Fairness of Components in System Computations

F. Corradini¹, M.R. Di Berardini²

Dipartimento di Matematica e Informatica
Università di Camerino, Camerino, Italy

W. Vogler³

Institut für Informatik
Universität Augsburg, Augsburg, Germany

Abstract

In this paper we provide a simple characterization of (weak) fairness of components as defined by Costa and Stirling in [6]. The study is carried out at system specification level by resorting to a common process description language. This paper follows and exploits similar techniques as those developed in [1] – where fairness of actions was taken into account and was contrasted to the PAFAS timed operational semantics – but the characterization of fair executions is based on a new semantics for PAFAS; it makes use of only two copies of each basic action instead of infinitely many as in [6] and allows for a simple and finite representation of fair executions by using regular expressions. The new semantics can also be understood as describing timed behaviour of systems with upper time bounds.

Keywords: Process Algebras, Timed Process Algebras, Worst-Case Efficiency, Fairness of Actions, Fairness of Components, Liveness properties

¹ Email: flavio.corradini@unicam.it
² Email: mariarita.diberardini@unicam.it
³ Email: vogler@informatik.uni-augsburg.de

1571-0661/$ – see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.entcs.2004.11.027
1 Introduction

In the theory and practice of parallel systems, fairness plays an important role when describing the system dynamics. It is a necessary requirement for proving liveness properties of the system. Several fairness notions applied to different entities in a system have been proposed in the literature.

Costa and Stirling define fairness for CCS without restriction in [5] and for fully fledged CCS in [6] and present very nice characterizations of fair runs. They distinguish between weak fairness and strong fairness as well as between fairness of actions (also called events) and of components; while these notions coincide in [5], they differ in [6], where weak and strong fairness of components are studied. In this paper, we will concentrate on weak fairness, which requires that if a component (an action, resp.) can almost always proceed then it must eventually do so, and in fact it must proceed infinitely often. characterizes fair computations as the concatenation of certain finite sequences, called LP-steps in [6]. This characterization permits to think of fairness in terms of a localizable property and not as a property of complete (maximal) executions; but even for a finite-state process, LP-steps usually give rise to a transition system with infinitely many transitions, which is therefore infinitely branching.

In our previous paper [1], we have related weak fairness of actions and a timed operational semantics by resorting to a common, well-known process description language PAFAS (a variant of CCS with TCSP parallel composition). The language is extended with labels allowing to filter out those process executions that are (weakly) fair (as in [5,6]), and with upper time bounds for the process activities (as in [2]), where these bounds are 1 for simplicity and time is discrete. (Upper time bounds have also been studied in [7] for the area of distributed algorithms, and in e.g. [4] for Petri nets.) The paper [1] shows that fairness and timing, two important features of parallel system computations, are closely related by giving two main results. First, it is shown that each everlasting (or non-Zeno) timed process execution is fair. Second, [1] provides a characterization for fair executions of untimed processes in terms of timed process executions. For finite state processes, it also results in a finite representation of fair executions using regular expressions.

In this paper we concentrate on weak fairness of components. It turned out that the PAFAS timed operational semantics is not a suitable abstraction for fairness of components as it is for fairness of actions. But we have found a variation of this semantics which allows us to characterize Costa and Stirling’s fairness of components in terms of a much simpler filtering of system executions compared to the label-based fairness definition in [5,6]. The results of this paper are conceptually the same as those in [1], but a number of technical changes were needed to define the new semantics and, consequently, the proof
details are quite different. The new operational semantics of processes we have arrived at can again be understood as the behaviour of timed processes with upper time bounds. We assume that for each parallel component this upper time bound is 1; hence, a component will perform some action within time 1 provided it is continually enabled (or live in the terminology of [5,6]). In other words, when time 1 passes, a live component becomes urgent and, before the next time step, it must perform an action (or get disabled). The phases between the time steps correspond to the above mentioned LP-steps in [6].

Our characterization of fair executions results in a representation with technical advantages compared to the approach of [5,6]. To keep track of the different instances of system activities along a system execution, Costa and Stirling associate labels to actions operators. The labels are essential in the definition of fair computations. New labels are created dynamically during the system evolution with the immediate effect of changing the syntax of terms and of assuming that different instances of the same basic actions exist; if a process has an infinite execution, there will be infinitely many instances of some actions – distinguished by their label. Consequently, because of this dynamic generation of labels, cycles in the transition system of a process are impossible and even finite-state processes (according to the ordinary operational semantics) usually become infinite-state. From the maximal runs of such a transition system, Costa and Stirling filter out the fair computations by a criterion that considers the processes and their labels on a maximal run.

Our alternative operational semantics also provides such a two-level description. We also change the syntax of processes to take note of urgency, but this is much simpler than the labels of [5,6]; e.g. we only assume two instances of the same basic action corresponding to two different states of the action itself: one in which the action is not forced to be performed, and one in which it has to be performed urgently. An important consequence of this fact is that our operational semantics leaves finite-state processes finite-state. To get the fair runs, we apply a simpler filter, which does not consider the processes: we simply require that infinitely many time steps occur in a run, i.e. we only consider non-Zeno runs. As a small price, we have to project away the time steps in the end.

As mentioned above, Costa and Stirling give a one-level characterization of fair computations with an SOS-semantics defining so-called LP-steps; these are (finite, though usually unbounded) sequences of actions leading from ordinary processes to ordinary processes, with the effect that even finite-state transition systems for LP-steps usually have infinitely many transitions – although they are at least finite-state. In contrast, our operational semantics only refers to
unit time steps and single actions, and consequently a finite-state transition system is really finite.

Finally, using standard automata-theoretic techniques, we can get rid of the time steps in such a finite-state transition system by constructing another finite-state transition system with regular expressions as arc labels; maximal runs in this transition system are exactly the fair runs. This way we also arrive at a one-level description, and ours is truly finite. With respect to the similar result in [1], we have overcome some technical problems with the treatment of recursive processes; as a consequence the transition system in the present paper provides a more faithful description of fair runs because it only contains standard processes (without any marking of urgent components) which can be reached from the initial process according to standard transitions.

2 PAFAS - A Process Algebra for Faster Asynchronous Systems

PAFAS is a CCS-like process description language [8] (with TCSP-like parallel composition), where basic actions are atomic and instantaneous but have associated a time bound interpreted as a maximal time delay for their execution. As explained in [2], these upper time bounds (which are either 0 or 1, for simplicity) are suitable for evaluating the performance of asynchronous systems. Moreover, time bounds do not influence functionality (which actions are performed); so compared to CCS, also PAFAS treats the full functionality of asynchronous systems. In the present paper, the time bounds are associated to the parallel components of a term, resulting in slightly different terms and different SOS-rules; this variant of PAFAS will be called PAFAS\textsuperscript{C} henceforth.

2.1 PAFAS\textsuperscript{C} Processes

We use standard notation. \( \mathbb{A} \) denotes an infinite set of basic actions. \( \tau \) represents internal activity. Let \( \mathbb{A}_\tau = \mathbb{A} \cup \{ \tau \} \). Elements of \( \mathbb{A} \) are denoted by \( a, b, c, \ldots \) and those of \( \mathbb{A}_\tau \) are denoted by \( \alpha, \beta, \ldots \). Actions in \( \mathbb{A}_\tau \) can let time 1 pass before their execution, i.e. 1 is their maximal delay. After that time, they become urgent actions written \( \underline{a} \) or \( \underline{\tau} \); these have maximal delay 0. The set of urgent actions is denoted by \( \mathbb{A}_\tau = \{ \underline{a} \mid a \in \mathbb{A} \} \cup \{ \underline{\tau} \} \) and is ranged over by \( \alpha, \beta, \ldots \). Elements of \( \mathbb{A}_\tau \cup \mathbb{A}_\tau \) are ranged over by \( \mu \). \( \mathcal{X} \) is the set of process variables, used for recursive definitions. Elements of \( \mathcal{X} \) are denoted by \( x, y, z, \ldots \). \( \Phi : \mathbb{A}_\tau \to \mathbb{A}_\tau \) is a general relabelling function if the set \( \{ \alpha \in \mathbb{A}_\tau \mid \emptyset \neq \Phi^{-1}(\alpha) \neq \{ \alpha \} \} \) is finite and \( \Phi(\tau) = \tau \). Such a function can also be used to define hiding: \( P/A \), where the actions in \( A \) are made internal, is
the same as $P[\Phi A]$, where the relabelling function $\Phi_A$ is defined by $\Phi_A(\alpha) = \tau$ if $\alpha \in A$ and $\Phi_A(\alpha) = \alpha$ if $\alpha \notin A$.

We assume that time elapses in a discrete way\(^4\). Thus, an action prefixed process $a.P$ can either do action $a$ and become process $P$ (as usual in CCS) or can let one unit time step pass and become $\underline{a}P$; $\underline{a}$ is called urgent $a$, and $a.P$ cannot let time pass, but can only do $a$ to become $P$. Since we associate time bounds to components, we also mark the other dynamic operator $+$ as urgent: a process $P+Q$ becomes $P+Q$ after a time step.

**Definition 2.1** (timed process terms)

The set $\hat{P}_1$ of initial timed process terms is generated by the grammar:

$$P ::= \text{nil} \mid x \mid \alpha.P \mid P + P \mid P\parallel AP \mid P[\Phi] \mid \text{rec } x.P$$

where $x \in \mathcal{X}$, $\alpha \in A_\tau$, $\Phi$ is a general relabelling function and $A \subseteq A$ possibly infinite. Elements in $\hat{P}_1$ correspond to ordinary CCS-like processes.

The set $\hat{P}$ of (general) timed process terms is generated by the grammar:

$$Q ::= P \mid \alpha.P \mid P + P \mid Q\parallel AQ \mid Q[\Phi] \mid \text{rec } x.Q$$

where $P \in \hat{P}_1$, $x \in \mathcal{X}$, $\alpha \in A_\tau$, $\Phi$ is a general relabelling function, and $A \subseteq A$ possibly infinite. We assume that recursion is always (time-)guarded, i.e. for $\text{rec } x.Q$ variable $x$ only appears in $Q$ within the scope of a prefix $\alpha.(.)$ with $\alpha \in A_\tau$. A term $Q$ is guarded if each occurrence of a variable is guarded in this sense; it is closed if every variable $x$ is bound by the corresponding $\text{rec } x.$-operator. The set of closed timed process terms in $\hat{P}$ and $\hat{P}_1$, simply called processes and initial processes resp., is denoted by $P$ and $P_1$ resp.\(^5\)

For studying fairness, we are interested in the initial processes, and these coincide in PAFAS and in PAFAS\(^C\); they are actually common CCS/TCSP-like processes. The additional terms of $\hat{P}$ turn up in evolutions of terms from $\hat{P}_1$ involving time steps, and here PAFAS and PAFAS\(^C\) differ.

We define function $A(\underline{\hspace{1cm}})$ on process terms, that returns the active (or enabled) actions of a process term. Given a process $Q$, $A(Q)$ abbreviates $A(Q, \emptyset)$ and $A(Q, A)$ denotes the set of actions that process $Q$ can perform when the environment prevents the actions in $A \subseteq A$.\(^4\) PAFAS is not time domain dependent, meaning that the choice of discrete or continuous time makes no difference for the testing-based semantics of asynchronous systems studied in [2,3].

\(^4\) As shown in [2], $P_1$ processes do not have time-stops; i.e. every finite process run can be extended such that time grows unboundedly. This result was proven for a different operational semantics than that defined in this paper but a similar proof applies also in the current setting.
Definition 2.2 (activated basic actions)

Let $Q \in \tilde{P}$ and $A \subseteq \mathbb{A}$. The set $\mathcal{A}(Q, A)$ is defined by induction on $Q$ as follows:

- **Var, Nil:** $\mathcal{A}(x, A) = \mathcal{A}(\text{nil}, A) = \emptyset$
- **Pref:** $\mathcal{A}(\alpha.P, A) = \mathcal{A}(\alpha.P, A) = \{\alpha\}$ if $\alpha \notin A$ and $\mathcal{A}(\alpha.P, A) = \emptyset$ otherwise
- **Sum:** $\mathcal{A}(P_1 + P_2, A) = \mathcal{A}(P_1 + P_2, A) = \mathcal{A}(P_1, A) \cup \mathcal{A}(P_2, A)$
- **Par:** $\mathcal{A}(Q_1 \parallel B Q_2, A) = \mathcal{A}(Q_1, A \cup A') \cup \mathcal{A}(Q_2, A \cup A'')$
  where $A' = (\mathcal{A}(Q_1) \setminus \mathcal{A}(Q_2)) \cap B$, $A'' = (\mathcal{A}(Q_2) \setminus \mathcal{A}(Q_1)) \cap B$
- **Rel:** $\mathcal{A}(Q[\Phi], A) = \emptyset$ if $\mathcal{A}(Q[\Phi], A) = \emptyset$
- **Rec:** $\mathcal{A}(\text{rec } x.Q, A) = \mathcal{A}(Q, A)$

The set $A$ represents the actions restricted upon; therefore $\mathcal{A}(\alpha.P, A) = \mathcal{A}(\alpha.P, A) = \emptyset$ if $\alpha \in A$ and $\mathcal{A}(\alpha.P, A) = \mathcal{A}(\alpha.P, A) = \{\alpha\}$, if $\alpha \notin A$. A nondeterministic process can perform all the actions that its alternative components can perform minus the restricted ones. Parallel composition increases the prevented set. $\mathcal{A}(P \parallel B Q, A)$ includes the actions that $P$ can perform when we prevent all actions in $A$ plus all actions in $B$ that $Q$ cannot perform, and it includes the analogous actions of $Q$.

2.2 The operational behaviour of PAFAS$C$ processes

The transitional semantics describing the functional behavior of PAFAS$C$ processes indicates which basic actions they can perform. Timing can be disregarded: when an action is performed, one cannot see whether it was urgent or not, i.e. $\alpha.P \xrightarrow{\alpha} P$; on the other hand, component $\alpha.P$ has to act within time 1, i.e. it can also act immediately, giving $\alpha.P \xrightarrow{\alpha} P$. The operational semantics exploits two functions on process terms: $\text{clean}(_\_)$ and $\text{unmark}(_\_)$. Function $\text{clean}(_\_)$ removes all inactive urgencies in a process term $Q \in \tilde{P}$. Indeed, when a process evolves, components may lose their urgency since their actions are no longer enabled due to changes of the context; the corresponding change of markings is performed by $\text{clean}$, where again set $A$ in $\text{clean}(Q, A)$ denotes the set of actions that are not enabled due to restrictions of the environment.

Definition 2.3 (cleaning inactive urgencies)

Given a process term $Q \in \tilde{P}$ we define $\text{clean}(Q)$ as $\text{clean}(Q, \emptyset)$ which, for a set $A \subseteq \mathbb{A}$, $\text{clean}(Q, A)$ is defined as follows:

- **Nil, Var:** $\text{clean}(\text{nil}, A) = \text{nil}$, $\text{clean}(x, A) = x$
- **Pref:** $\text{clean}(\alpha.P, A) = \begin{cases} \alpha.P & \text{if } \alpha \in A \\ \alpha.P & \text{otherwise} \end{cases}$

clean(\alpha.P, A) = \alpha.P

Sum: \[\begin{aligned}
clean(P_1 + P_2, A) &= \begin{cases} P_1 + P_2 & \text{if } \mathcal{A}(P_1) \cup \mathcal{A}(P_2) \subseteq A \\ P_1 + P_2 & \text{otherwise} \end{cases} \\
clean(P_1 + P_2, A) &= P_1 + P_2
\end{aligned}\]

Par: \[\begin{aligned}
clean(Q_1 \parallel_B Q_2, A) &= \clean(Q_1, A \cup A') \parallel_B \clean(Q_2, A \cup A'') \\
\text{where } A' &= (\mathcal{A}(Q_1) \setminus \mathcal{A}(Q_2)) \cap B, A'' = (\mathcal{A}(Q_2) \setminus \mathcal{A}(Q_1)) \cap B
\end{aligned}\]

Rel: \[\begin{aligned}
clean(Q[\Phi], A) &= \clean(Q, \Phi^{-1}(A))[\Phi] \\
\end{aligned}\]

Rec: \[\begin{aligned}
clean(\text{rec } x.Q, A) &= \text{rec } x.\clean(Q, A)
\end{aligned}\]

Function unmark(\_\_) simply removes all urgencies (inactive or not) in a process term \(Q \in \hat{P}\) and can be defined, as expected, by induction on the process structure.

**Definition 2.4 (Functional operational semantics)** The following SOS-rules define the transition relations \(\xrightarrow{\alpha} \subseteq (\hat{P} \times \hat{P})\) for \(\alpha \in \mathcal{A}_\tau\), the action transitions. We write \(Q \xrightarrow{\alpha} Q'\) if \((Q, Q') \in \xrightarrow{\alpha}\) and \(Q \xrightarrow{\alpha}\) if there exists a \(Q' \in \hat{P}\) such that \((Q, Q') \in \xrightarrow{\alpha}\), and similar conventions will apply later on.

\[
\begin{array}{c}
PREF_{a1} \quad \alpha.P \xrightarrow{\alpha} P \\
SUM_{a1} \quad P_1 \xrightarrow{\alpha} P_1' \\
\quad \frac{P_1 + P_2 \xrightarrow{\alpha} P_1'}{P_1 + P_2} \quad \alpha \notin A, Q_1 \xrightarrow{\alpha} Q_1' \\
\quad \frac{Q_1 \parallel_A Q_2 \xrightarrow{\alpha} \clean(Q_1' \parallel_A Q_2)}{Q_1 \parallel_A Q_2} \quad \alpha \notin A, Q_1 \xrightarrow{\alpha} Q_1', Q_2 \xrightarrow{\alpha} Q_2' \\
\quad \frac{Q \xrightarrow{\alpha} Q'}{Q[\Phi] \xrightarrow{\Phi(\alpha)} Q'[\Phi]} \\
\quad \frac{Q \xrightarrow{\alpha} Q'}{\text{rec } x.Q \xrightarrow{\alpha} Q'}
\end{array}
\]

Additionally, there are symmetric rules for \(\text{Par}_{a1}, \text{Sum}_{a1}\) and \(\text{Sum}_{a2}\) for actions of \(P_2\).

Observe the following: due to our syntax, \(P_1\) in \(P_1 + P_2\) is an initial process, i.e. has no components marked as urgent, and the same applies to \(P_1'\). Thus, \(P_1 + P_2\) loses its urgency in a transition according to \(\text{Sum}_{a2}\); this corresponds to our intuition, since this atomic component (i.e. without parallel subcomponents) performs an action, which it had to perform urgently, and can afterwards wait with any further activity for time 1. When in the rules for parallel composition one component changes, this changes the context for
the other component such that some urgent components might get disabled. For the necessary changes to the marking, clean is called upon as announced above. The use of unmark in rule \textsc{Rec} has to be contrasted with the temporal behaviour defined next that marks as urgent recursive terms according to a rule \texttt{urgent} \texttt{(rec} x.\texttt{P}) = \texttt{rec} x.\texttt{urgent} \texttt{(P)}. Since occurrences of x in P are guarded, each x stands for a process which is not enabled yet and cannot be urgent; thus, recursive calls in \texttt{rec} x.\texttt{urgent} \texttt{(P)} refer to \texttt{P} and not to \texttt{urgent} \texttt{(P)}.

In addition to the purely functional transitions, we also consider transitions corresponding to the passage of one unit of time. The function \texttt{urgent} we exploit marks the enabled parallel components of a process as urgent; such a component can be identified with a dynamic operator (an action or a choice), which gets underlined. This marking occurs when a time step is performed, because afterwards the marked components have to act in zero time – unless they are disabled. If such an urgent component acts, it should lose its urgency; and indeed, the marking vanishes with the dynamic operator. The next time step will only be possible, if no component is marked as urgent.

\textbf{Definition 2.5} (time step, execution sequence, timed execution sequence)

For \(P \in \hat{\cal P}_1\), we write \(P \stackrel{\lambda}{\rightarrow} Q\) when \(Q = \texttt{urgent} \texttt{(P)}\), where \texttt{urgent} \texttt{(P)} abbreviates \texttt{urgent} \texttt{(P, }\emptyset\texttt{)} and \texttt{urgent} \texttt{(P, A)} is defined as follows:

\begin{align*}
\text{Nil, Var:} \quad \texttt{urgent} \texttt{(nil, A)} &= \texttt{nil}, \\
\text{Pref:} \quad \texttt{urgent} \texttt{(}\alpha.\texttt{P}, A\texttt{)} &= \begin{cases} \
\alpha.\texttt{P} & \text{if } \alpha \notin A \\
\alpha.\texttt{P} & \text{otherwise}
\end{cases} \\
\text{Sum:} \quad \texttt{urgent} \texttt{(P}_1 + P_2, A\texttt{)} &= \begin{cases} \
P_1 + P_2 & \text{if } (\mathcal{A}(P_1) \cup \mathcal{A}(P_2)) \setminus A \neq \emptyset \\
P_1 + P_2 & \text{otherwise}
\end{cases} \\
\text{Par:} \quad \texttt{urgent} \texttt{(P}_1 \parallel B P_2, A\texttt{)} &= \texttt{urgent} \texttt{(P}_1, A \cup A'\texttt{)} \parallel B \texttt{urgent} \texttt{(P}_2, A \cup A''\texttt{)} \texttt{where } A' = (\mathcal{A}(P_1) \setminus \mathcal{A}(P_2)) \cap B, A'' = (\mathcal{A}(P_2) \setminus \mathcal{A}(P_1)) \cap B \\
\text{Rel:} \quad \texttt{urgent}(P[\Phi, A]) &= \texttt{urgent}(P, \Phi^{-1}(A))[\Phi] \\
\text{Rec:} \quad \texttt{urgent}(\texttt{rec} x.\texttt{P}, A) &= \texttt{rec} x.\texttt{urgent} \texttt{(P, A)}
\end{align*}

We say that a sequence of transitions \(\gamma = Q_0 \xrightarrow{\lambda_0} Q_1 \xrightarrow{\lambda_1} \ldots\) with \(\lambda_i \in \mathbb{A}_T \cup \{1\}\) is a \textit{timed execution sequence} if it is an infinite sequence of action transitions and time steps; note that a maximal sequence of such transitions/steps is never finite, since for \(\gamma = Q_0 \xrightarrow{\lambda_0} Q_1 \xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_{n-1}} Q_n\), we have \(Q_n \xrightarrow{\alpha}\) or \(Q_n \xrightarrow{\frac{1}{\lambda}}\).

For an \textit{initial} process \(P_0\), we say that a sequence of transitions \(\gamma = P_0 \xrightarrow{\alpha_0} P_1 \xrightarrow{\alpha_1} \ldots\) with \(\alpha_i \in \mathbb{A}_T\) is an \textit{execution sequence} if it is a maximal sequence of action transitions; i.e. it is infinite or ends with a process \(P_n\) such that \(P_n \xrightarrow{\alpha}\) for any action \(\alpha\).
As an example for the use of the various definitions, consider the following behaviour of \( P = (a.\text{nil} + b.\text{nil}) \parallel_{\{a,b\}} \text{rec } x. (a.\text{nil} + c.(b.\text{nil} + d.x)) \):

\[
P \xrightarrow{\Downarrow} (a.\text{nil} + b.\text{nil}) \parallel_{\{a,b\}} \text{rec } x. (a.\text{nil} + c.(b.\text{nil} + d.x))
\]

\[
(a.\text{nil} + b.\text{nil}) \parallel_{\{a,b\}} (b.\text{nil} + d.\text{rec } x. (a.\text{nil} + c.(b.\text{nil} + d.x))) \xrightarrow{d}
\]

\[
(a.\text{nil} + b.\text{nil}) \parallel_{\{a,b\}} (b.\text{nil} + d.\text{rec } x. (a.\text{nil} + c.(b.\text{nil} + d.x))) \xrightarrow{c}
\]

After the time step, both components are urgent; the left hand component can synchronize on \( a \), while \( b \) is not possible. Then the right hand component performs \( c \) and loses its urgency. Now \( a \) is not possible anymore, but the left hand component remains urgent since now it can synchronize on \( b \). Also, observe the application of \( \text{Rec}_a \). The process reached returns to itself with \( dc \), so this behaviour could be repeated indefinitely. But since the left hand component is urgent all the time, a time step is never possible, matching the intuitive idea that this component has to act within time 1.

3 Fairness and PAFAS\(^C\)

In this section we briefly describe our theory of fairness. It closely follows Costa and Stirling’s theory of (weak) fairness. The main ingredients are:

- \textit{A labelling for process terms}. This allows to detect during a transition which component actually moves; e.g., for process \( P = \text{rec } x. \alpha.x \), we need additional information to detect whether the left hand side or the right hand side actually moves in the transition \( P \parallel_{\emptyset} P \xrightarrow{\alpha} P \parallel_{\emptyset} P \).

- \textit{Live components}. A component of a process term is live if it can perform an action. In a term like \( a.b.\text{nil}\parallel\{b\} b.\text{nil} \) only action \( a \) can be performed while \( b \) cannot, momentarily. Thus the left component of the parallel composition is live and such a component corresponds to a label. Intuitively, the components becoming urgent with a time step should exactly be the live components.

- \textit{Fair sequences}. A maximal sequence is fair when no component in a process term becomes live and then remains live throughout.

These items sketch the general methodology used by Costa and Stirling to define and isolate fair computations in \([5,6]\). Most of the definitions in the rest of this section are taken from \([6]\) with the obvious slight variations due to the different language we are using (the timed process algebra PAFAS\(^C\) with TCSP parallel composition instead of CCS). We also take from \([6]\) those results that are language independent.
3.1 A labelling for process terms

Costa and Stirling associate labels with all basic actions and operators inside a process, in such a way that no label occurs more than once in an expression. We call this property unicity of labels. Also along a computation, labels are unique and, once a label disappears, it will not reappear anymore.

The set of labels is $\text{LAB} = \{1, 2\}^*$ with $\varepsilon$ as the empty label and $u, v, w, \ldots$ as typical elements. Labels are written as indexes and in case of parallel composition as upper indexes; they are assigned systematically following the structure of PAFA$\text{S}^C$ terms. Due to recursion the labelling is dynamic: the rule for rec generates new labels.

**Definition 3.1 (labelled process algebra)**

The labelled process algebra $L(\tilde{P})$ (and similarly $L(\tilde{P}_1)$ etc.) is defined as $\bigcup_{u \in \text{LAB}} L_u(\tilde{P})$, where $L_u(\tilde{P}) = \bigcup_{P \in \tilde{P}} L_u(P)$ and $L_u(P)$ is defined inductively as follows:

- **Nil, Var:** $L_u(\text{nil}) = \{\text{nil}_u\}$, $L_u(x) = \{x_u\}$

  In examples, we will often write $\text{nil}$ for $\text{nil}_u$, if the label $u$ is not relevant.

- **Pref:** $L_u(\mu.P) = \{\mu_u.P' \mid P' \in L_u \mu_1(P)\}$

- **Sum:** $L_u(P_1 + P_2) = \{P_1 +_u P_2' \mid P_1' \in L_u(P_1), P_2' \in L_u(P_2)\}$

  $L_u(\nu_1 \nu_2 P_1) = \{P_1 +_u \nu_2 P_1' \mid P_1' \in L_u(P_1), P_2' \in L_u(P_2)\}$

- **Par:** $L_u(Q_1 \parallel_A Q_2) = \{Q_1 +_u Q_2' \mid Q_1 \in L_u\varepsilon (Q_1), Q_2' \in L_u\varepsilon (Q_2)\}$

  where $v, v' \in \text{LAB}$

- **Rel:** $L_u(Q[\Phi]) = \{Q'[\Phi_u] \mid Q' \in L_u\varepsilon (Q)\}$ where $v \in \text{LAB}$

- **Rec:** $L_u(\text{rec } x.Q) = \{\text{rec } x_u.Q' \mid Q' \in L_u(Q)\}$

  We assume that, in $\text{rec } x_u.Q$, $\text{rec } x_u$ binds all free occurrences of a labelled $x$; analogously, $\Phi_u$ acts on actions as $\Phi$. We let $L(Q) = \bigcup_{u \in \text{LAB}} L_u(Q)$ and $\text{LAB}(Q)$ is the set of labels occurring in $Q$.

The unicity of labels must be preserved under derivation. For this reason, in the rec rule the standard substitution must be replaced by a substitution operation which also changes the labels of the substituted expression.

**Definition 3.2 (a new substitution operator)**

The new substitution operation, denoted by $\{\_\}_u$, is defined on $L(\tilde{P})$ using the following operators:

- i. $(\_)+^v$ If $Q \in L_u(\tilde{P})$, then $(Q)+^v$ is the term in $L_vu(\tilde{P})$ obtained by prefixing $v$ to labels in $Q$.

- ii. $(\_)_v$ If $Q \in L_u(\tilde{P})$, then $(Q)_v$ is the term in $L_v(\tilde{P})$ obtained by removing the prefix $u$ from all labels in $Q$. (Note that $u$ is the unique prefix-
Suppose $Q, R \in L(\tilde{P})$ and $x_u, \ldots, x_v$ are all free occurrences of a labelled $x$ in $Q$ then $Q[\{R/x\}] = Q\{(R)_e^+u/x_u, \ldots, (R)_e^+v/x_v\}$. The motivation of this definition is that in $Q[\{R/x\}]$ each substituted $R$ inherits the label of the $x$ it replaces.

Moreover, for $P \in L(\tilde{P}_1)$ and $A \subseteq A_\tau$ we can define $\text{urgent}(P, A)$ just as in Definition 2.5. Similarly, we can define $\mathcal{A}(Q, A)$, $\text{clean}(Q, A)$ and $\text{unmark}(Q)$ for labelled terms as above. Now, the behavioural operational semantics of the labelled PAFAS$^C$ is obtained by replacing the rules $\text{Rec}_a$ in Definition 2.4 with the rule:

\[
\text{Rec}_a
\begin{array}{c}
Q[\{\text{rec } x_u.\text{unmark}(Q)/x\}] \\
\text{rec } x_u.Q
\end{array}
\xrightarrow{\alpha}
Q'
\]

and the rules $\text{Pref}_{a1}$ and $\text{Pref}_{a2}$ in Definition 2.4 with the rules:

\[
\begin{array}{c}
\text{Pref}_{a1}
\hline
\text{Pref}_{a2}
\end{array}
\begin{array}{c}
\alpha_u.P \\
\alpha_u.P
\end{array}
\xrightarrow{\alpha}
\begin{array}{c}
P \\
P
\end{array}
\]

because we assume that labels are not observable when actions are performed. The other rules are unchanged.

Easy but important are the relationships between activated actions and transitions of PAFAS$^C$ and labelled PAFAS$^C$ processes. The following proposition shows that labels are just annotations that do not interfere with these notions. Let $R$ be the operation of removing labels from a labelled term.

**Proposition 3.3** Let $Q \in L_u(\tilde{P})$ and $A \subseteq A_\tau$. Then:

i. $Q \xrightarrow{\alpha} R$ implies $R(Q) \xrightarrow{\alpha} R(R)$ in unlabelled PAFAS$^C$;

ii. if $Q' \xrightarrow{\alpha} R'$ in unlabelled PAFAS$^C$ and $Q' = R(Q)$, then $Q \xrightarrow{\alpha} R$ for some $R$ with $R' = R(R)$;

iii. $\mathcal{A}(Q, A) = \mathcal{A}(R(Q), A)$.

An immediate consequence of the labelling are the following facts that have been proven in [6]: No label occurs more than once in a given process $P \in L_u(\tilde{P})$. Moreover, central to labelling is the persistence and disappearance of labels under derivation. In particular, once a label disappears it can never reappear. It is these features which allow us to recognize when a component contributes to the performance of an action. Throughout the rest of this section we assume the labelled calculus.
3.2 Live components

To capture the fairness constraint for execution sequences, we need to define the live components. We now define $\text{LC}(Q, A)$ as the set of live components of $Q$ (when the execution of actions in $A$ are prevented by the environment).

**Definition 3.4 (live components)**

Let $Q \in L(\bar{P})$ and $A \subseteq A$. The set $\text{LC}(Q, A)$ is defined by induction on $Q$.

- **Var, Nil:** $\text{LC}(x_u, A) = \text{LC}(\text{nil}_u, A) = \emptyset$
- **Pref:** $\text{LC}(\mu_u.P, A) = \begin{cases} \{u\} & \text{if } \mu = \alpha \text{ or } \mu = \overline{\alpha} \text{ and } \alpha \notin A \\ \emptyset & \text{otherwise} \end{cases}$
- **Sum:** $\text{LC}(P_1 \oplus_u P_2, A) = \begin{cases} \{u\} & \text{if } \text{LC}(P_1, A) \cup \text{LC}(P_2, A) \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$

where $\oplus \in \{+, \cdot\}$
- **Par:** $\text{LC}(Q_1 \parallel_u Q_1', A) = \text{LC}(Q_1, A \cup A') \cup \text{LC}(Q_2, A \cup A'')$

where $A' = (A(Q_1) \setminus A(Q_2)) \cap B$, $A'' = (A(Q_2) \setminus A(Q_1)) \cap B$
- **Rel:** $\text{LC}(Q[\Phi_u], A) = \text{LC}(Q, \Phi^{-1}(A))$
- **Rec:** $\text{LC}(\text{rec } x_u.Q, A) = \text{LC}(Q, A)$

The set of live components in $Q$ is defined as $\text{LC}(Q, \emptyset)$ which we abbreviate to $\text{LC}(Q)$.

An important subset of the live components of a process $Q$ is the subset of urgent live components. Let $Q \in L(\bar{P})$ and $A \subseteq A$. The set $\text{UC}(Q, A)$ is defined as in Definition 3.4 when $\text{LC}(\_)$ is replaced by $\text{UC}(\_)$ and rules Pref and Sum are replaced by the following one (Again, define $\text{UC}(Q) = \text{UC}(Q, \emptyset)$):

**Definition 3.5 (urgent live components)**

Let $Q \in L(\bar{P})$ and $A \subseteq A$. The set $\text{UC}(Q, A)$ is defined by induction on $Q$.

- **Pref:** $\text{UC}(\mu_u.P, A) = \begin{cases} \{u\} & \text{if } \mu = \alpha \text{ and } \alpha \notin A \\ \emptyset & \text{otherwise} \end{cases}$
- **Sum:** $\text{UC}(P_1 +_u P_2, A) = \emptyset$

$\text{UC}(P_1 \pm_u P_2, A) = \begin{cases} \{u\} & \text{if } \text{LC}(P_1, A) \cup \text{LC}(P_2, A) \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$

Of course, $\text{UC}(Q, A) \subseteq \text{LC}(Q, A)$, for every $Q$ and $A \subseteq A$.

3.3 Fair execution sequences

**Definition 3.6 (fair execution sequences)**
Let $\gamma = P_0 \xrightarrow{\lambda_0} P_1 \xrightarrow{\lambda_1} \ldots$ be an execution sequence or a timed execution sequence; we will write ‘(timed) execution sequence’ for such a sequence. We say that $\gamma$ is fair if

$$\neg(\exists s \exists i. \forall k \geq i : s \in \text{LC}(P_k))$$

Following [6], we now present an alternative, more local, definition of fair computations which will be useful to prove our main statements.

**Definition 3.7 (B-step)**

For any process $P_0$, we say that $P_0 \xrightarrow{\lambda_0} P_1 \xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_{n-1}} P_n$ with $n > 0$ is a timed $B$-step when

i. $B$ is a finite set of event labels,

ii. $B \cap \text{LC}(P_0) \cap \ldots \cap \text{LC}(P_n) = \emptyset$.

If $\lambda_i \in \mathcal{A}_\tau, i = 0, \ldots, n - 1$, then the sequence is a $B$-step. If $P_0 \xrightarrow{\lambda_0} P_1 \xrightarrow{\lambda_1} \ldots \xrightarrow{\lambda_{n-1}} P_n$ is a (timed) $B$-step and $v = \lambda_0 \ldots \lambda_{n-1}$ we write $P_0 \xrightarrow{v_B} P_{n+1}$; if $B = \text{LC}(P_0)$, we also speak of a (timed) LC-step.

In particular, a (timed) LC-step from $P$ is “locally” fair: all live events of $P$ lose their liveness at some point in the step.

**Definition 3.8 (fair-step sequences)**

A (timed) fair-step sequence from $P_0$ is any maximal sequence of (timed) steps of the form

$$P_0 \xrightarrow{v_0} \text{LC}(P_0) P_1 \xrightarrow{v_1} \text{LC}(P_1) \ldots$$

A fair-step sequence is simply a concatenation of locally fair steps. If $\delta$ is a (timed) fair-step sequence, then its associated (timed) execution sequence is the sequence which drops all references to the sets $\text{LC}(P_i)$.

The following theorem shows that fair execution sequences and fair-step sequences are essentially the same and has been proven, as in [5,6], with yet another, intermediate notion of local fairness.

**Theorem 3.9** A (timed) execution sequence is fair if and only if it is the sequence associated with a (timed) fair-step sequence.

### 4 Fairness and Timing

This section is the core of the paper. It relates fairness and timing in a process algebraic setting, and it contains two main contributions:
(i) We provide a characterization of fair execution sequences of initial PAFAS processes (PAFAS processes evolving only via functional operational semantics) in terms of timed execution sequences.

(ii) For the case of a finite state process, we derive from this a finite representation of the fair runs with a transition system that has arcs labelled by regular expressions.

The following propositions are key statements for proving our main results. They also provide some intuition on the reasons why fairness and (our notion of) timing are so strictly related.

**Proposition 4.1** Let \( P_0 \in L(P_1) \), \( Q_0 = \text{urgent}(P_0) \) and \( v = \alpha_1 \ldots \alpha_n \in \mathbb{A}_\tau^* \). Then:

1. \( \overset{v}{P_0} \overset{\mathcal{L}_C(P_0)}{\longrightarrow} P_n \) implies \( \overset{v}{Q_0} \overset{\longrightarrow}{\longrightarrow} P_n \);
2. \( \overset{v}{Q_0} \overset{\longrightarrow}{\longrightarrow} Q_n \) and \( \mathcal{U}_C(Q_n) = \emptyset \) implies \( \overset{v}{P_0} \overset{\mathcal{L}_C(P_0)}{\longrightarrow} Q_n \).

**Proposition 4.2** Let \( P_0, P_1, P_2 \in L(P_1) \), \( v \) and \( w \in (\mathbb{A}_\tau)^* \). Then:

1. \( \overset{v}{P_0} \overset{\longrightarrow}{\longrightarrow} Q \overset{\longrightarrow}{\longrightarrow} P_1 \) implies \( \overset{v}{P_0} \overset{\mathcal{L}_C(P_0)}{\longrightarrow} P_1 \);
2. \( \overset{v}{P_0} \overset{\longrightarrow}{\longrightarrow} P_1 \overset{\longrightarrow}{\longrightarrow} Q \overset{\longrightarrow}{\longrightarrow} P_2 \) implies \( \overset{vw}{P_0} \overset{\mathcal{L}_C(P_0)}{\longrightarrow} P_2 \).

Then we show that each everlasting timed execution sequence is fair.

**Theorem 4.3** Each everlasting timed execution sequence, i.e. each timed execution sequence of the form

\[ \gamma = P_0 \overset{v_0}{\longrightarrow} P_1 \overset{1}{\longrightarrow} Q_1 \overset{v_1}{\longrightarrow} P_2 \overset{1}{\longrightarrow} Q_2 \overset{v_2}{\longrightarrow} P_3 \overset{1}{\longrightarrow} \ldots \]

with infinitely many time steps and \( v_0, v_1, v_2 \ldots \in (\mathbb{A}_\tau)^* \) is fair.

**Proof.** By Proposition 4.2 we have that \( P_0 \overset{v_0v_1}{\longrightarrow} \mathcal{L}_C(P_0) P_2 \), \( P_2 \overset{v_2}{\longrightarrow} \mathcal{L}_C(P_2) P_3 \) and so on. Then \( \gamma \) is a sequence associated with a timed fair-step sequence and is fair by Theorem 3.9.

\[ \square \]

### 4.1 Relating Timed Execution Sequences and Fair Execution

Our characterization results will be presented in two separate theorems where we distinguish between finite and infinite sequence of untimed systems. These results immediately carry over to fair execution sequences by Theorem 3.9. Furthermore, the timed execution sequences ignore labels, so they give indeed the announced characterizations with the simple filtering mechanism of requiring infinitely many time steps.

**Theorem 4.4** Let \( P_0 \in L(P_1) \) and \( v_0, v_1, v_2 \ldots \in (\mathbb{A}_\tau)^* \). Then:
1. For any finite fair-step sequence from $P_0$

$$P_0 \xrightarrow{v_0} \text{LC}(P_0) P_1 \xrightarrow{v_1} \text{LC}(P_1) P_2 \ldots P_{n-1} \xrightarrow{v_{n-1}} \text{LC}(P_{n-1}) P_n$$

there exists a timed execution sequence

$$P_0 \xrightarrow{1} Q_0 \xrightarrow{v_0} P_1 \xrightarrow{1} Q_1 \xrightarrow{v_1} P_2 \ldots P_{n-1} \xrightarrow{1} Q_{n-1} \xrightarrow{v_{n-1}} P_n \xrightarrow{1} Q_n \xrightarrow{1} Q_n \ldots$$

2. For any timed execution sequence from $P_0$

$$P_0 \xrightarrow{1} Q_0 \xrightarrow{v_0} P_1 \xrightarrow{1} Q_1 \xrightarrow{v_1} P_2 \ldots P_{n-1} \xrightarrow{1} Q_{n-1} \xrightarrow{v_{n-1}} P_n \xrightarrow{1} Q_n \xrightarrow{1} Q_n \ldots$$

the following is a finite fair-step sequence:

$$P_0 \xrightarrow{v_0} \text{LC}(P_0) P_1 \xrightarrow{v_1} \text{LC}(P_1) P_2 \ldots P_{n-1} \xrightarrow{v_{n-1}} \text{LC}(P_{n-1}) P_n$$

Similarly we can prove our characterization result for infinite sequence of untimed systems.

**Theorem 4.5** Let $P_0 \in L(P_1)$ and $v_0, v_1, v_2, \ldots \in (A \tau)^*$. Then:

1. For any infinite fair-step sequence from $P_0$

$$P_0 \xrightarrow{v_0} \text{LC}(P_0) P_1 \xrightarrow{v_1} \text{LC}(P_1) P_2 \ldots P_i \xrightarrow{v_i} \text{LC}(P_i) P_{i+1} \ldots$$

there exists a timed execution sequence

$$P_0 \xrightarrow{1} Q_0 \xrightarrow{v_0} P_1 \xrightarrow{1} Q_1 \xrightarrow{v_1} P_2 \ldots P_i \xrightarrow{1} Q_i \xrightarrow{v_i} P_{i+1} \xrightarrow{1} Q_{i+1} \ldots$$

2. For any timed execution sequence from $P_0$

$$P_0 \xrightarrow{1} Q_0 \xrightarrow{v_0} P_1 \xrightarrow{1} Q_1 \xrightarrow{v_1} P_2 \ldots P_i \xrightarrow{1} Q_i \xrightarrow{v_i} P_{i+1} \xrightarrow{1} Q_{i+1} \ldots$$

with infinitely many time steps, the following is an fair-step sequence:

$$P_0 \xrightarrow{v_0} \text{LC}(P_0) P_1 \xrightarrow{v_1} \text{LC}(P_1) P_2 \ldots P_i \xrightarrow{v_i} \text{LC}(P_i) P_{i+1} \ldots$$

5 Transition systems for fair execution sequences and finite state processes

We say that one process is *action-reaching* from another, if it can be reached according to the standard functional operational semantics, i.e. with transi-
For an unlabelled initial process $P \in \mathbb{P}_1$ (i.e. a standard untimed process), we denote by $\mathcal{AT}(P)$ the set of processes action-reachable from $P$; we call $P$ finite state, if $\mathcal{AT}(P)$ is finite.

For the definition of fair executions, we followed Costa and Stirling and introduced two semantic levels: one level (the positive) prescribes the finite and infinite execution sequences of labelled processes disregarding their fairness, while the other (the negative) filters out the unfair ones. The labels are notationally heavy, and keeping track of them is pretty involved. Since the labels evolve dynamically along computations, the transition system defined for the first level is in general infinite state even if the process without labels were finite state (namely if it has at least one infinite computation). Also the filtering mechanism is rather involved, since we have to check repeatedly what happens to live events along the computation, and for this we have to consider and compare the processes passed in the computation.

With the characterization results of the previous section, we have not only shown a conceptional relationship between timing (which is analogous to the timing as used in the PAFAS approach to the efficiency of asynchronous processes) and fairness. We have also given a much lighter description of the fair execution sequences of a process $P \in \mathbb{P}_1$ via the transition system of processes time-reachable (i.e. with transitions $\mathcal{\alpha} \rightarrow$ and $1 \rightarrow$) from $P$, which we will denote by $\mathcal{TT}(P)$: the marking of some actions with underlines is easier than the labelling mechanism, and the filtering simply requires infinitely many time steps, i.e. non-Zeno behaviour; hence, for filtering one does not have to consider the processes passed. Moreover, we will show that the transition system $\mathcal{TT}(P)$ is finite for finite state processes.

**Theorem 5.1** If $P \in \mathbb{P}_1$ is finite state, then $\mathcal{TT}(P)$ is finite.

The main result in [5,6] is a characterization of fair execution sequences with only one (positive) level: SOS-rules are given that describe all transitions $P \xrightarrow{v} Q$ with $v \in (A_{\tau})^*$ such that $P \xrightarrow{v,\text{LC}(P)} Q$. This is conceptionally very simple, since there is only one level and there is no labelling or marking of processes: the corresponding transition system for $P$ only contains processes reachable from $P$. In particular, the transition system is finite-state if $P$ is finite-state. The drawback is that, in general, $P$ has infinitely many $\text{LC}(P)$-steps (namely, if it has an infinite computation), and therefore the transition system has infinitely many arcs and is infinitely branching. (Observe that this drawback is not shared by our transition system of timed-reachable processes.)

As a second main result, we will now derive from $\mathcal{TT}(P)$ for a finite-state process $P$ a finite transition system with finitely many arcs that describes the fair execution sequences in one level: the essential idea is that the arcs are
inscribed with regular expressions (and not just with sequences as in [5,6]); this idea has already been used for the analogous fairness of actions in [1], but only the construction here has a nice feature as explained below.

The states of the new transition system are the initial processes in $TT(P)$, i.e. the states $Q$ with $Q \stackrel{1}{\to} Q'$; if $R$ is another such state, we have an arc from $Q$ to $R$ labelled with a regular expression $e$. This expression is obtained by taking $TT(P)$ with $Q'$ as initial state and $R$ as the only final state, deleting all transitions $\xrightarrow{1}$ and applying the well-known Kleene construction to get an (equivalent) regular expression from a nondeterministic automaton. (The arc can be omitted, if $e$ describes the empty set.) By Proposition 4.2.1, such an arc corresponds to a set of LC-steps which are also present in the one-level characterization of Costa and Stirling; vice versa, each LC-step is represented by such an arc by Proposition 4.1.1. There is one exception: if $Q' \xrightarrow{1}$, then $Q = Q'$ and $Q$ cannot perform any action; hence, there will only be an $\varepsilon$-labelled arc from $Q$ to itself. With these loops, fair executions correspond to infinite paths in the new transition system, where we replace each $e$-labelled arc on the path by some $v$ in the language of $e$. If we omit the loops, we can take maximal paths instead.

Note that, by definition of time step, the new transition system has only arcs $P \xrightarrow{e} Q$ such that $P$ and $Q$ are initial processes and for each $v$ belonging to $e$ one has $P \xrightarrow{v} Q$. This is a nice property that is not shared by the analogous construction in [1], which considers also states that are not initial. The property is achieved in particular by our specific treatment of recursion, where components in the body of a recursion can be urgent. (In [1] this is not the case; instead, function urgent unfolds a recursion.)

References


