal{R}\, Q$ implies that:

1. $P \xrightarrow{\mu} P'$ then $\exists Q' : Q \xrightarrow{\mu} Q' \wedge P' \,\mathcal{R}\, Q'$;
2. $Q \xrightarrow{\mu} Q'$ then $\exists P' : P \xrightarrow{\mu} P' \wedge P' \,\mathcal{R}\, Q'$.

Bisimilarity $\sim$ is the largest bisimulation $\mathcal{R}$ such that $P \,\mathcal{R}\, Q$.

We recall that bisimilarity is a congruence.

We now prove that *sfmust* and *wfmust* cannot be characterized by a notion that, like *fair*-testing, relies on the transition tree only.

Theorem 7.4. *$\exists E, F \in \mathcal{P}^e.\ Unl(E) \sim Unl(F)$ but $E \not\approx_{sat} F$, where $sat \in \{$wfmust, sfmust$\}$.*

Proof. Let

$$E = (\nu c)(\bar{c}_{w_0^0} \mid !_{w_1^0} c.(\bar{c} \mid \bar{a})) \ \mid\ (\nu c)(\bar{c}_{w_2^0} \mid !_{w_3^0} c.\bar{c})$$

and

$$F = !_{v_0^0}((\nu b)(\bar{b} \mid b \mid b.\bar{a})) \mid (\nu c)(\bar{c}_{v_1^0} \mid !_{v_2^0} c.\bar{c}).$$

E and F are neither *sfmust* nor *wfmust* equivalent, since the observer $\rho = a_{v_3}.\omega$ distinguishes E and F w.r.t. both *sfmust* and *wfmust*. In fact, every *strong-fair* (respectively, *weak-fair*) computation from $E \mid \rho$ forces the synchronization between $\bar{c}_{w_0^0}$ and $!_{w_1^0} c.(\bar{c} \mid \bar{a})$, i.e. the transition $(\nu c)(\bar{c}_{w_0^0} \mid !_{w_1^0} c.(\bar{c} \mid \bar{a})) \xrightarrow{\tau} \bar{a}_{w_4^1} \mid (\nu c)(\bar{c}_{w_0^1} \mid !_{w_1^1} c.(\bar{c} \mid \bar{a}))$ and it also forces the execution of $\bar{a}_{w_4^1}$ (or equivalently of $\bar{a}_{w_4^i}$ for some $i \geq 1$ such that $(\nu c)(\bar{c}_{w_0^{i-1}} | !_{w_1^{i-1}} c.(\bar{c}|\bar{a})) \xrightarrow{\tau} \bar{a}_{w_4^{i-1}} \mid (\nu c)(\bar{c}_{w_0^{i-1}} \mid !_{w_1^{i-1}} c.(\bar{c} \mid \bar{a}))$ occurred in the computation).

It follows that there exists a transition in which a_{v_3} is performed, implying that there exists a term which has ω enabled.

This is not the case of the following *strong-fair* (and *weak-fair*) computation from $F \mid \rho$:

$$\begin{aligned}
F \mid \rho = & !_{v_0^0}((\nu b)(\bar{b} \mid b \mid b.\bar{a})) \mid (\nu c)(\bar{c}_{v_1^0} \mid !_{v_2^0} c.\bar{c}) \mid a_{v_3}.\omega \stackrel{\varepsilon}{\Longrightarrow} \\
& (\nu b)(b_{v_4^1}.\bar{a}_{v_5^1}) \mid !_{v_0^1}((\nu b)(\bar{b} \mid b \mid b.\bar{a})) \mid (\nu c)(\bar{c}_{v_1^1} \mid !_{v_2^1} c.\bar{c}) \mid a_{v_3}.\omega \stackrel{\varepsilon}{\Longrightarrow} \\
& (\nu b)(b_{v_4^1}.\bar{a}_{v_5^1}) \mid (\nu b)(b_{v_4^2}.\bar{a}_{v_5^2}) \mid !_{v_0^2}((\nu b)(\bar{b} \mid b \mid b.\bar{a})) \mid (\nu c)(\bar{c}_{v_1^2} \mid !_{v_2^2} c.\bar{c}) \mid a_{v_3}.\omega \stackrel{\varepsilon}{\Longrightarrow} \\
& \ldots \stackrel{\varepsilon}{\Longrightarrow} \\
& \prod_{i \in [1..k]} (\nu b)(b_{v_4^k}.\bar{a}_{v_5^k}) \mid !_{v_0^k}((\nu b)(\bar{b} \mid b \mid b.\bar{a})) \mid (\nu c)(\bar{c}_{v_1^k} \mid !_{v_2^k} c.\bar{c}) \mid a_{v_3}.\omega \stackrel{\varepsilon}{\Longrightarrow} \\
& \ldots \stackrel{\varepsilon}{\Longrightarrow}
\end{aligned}$$

where there are no terms with ω enabled. Notice that ω is always prefixed in $a_{v_3}.\omega$ and a_{v_3} is always disabled since every occurrence of $\bar{a}_{v_5^i}$ is prefixed in a deadlock term $(\nu b)(b_{v_4^i}.\bar{a}_{v_5^i})$.

However $Unl(E) \sim Unl(F)$, implying that $(Unl(E) \mid Unl(\rho)) \sim (Unl(F) \mid Unl(\rho))$, for any observer ρ. $\square$

8. Conclusion and future work

We have designed a labeled version of the π-calculus, we have defined weak and strong fairness, and we have introduced the natural (weak and strong) fair versions of testing semantics. We have compared the various notions and proved that neither weak nor strong fairness correspond to *fair*-testing, and we have investigated the reason of this failure.

Our results are quite general, since they also hold for CCS, for the asynchronous π-calculus [3] (it is easy to see that all proofs can be adapted immediately to these other calculi), to a π-calculus with choice operator (as explained in the introduction), and they do not depend on the labeling method (i.e. they hold for any labeling method for which unicity, disappearance and persistence hold).

As a future work, we plan to investigate on the existence of alternative characterizations of the fairness notions, allowing simple and finite representations of fair computations such as the use of regular expressions as in [8, 9]. It is also interesting to investigate the impact that these different notions of fairness may have on the encodings from the π-calculus into the asynchronous π-calculus [7].

Another line of research that seems worth exploring is the the adaptation in our framework of the fairness notions of [18]. As we have mentioned in the introduction, it is possible to represent several forms of choice in the choiceless π-calculus using the parallel operator, and it would be interesting to see how the fairness notions of [18] relative to the choice operator get translated in our formalism.

Acknowledgemnent

We wish to thank the anonymous reviewers for their valuable comments and suggestion which helped to improve the paper in a substantial way.

References

[1] Agha, G., Mason, I. A., Smith, S. & Talcott, C. L., *A Foundation for Actor Computation*, JFP, **7**(1) (1997), 1-72.

[2] Boreale, M. & De Nicola, R., *Testing Equivalence for Mobile Processes*, Information and Computation, **120** (1995), 279-303.

[3] Boudol, G., *Asynchrony and the π-calculus'*, Technical Report 1702, INRIA, Sophia-Antipolis (1992).

[4] Brinksma, E., *Cache Consistency by Design*, In *Protocols Specification, Testing and Verification* (VIII), Aggarwal & Sabnani (1988), 63-74.

[5] Brinksma, E., *A theory for the Derivation of Tests*, In "Protocols Specification, Testing and Verification" (XIV), Chapman & Hall (1995), 53-67.

[6] Brinksma. E., Rensink, A. & Vogler, W., *Applications of Fair Testing*, In *Protocols Specification, Testing and Verification* (XVI), Chapman & Hall (1996), 145-160.

[7] Cacciagrano, D., Corradini, F. & Palamidessi, C., *Separation of Synchronous and Asynchronous Communication Via Testing*, Theoretical Computer Science, **386**(3) (2007), 218-235.

[8] Corradini, F., Di Berardini, M.R. & Vogler, W., *Fairness of Actions in System Computations*, Acta Informatica, **43**(2) (2006), 73-130.

[9] Corradini, F., Di Berardini, M.R. & Vogler, W., *Fairness of Components in System Computations*, Theoretical Computer Science, **356**(3) (2006), 291-324.

[10] Costa, G. & Stirling, C., *A Fair Calculus of Communicating Systems*, Acta Informatica, **21** (1984), 417-441.

[11] Costa, G. & Stirling, C., *Weak and Strong Fairness in CCS*, Information and Computation, **73** (1987), 207-244.

[12] De Nicola, R. & Hennessy, M., *Testing Equivalences for Processes*, Theoretical Computer Science, **34** (1984), 83-133.
[13] Fournet, C. & Gonthier, G., *A Hierarchy of Equivalences for Asynchronous Calculi*, Proc. of ICALP'98 (1998), 844-855.
[14] Fournet, C. & Gonthier, G., *The Join Calculus: A Language for Distributed Mobile Programming*, Proc. of APPSEM 2000, LNCS, **2395** (2000), 268-332.
[15] Francez, N., *Fairness*, Springer-Verlag (1986).
[16] Hennessy, M., *Acceptance trees*, JACM, **32**(4) (1985), 896-928.
[17] Hennessy, M., *An Algebraic Theory of Fair Asynchronous Communicating Processes*, Theoretical Computer Science, **49** (1987), 121-143.
[18] Kuiper, R. & de Roever, W. P., *Fairness assumptions for CSP in a temporal logic framework*, Proc. of IFIP Working Conference on Formal Description of Programming Concepts (1983), 159-167.
[19] Honda, K. & Tokoro, M., *An Object calculus for Asynchronous Communication*, Proc. of ECOOP '91, LNCS, **512** (1991), 133-147.
[20] Honda, K. & Yoshida, N., *Replication in Concurrent Combinators*, Proc. of TACS '94, LNCS, **789** (1994).
[21] Koomen, C., *Albegraic Specification and Verification of Communications protocols*, Science of Computer Programming, **5** (1985), 1-36.
[22] Larsen, K. G. & Milner, R., *Verifying a Protocol using Relativized Bisimulation*, LNCS, **267** (1987), 126-135.
[23] Lehmann, D., Pnueli, A. & Stavi, J., *Impartiality, justice and Fairness:the Ethics of Concurrent Termination*, Proc. of 8th Int. Colloq. Aut. Lang. Prog., LNCS, **115** (1981), 264-277.
[24] Milner, R., *Communication and Concurrency*, Prentice-Hall International (1989).
[25] Milner, R., Parrow, J. & Walker, D., *A Calculus of Mobile Processes*, Part I and II, Information and Computation, **100** (1992), 1-78.
[26] Natarajan, V. & Cleaveland, R., *Divergence and Fair Testing*, Proc. of ICALP '95, LNCS, **944** (1995), 648-659.
[27] Nestmann, U. What is a 'good' encoding of guarded choice? *Journal of Information and Computation*, 156:287–319, 2000.
[28] Nestmann, U. and Pierce, B. C. Decoding choice encodings. *Journal of Information and Computation*, 163:1–59, 2000.
[29] Núñez, M. & Rupérez, D., *Fair testing through probabilistic testing*, Protocol Specification, Testing, and Verification, **19** (1999), 135-150.
[30] Palamidessi, C. *Comparing the Expressive Power of the Synchronous and Asynchronous π-calculus*, Mathematical Structures in Computer Science, **13**(5), pp. 685-719, 2003.
[31] Park, D. M. R., *Concurrency and Automata on Infinite Sequences*, LNCS, **104** (1980).
[32] Pnueli, A., *On the Extremely Fair Treatment of Probabilistic Algorithms*, Proc. of ACM Symph. Theory of Comp. (1983), 278-290.
[33] Pierce, B. C. & Turner, D. N., *Pict: A Programming Language Based on the Pi-Calculus*, in *Proof, Language and Interaction: Essays in Honour of Robin Milner*, MIT Press (2000).
[34] Queille, J.P. & Sifakis, J., *Fairness and Related Properties in Transition Systems-A Temporal Logic to Deal with Fairness*, Acta Informatica, **19** (1983), 195-210.
[35] Rensink, A. & Vogler, W., *Fair Testing*, Information and Computation, **205** (2007), 125-198. A short version of this paper appeared in the Proc. of CONCUR95, LNCS, 962 (1995), 313-327.
[36] Sangiorgi, D., *On the Bisimulation Proof Method*, JMSCS, **8** (1998), 447479.
[37] Sangiorgi, D. & Walker, D., *The Pi-calculus: a Theory of Mobile Processes*, Cambridge University Press (2001).
[38] Vogler, W., *Modular Construction and Partial Order Semantics of Petri Nets*, LNCS, **625** (1992).

Appendix A. A labeled version of the π-calculus

This appendix section contains intermediate results and proofs of the statements omitted in Section 4. Several proofs follow the same lines as the corresponding results in [11].

Lemma A.1. *Let $r_0, r_1, s \in \{0,1\}^*$. $r_0 \sqsubseteq s$ and $r_1 \sqsubseteq s$. Then either $r_0 \sqsubseteq r_1$ or $r_1 \sqsubseteq r_0$.*

Proof. For $i \in \{0,1\}$, $r_i \sqsubseteq s$ implies $s = r_i\alpha_i$ for some $\alpha_i \in \{0,1\}^*$. Then $r_0\alpha_0 = s = r_1\alpha_1$. Let $|r_i|$ the length of r_i. If $|r_0| \leq |r_1|$, then $r_0 \sqsubseteq r_1$. Otherwise, $r_1 \sqsubseteq r_0$. □

Lemma A.2. *Let $\langle r_0, m_0\rangle, \langle r_1, m_1\rangle$, $\langle r'_0, m'_0\rangle, \langle r'_1, m'_1\rangle \in (\{0,1\}^* \times \mathbb{N})$. $r_0 \sqsubseteq r'_0$, $r_1 \sqsubseteq r'_1$ and $\{\langle r_0, m_0\rangle\}\Re\{\langle r_1, m_1\rangle\}$. Then $\{\langle r'_0, m'_0\rangle\}\Re\{\langle r'_1, m'_1\rangle\}$.*

Proof. For $i \in \{0,1\}$, $r_i \sqsubseteq r'_i$ implies $r'_i = r_i\alpha_i$ for some $\alpha_i \in \{0,1\}^*$. By contradiction, suppose $r'_0 \sqsubseteq r'_1$ (the other case is similar). Then $r_0\alpha_0 \sqsubseteq r_1\alpha_1$. Let $|r_i|$ the length of r_i. In the case $|r_0| \leq |r_1|$, then $r_0 \sqsubseteq r_1$, contradicting $\{\langle r_0, m_0\rangle\}\Re\{\langle r_1, m_1\rangle\}$. In the case $|r_1| \leq |r_0|$, then $r_1 \sqsubseteq r_0$, contradicting again $\{\langle r_0, m_0\rangle\}\Re\{\langle r_1, m_1\rangle\}$. □

Lemma A.3. *Let $E = L_{\langle r,m\rangle}(P)$, for some $P \in \mathcal{P}$. $\forall\langle s, n\rangle \in lab(E)$. $r \sqsubseteq s$ and $m \leq n$.*

Proof. By induction on the structure of P.

- $E = 0$: then $lab(0) = \emptyset$;
- $E = L_{\langle r,m\rangle}(\mu.P)$: then $lab(E) = \{\langle r, m\rangle\} \cup lab(L_{\langle r,m+1\rangle}(P))$.
- $E = L_{\langle r,m\rangle}(P_0 \mid P_1)$: $lab(L_{\langle r,m\rangle}(P_0 \mid P_1)) = lab(L_{\langle r0,m\rangle}(P_0)) \cup lab(L_{\langle r1,m\rangle}(P_1))$. By induction, $\forall\langle s_0, n_0\rangle \in lab(L_{\langle r0,m\rangle}(P_0))$. $r \sqsubseteq r0 \sqsubseteq s_0$ and $m \leq n_0$. Analogously, $\forall\langle s_1, n_1\rangle \in lab(L_{\langle r1,m\rangle}(P_1))$. $r \sqsubseteq r1 \sqsubseteq s_1$ and $m \leq n_1$.
- $E = !_{\langle r,m\rangle}P$: then $lab(E) = \{\langle r, m\rangle\}$.
- Case $E = L_{\langle r,m\rangle}((\nu x)P)$ can be proved similarly. □

Lemma A.4. *$\forall P \in \mathcal{P}. \forall\langle r, m\rangle \in (\{0,1\}^* \times \mathbb{N})$. $wf(L_{\langle r,m\rangle}(P))$.*

Proof. By induction on the structure of P.

- $P = 0, \mu.P', !P'$: these cases are trivial.
- $P = P_0 \mid P_1$: then $L_{\langle r,m\rangle}(P_0 \mid P_1) = L_{\langle r0,m\rangle}(P_0) \mid L_{\langle r1,m\rangle}(P_1)$ and by Lemma A.3 on $top(L_{\langle ri,m\rangle}(P_i))$ we have that $\forall\langle s_i, n_i\rangle \in top(L_{\langle ri,m\rangle}(P_i))$. $ri \sqsubseteq s_i$ and $m \leq n_i$ ($i \in \{0,1\}$). Hence $top(L_{\langle r0,m\rangle}(P_0))\ \Re\ top(L_{\langle r1,m\rangle}(P_1))$.
- $P = (\nu x)P'$: $L_{\langle r,m\rangle}(P) = (\nu x)L_{\langle r,m\rangle}(P')$, where $wf(L_{\langle r,m\rangle}(P'))$. Hence $wf(L_{\langle r,m\rangle}(P))$. □

Lemma A.5. *Let $E \in \mathcal{P}^e$.*

1. *For any distinct $\langle r, m\rangle, \langle r', m'\rangle \in top(E)$. $\{\langle r, m\rangle\}\Re\{\langle r', m'\rangle\}$;*
2. *$\forall\langle s, n\rangle \in lab(E)$. $\exists\langle r, m\rangle \in top(E)$. $r \sqsubseteq s$ and $m \leq n$.*

Let $E' \in \mathcal{P}^e_{gr}$. $E \xrightarrow{\mu} E'$. Then:

3. *$\forall\langle r', m'\rangle \in top(E')$. $\exists\langle r, m\rangle \in top(E)$. $r \sqsubseteq r'$ and $m \leq m'$;*
4. *$E' \in \mathcal{P}^e$.*

Proof.

(1) By induction on the structure of E.
- $E = 0$: $top(0) = \emptyset$.
- $E = L_{\langle s,n\rangle}(\mu.P)$: then $top(E) = \{\langle s, n\rangle\}$.
- $E = (E_0 \mid E_1)$: since $wf(E_0 \mid E_1)$ then $top(E_0)\Re top(E_1)$. Moreover, by induction hypothesis, $\forall\langle r_0, m_0\rangle, \langle r'_0, m'_0\rangle \in top(E_0)$. $\{\langle r_0, m_0\rangle\}\Re\{\langle r'_0, m'_0\rangle\}$ and, similarly, $\forall\langle r_1, m_1\rangle, \langle r'_1, m'_1\rangle \in top(E_1)$. $\{\langle r_1, m_1\rangle\}\Re\{\langle r'_1, m'_1\rangle\}$.
- Case $E = (\nu x)E'$: it can be proved similarly.
- $E = !_{\langle s,n\rangle}P$: then $top(E) = \{\langle s, n\rangle\}$.

(2) By induction on the structure of E.
- $E = 0$: $top(0) = \emptyset$ and $lab(0) = \emptyset$.

- $E = L_{\langle s,n\rangle}(\mu.P)$: then $top(E) = \{\langle s,n\rangle\}$. By Lemma A.3 on $L_{\langle s,n\rangle}(\mu.P)$, $\forall\langle s',n'\rangle \in lab(L_{\langle s,n\rangle}(\mu.P))$. $s \sqsubseteq s'$ and $n \leq n'$.
- $E = (E_0 \mid E_1)$: By induction, $\forall\langle s_0,n_0\rangle \in lab(E_0)$. $\exists\langle r_0,m_0\rangle \in top(E_0)$. $r_0 \sqsubseteq s_0$ and $m_0 \leq n_0$. Analogously $\forall\langle s_1,n_1\rangle \in lab(E_1)$. $\exists\langle r_1,m_1\rangle \in top(E_1)$. $r_1 \sqsubseteq s_1$ and $m_1 \leq n_1$. It follows that $\forall\langle s',n'\rangle \in lab(E_0 \mid E_1) = lab(E_0) \cup lab(E_1)$. $\exists\langle r',m'\rangle \in top(E_0 \mid E_1) = top(E_0) \cup top(E_1)$. $r' \sqsubseteq s'$ and $m' \leq n'$.
- Case $E = (\nu x)E'$: it can be proved similarly.
- $E = !_{\langle s,n\rangle}P$: then $top(E) = \{\langle s,n\rangle\} = lab(E)$.

(3) By induction on the depth of $E \xrightarrow{\mu} E'$.
- *Rule Input/Output*: $E = L_{\langle s,n\rangle}(\mu.P) \xrightarrow{\mu} E' = L_{\langle s,n+1\rangle}(P')$ (either $P' = P$ or $P' = P\{z/y\}$). Then $top(L_{\langle s,n\rangle}(\mu.P)) = \{\langle s,n\rangle\}$. By Lemma A.3 on $L_{\langle s,n+1\rangle}(P')$, we have that $\forall\langle r',m'\rangle \in top(L_{\langle s,n+1\rangle}(P')) \subseteq lab(L_{\langle s,n+1\rangle}(P'))$. $s \sqsubseteq r'$ and $n+1 \leq m'$. It follows that $\forall\langle r',m'\rangle \in top(L_{\langle s,n+1\rangle}(P'))$. $s \sqsubseteq r'$ and $n < m'$.
- *Rule Par*: $E = (E_0 \mid E_1) \xrightarrow{\mu} (E_0' \mid E_1)$, where $bn(\mu) \cap fn(E_1) = \emptyset$ and $E_0 \xrightarrow{\mu} E_0'$. Since $wf(E_0 \mid E_1)$, then $top(E_0) \Re top(E_1)$. By induction, $E_0 \xrightarrow{\mu} E_0'$ implies that $\forall\langle r_0',m_0'\rangle \in top(E_0')$. $\exists\langle r_0,m_0\rangle \in top(E_0)$. $r_0 \sqsubseteq r_0'$ and $m_0 \leq m_0'$. Since $top(E_0 \mid E_1) = top(E_0) \cup top(E_1)$ and $top(E_0' \mid E_1) = top(E_0') \cup top(E_1)$, then $\forall\langle r',m'\rangle \in top(E_0' \mid E_1)$. either $\exists\langle r_0,m_0\rangle \in top(E_0)$. $r_0 \sqsubseteq r'$ and $m_0 \leq m'$ (in the case $\langle r',m'\rangle \in top(E_0')$) or $\exists\langle r_1,m_1\rangle \in top(E_1)$. $r_1 = r'$ and $m_1 = m'$ (in the case $\langle r',m'\rangle \in top(E_1)$).
- *Rule Com*: $E = (E_0 \mid E_1) \xrightarrow{\tau} (E_0' \mid E_1')$, where $E_0 \xrightarrow{xy} E_0'$ and $E_1 \xrightarrow{\bar{x}y} E_1'$. By induction hypothesis, $\forall\langle r_0',m_0'\rangle \in top(E_0')$. $\exists\langle r_0,m_0\rangle \in top(E_0)$. $r_0 \sqsubseteq r_0'$ and $m_0 \leq m_0'$. Analogously, $\forall\langle r_1',m_1'\rangle \in top(E_1')$. $\exists\langle r_1,m_1\rangle \in top(E_1)$. $r_1 \sqsubseteq r_1'$ and $m_1 \leq m_1'$. Since $top(E_0 \mid E_1) = top(E_0) \cup top(E_1)$ and $top(E_0' \mid E_1') = top(E_0') \cup top(E_1')$, then $\forall\langle r',m'\rangle \in top(E_0' \mid E_1')$ either $\exists\langle r_0,m_0\rangle \in top(E_0)$. $r_0 \sqsubseteq r'$ and $m_0 \leq m'$ (in the case $\langle r',m'\rangle \in top(E_0')$) or $\exists\langle r_1,m_1\rangle \in top(E_1)$. $r_1 \sqsubseteq r'$ and $m_1 \leq m'$ (in the case $\langle r',m'\rangle \in top(E_1')$).
- *Rule Open/Res/Close*: These cases can be proved similarly.
- *Rule Rep*: $!_{\langle s,n\rangle}P \xrightarrow{\mu} L_{\langle s0,n+1\rangle}(P) \mid !_{\langle s1,n+1\rangle}P$. Then we have $top(!_{\langle s,n\rangle}P) = \{\langle s,n\rangle\}$ and $top(L_{\langle s0,n+1\rangle}(P) \| !_{\langle s1,n+1\rangle}P) = top(L_{\langle s0,n+1\rangle}(P)) \cup \{\langle s1,n+1\rangle\}$. By Lemma A.3 on $L_{\langle s0,n+1\rangle}(P)$, we have that $\forall\langle r',m'\rangle \in top(L_{\langle s0,n+1\rangle}(P)) \subseteq lab(L_{\langle s0,n+1\rangle}(P))$. $s0 \sqsubseteq r'$ and $n+1 \leq m'$. It follows that $\langle s,n\rangle$ is such that $s \sqsubseteq s1$ and $n < n+1$, as well as $s \sqsubseteq s0 \sqsubseteq r'$ and $n < n+1 \leq m'$ for any $\langle r',m'\rangle \in top(L_{\langle s0,n+1\rangle}(P))$.

(4) We prove that $wf(E')$ holds, by induction on the depth of $E \xrightarrow{\mu} E'$.
- *Rule Input/Output*: $E = L_{\langle s,n\rangle}(\mu.P) \xrightarrow{\mu} E' = L_{\langle s,n+1\rangle}(P')$ (either $P' = P$ or $P' = P\{z/y\}$). By Lemma A.4, $wf(L_{\langle s,n+1\rangle}(P'))$.
- *Rule Par*: $E = (E_0 \mid E_1) \xrightarrow{\mu} (E_0' \mid E_1)$, where $bn(\mu) \cap fn(E_1) = \emptyset$ and $E_0 \xrightarrow{\mu} E_0'$. Since $wf(E_0 \mid E_1)$, then $top(E_0) \Re top(E_1)$. By induction, $E_0 \xrightarrow{\mu} E_0'$ implies that $wf(E_0')$. By item (3) and Lemma A.2, $top(E_0) \Re top(E_1)$ implies $top(E_0') \Re top(E_1)$. Hence $wf(E_0' \mid E_1)$.
- *Rule Com*: $E = (E_0|E_1) \xrightarrow{\tau} (E_0'|E_1')$, where $E_0 \xrightarrow{xy} E_0'$ and $E_1 \xrightarrow{\bar{x}y} E_1'$. By induction hypothesis, $E_0 \xrightarrow{xy} E_0'$ implies that $wf(E_0')$; analogously, $E_1 \xrightarrow{\bar{x}y} E_1'$ implies that $wf(E_1')$. By item (3) and Lemma A.2, $top(E_0) \Re top(E_1)$ implies $top(E_0') \Re top(E_1')$. Hence $wf(E_0' \mid E_1')$.
- *Rule Open/Res/Close*: These cases can be proved similarly.

– *Rule Rep*: It suffices to recall that $top(L_{\langle s0,n+1\rangle}(P')|!_{\langle s1,n+1\rangle}P') = \{\langle s1, n+1\rangle\} \cup top(L_{\langle s0,n+1\rangle}(P'))$ and to apply Lemma A.3 on $L_{\langle s0,n+1\rangle}(P')$. □

Lemma A.6. *Let $E_0, E_1 \in \mathcal{P}^e$. $top(E_0) \Re top(E_1)$ implies $lab(E_0) \Re lab(E_1)$.*

Proof. By item (2) of Lemma A.5, $\forall i \in \{0,1\}$. $\forall\langle s_i, n_i\rangle \in lab(E_i)$. $\exists\langle r_i, m_i\rangle \in top(E_i)$. $r_i \sqsubseteq s_i$ and $m_i \leq n_i$. $top(E_0) \Re top(E_1)$ and Lemma A.2 imply $\forall\langle s_0, n_0\rangle \in lab(E_0)$. $\forall\langle s_1, n_1\rangle \in lab(E_1)$. $\{\langle s_0, n_0\rangle\}\Re\{\langle s_1, n_1\rangle\}$, i.e. $lab(E_0) \Re lab(E_1)$. □

Lemma A.7. *Let $E, E' \in \mathcal{P}^e$. $E \xrightarrow{\mu} E'$. Let $\langle s, n\rangle \in lab(E)$ and $\langle s, n\rangle \notin lab(E')$. Then $\langle s, n\rangle \in top(E)$.*

Proof. By induction on the depth of $E \xrightarrow{\mu} E'$.

– *Rule Input/Output*: $E = L_{\langle s',n'\rangle}(\mu.P) \xrightarrow{\mu} E' = L_{\langle s',n'+1\rangle}(P')$ (where either $P' = P$ or $P' = P\{z/y\}$). Then $top(E) = \{\langle s', n'\rangle\}$ and $lab(E) = lab(E') \cup \{\langle s', n'\rangle\}$. It follows that $s = s', n = n'$, and therefore $\langle s, n\rangle \in top(E)$.
– *Rule Par*: $E = (E_0|E_1) \xrightarrow{\mu} (E_0'|E_1)$ and $E_0 \xrightarrow{\mu} E_0'$. Since $lab(E_0|E_1) = lab(E_0) \cup lab(E_1)$ and $lab(E_0' \mid E_1) = lab(E_0') \cup lab(E_1)$, we have $\langle s, n\rangle \notin lab(E_0')$, $\langle s, n\rangle \notin lab(E_1)$, and therefore $\langle s, n\rangle \in lab(E_0)$. By induction, $\langle s, n\rangle \in top(E_0)$ and therefore $\langle s, n\rangle \in top(E_0) \cup top(E_1) = top(E_0 \mid E_1) = top(E)$.
– *Rule Com*: $E = (E_0 \mid E_1) \xrightarrow{\tau} (E_0' \mid E_1')$, where $E_0 \xrightarrow{xy} E_0'$ and $E_1 \xrightarrow{\bar{x}y} E_1'$. Since $lab(E_0 \mid E_1) = lab(E_0) \cup lab(E_1)$, we have that either $\langle s, n\rangle \in lab(E_0)$ or $\langle s, n\rangle \in lab(E_1)$. Let us consider the first case (the other one is analogous). Since $lab(E_0' \mid E_1') = lab(E_0') \cup lab(E_1')$, we have that $\langle s, n\rangle \notin lab(E_0')$. The rest is the same as in the case of *Par*.
– *Rules Open/Res*: Immediate, by induction.
– *Rule Close*: Similar to the case of *Com*.
– *Rule Rep*: Trivial, since $E =!_{\langle s',n'\rangle}P$ and $lab(!_{\langle s',n'\rangle}P) = \{\langle s', n'\rangle\} = top(!_{\langle s',n'\rangle}P)$. □

Lemma A.8. *Let $E, E' \in \mathcal{P}^e$. $E \xrightarrow{\mu} E'$ and $\langle r, m\rangle \in top(E)$. $\langle r, m\rangle \in lab(E')$ implies $\langle r, m\rangle \in top(E')$.*

Proof. Let $\langle r, m\rangle \in top(E) \cap lab(E')$ and suppose, by contradiction, that $\langle r, m\rangle \notin top(E')$. By item (2) of Lemma A.5, $\exists\langle r', m'\rangle \in top(E')$. $r' \sqsubseteq r$ and $m' \leq m$. By item (3) of Lemma A.5, $\exists\langle r'', m''\rangle \in top(E)$. $r'' \sqsubseteq r' \sqsubseteq r$ and $m'' \leq m' \leq m$. It follows that $\exists\langle r, m\rangle, \langle r'', m''\rangle \in top(E)$ such that $r'' \sqsubseteq r$ and $m'' \leq m$. Note also that the pairs $\langle r, m\rangle, \langle r', m'\rangle$ must be different and therefore also $\langle r, m\rangle, \langle r'', m''\rangle$ are different. Thus we get a contradiction with item (1) of Lemma A.5. □

Appendix B. Must- and fair-testing semantics

This appendix section contains intermediate results and proofs of the statements omitted in Section 5.

Proposition B.1. *Let $P \in \mathcal{P}$ and $o \in \mathcal{O}$. P must o implies P fair o.*

Proof. By contradiction, suppose P $\not{fair}$ o, i.e. there is a maximal computation from $P \mid o$

$$P \mid o = T_0 \xrightarrow{\tau} T_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} T_i \, [\xrightarrow{\tau} \ldots]$$

such that $T_i \not\Longrightarrow^{\omega}$ for some $i \geq 0$, i.e. for every T'. $T_i \Longrightarrow^{\varepsilon} T'$ it holds that $T' \not\xrightarrow{\omega}$. It follows that $T_i \not\xrightarrow{\omega}$, $\forall j \in [0..(i-1)].T_j \not\xrightarrow{\omega}$ and $\forall h \geq i.\, T_h \not\xrightarrow{\omega}$, by hypothesis on T_i. In fact, since

ω can not synchronize, it does not disappear once it is at the top level of a term. It follows that the above computation is such that $\forall j \geq 0.\, T_j \not\xrightarrow{\omega}$, i.e. P *must* o. □

Proposition B.2. $\exists P \in \mathcal{P}.\ \exists o \in \mathcal{O}.\ P$ *fair* o *and* P *must* o.

Proof. Consider $P = (\nu b)(\bar{b} \,|\, !b.\bar{b}) \,|\, \bar{a}$ and $o = a.\omega$. Since $(\nu b)(\bar{b} \,|\, !b.\bar{b}) \xrightarrow{\tau} (\nu b)(\bar{b} \,|\, !b.\bar{b}) \xrightarrow{\tau} \ldots$, there is an unsuccessful maximal computation from $P \,|\, o$, i.e. P *must* o. However, P *fair* o, since every maximal computation from $P \,|\, o$

$$P \,|\, o = T_0 \xrightarrow{\tau} T_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} T_i \xrightarrow{\tau} \ldots$$

is such that either $\forall i \geq 0.\, T_i = (\nu b)(\bar{b} \,|\, !b.\bar{b}) \,|\, \bar{a} \,|\, a.\omega$ or $\exists j \geq 1.\, T_j = (\nu b)(\bar{b} \,|\, !b.\bar{b}) \,|\, \omega \xrightarrow{\omega}$ and $\forall i \in [0..(j-1)].\, T_i = (\nu b)(\bar{b} \,|\, !b.\bar{b}) \,|\, \bar{a} \,|\, a.\omega$ and $T_i \stackrel{\varepsilon}{\Longrightarrow} T_j$. □

Appendix C. Weak-fair must, strong-fair must and fair-testing semantics

Lemma C.1. $\forall S \in \mathcal{E}^e$.

1. $Ll(S)$ *is a finite set;*
2. $S \not\xrightarrow{\tau}$ *implies* $Ll(S) = \emptyset$*;*
3. $v \in Ll(S)$ *implies* $\exists S' \in \mathcal{E}^e.\ S \xrightarrow{\mu} S'$ *and* $\forall S''.\ S' \stackrel{\varepsilon}{\Longrightarrow} S''.\ v \notin Ll(S'')$*;*
4. $\exists S' \in \mathcal{E}^e.\ S \stackrel{\varepsilon}{\Longrightarrow} S',\ Ll(S) \cap Ll(S') = \emptyset$ *and* $\forall S''.\ S' \stackrel{\varepsilon}{\Longrightarrow} S''.\ Ll(S) \cap Ll(S'') = \emptyset$.

Proof. We recall that $\forall S \in \mathcal{E}^e.\ Ll(S) \subseteq top(S) \subseteq lab(S)$. Items (1) and (2) are trivial. Consider item (3). S' is the term obtained from S by performing the action labeled by v: by Theorem 4.8 , $v \notin lab(S')$ and for every S'' such that $S' \stackrel{\varepsilon}{\Longrightarrow} S''$, $v \notin lab(S'')$ holds. Hence $v \notin Ll(S')$ and for every S'' such that $S' \stackrel{\varepsilon}{\Longrightarrow} S''$, $v \notin Ll(S'')$ holds.

To prove item (4) it suffices to apply the previous item, where $\mu = \tau$. The term S' is obtained from S by performing any $v \in Ll(S)$ and such that for every $v \in Ll(S)$ and every S'' such that $S' \stackrel{\varepsilon}{\Longrightarrow} S''$ either $v \notin lab(S')$ (following that $v \notin lab(S'')$) or $v \notin Ll(S'')$ and $v \in lab(S')$. In both cases, $Ll(S) \cap Ll(S') = \emptyset$ and $Ll(S) \cap Ll(S'') = \emptyset$. Since $Ll(S)$ is finite, such S' exists. □

Lemma C.2. *Let* $S \in \mathcal{E}^e$ *and* $S = S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S_i\, [\xrightarrow{\tau} \ldots]$ *be a strong-fair computation from* S*. If* $\exists S'_0, S'_1, S'_2, \ldots, S'_n \in \mathcal{E}^e$ *such that*

$$S' = S'_0 \xrightarrow{\tau} S'_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S'_n = S,$$

then

$$S' \xrightarrow{\tau} S'_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S'_n \xrightarrow{\tau} S_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S_i\, [\xrightarrow{\tau} \ldots]$$

is a strong-fair computation from S'*.*

Proof. Consider $\mathcal{C} = S' \xrightarrow{\tau} S'_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S'_n \xrightarrow{\tau} S'_{n+1} \xrightarrow{\tau} \ldots \xrightarrow{\tau} S'_{n+i}\, [\xrightarrow{\tau} \ldots]$, where $\forall j \geq 0.\ S'_{n+j} ::= S_j$. Obviously $\mathcal{C}$ is a maximal computation from S'. To prove that $\mathcal{C}$ is also *strong-fair*, it suffices to prove that $\forall v \in (\{0,1\}^* \times \mathbb{N}).\ \exists h \geq 0.\ \forall k \geq h.\ v \notin Ll(S'_k)$. Since $S'_n \xrightarrow{\tau} S'_{n+1} \xrightarrow{\tau} \ldots \xrightarrow{\tau} S'_{n+i}\, [\xrightarrow{\tau} \ldots]$ is a *strong-fair* computation from S'_n, then $\forall v \in (\{0,1\}^* \times \mathbb{N}).\ \exists h \geq n.\ \forall k \geq h.\ v \notin Ll(S'_k)$. Since $n \geq 0$, $\forall v \in (\{0,1\}^* \times \mathbb{N}).\ \exists h \geq 0.\ \forall k \geq h.\ v \notin Ll(S'_k)$. I.e., $\mathcal{C}$ is a *strong-fair* computation from S'. □

This work is licensed under the Creative Commons Attribution-NoDerivs License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nd/2.0/ or send a letter to Creative Commons, 171 Second St, Suite 300, San Francisco, CA 94105, USA, or Eisenacher Strasse 2, 10777 Berlin, Germany