Performance and other non-functional aspects of systems: an approach with PA and TA

Speakers: M.R. Di Berardini, L. Tesei
Dipartimento di Matematica e Informatica
Università di Camerino

Kick-off Meeting, Bertinoro, 23-24 ottobre 2008
Publications

Performance


Timing and Fairness


Liveness Property of Systems

Part I

PAFAS: A Process Algebra for Faster Asynchronous Systems
A Basic Process Algebra

The set of processes is generated by

\[ P := \text{nil} \mid x \mid \alpha.P \mid P + P \mid P \parallel_{A} P \mid P[\Phi] \mid \text{rec } x.P \]

where \( \alpha \in A_{\tau} \) is a basic action (either visible or internal – in the testing framework we assume that \( A \) contains also \( \omega \), the success action), \( A \subseteq \hat{A} \) and \( \Phi \) is a relabeling function.
**PAFAS**

**Basic Assumptions:** actions have an upper time bound – either 0 or 1 – as a maximal delay for their execution. We distinguish between:

- **patient prefixes** (time bound 1, denoted by \(\alpha.P\)): *can either perform \(\alpha\) immediately (and then evolve in \(P\)) or let pass one time unit and become urgent*

- **urgent prefixes** (time bound 0, denoted by \(\overline{\alpha}.P\)): *has to perform \(\alpha\) before the next time-step*

\[
\begin{align*}
\alpha.P & \xrightarrow{1} \overline{\alpha}.P \not\xrightarrow{\frac{1}{a}} \\
\quad \quad \quad \downarrow a \\
\quad \quad \quad \quad \quad \quad P \\
\end{align*}
\]

as a stand-alone process, \(\overline{\alpha}.P\) must perform \(a\) immediately

\[
\begin{align*}
\alpha.P \parallel a \cdot \text{nil} & \xrightarrow{1} \alpha.P \parallel a \cdot \text{nil} \not\xrightarrow{\frac{1}{a}} \\
\quad \quad \quad \quad \downarrow a \\
\quad \quad \quad \quad \quad \quad P \parallel a \cdot \text{nil} \\
\end{align*}
\]

but, as component of a larger system \(\overline{\alpha}.P\) can wait for a synchronization
Transitional Semantics of PAFAS

1. Functional Behaviour

\[ Q \xrightarrow{\alpha} Q' \]  

Q evolves into Q’ by performing the action \( \alpha \)

2. Refusal Behaviour

\[ Q \xrightarrow{X} Q' \]  

It is a conditional time step (of duration 1). \( X \) is a set of actions that are not just waiting for a synchronization i.e. these actions are not urgent and can be refused by Q. These steps can take part in a ‘real’ time step only in a suitable environment.

Whenever \( X = A \), Q perform a (full) time-step, \( Q \xrightarrow{1} Q' \)

Initial processes  \( \tilde{P}_1 \) ranged over \( P, P_1, \ldots, P', \ldots \)

General processes  \( \tilde{P} \) ranged over \( Q, Q_1, \ldots, Q', \ldots \)
The Functional Behaviour of PAFAS-terms

\[
\begin{align*}
\text{Act}_1: & \quad \alpha.P \xrightarrow{\alpha} P \\
\text{Act}_2: & \quad \alpha.P \xrightarrow{\alpha} P \\
\text{Sum:} & \quad Q_1 \xrightarrow{\alpha} Q', \quad Q_1 + Q_2 \xrightarrow{\alpha} Q' \\
\text{Synch:} & \quad \alpha \in A, \quad Q_1 \xrightarrow{\alpha} Q_1', \quad Q_2 \xrightarrow{\alpha} Q_2' \quad Q_1 \parallel_A Q_2 \xrightarrow{\alpha} Q_1 \parallel_A Q_2' \\
\text{Par:} & \quad \alpha \notin A, \quad Q_1 \xrightarrow{\alpha} Q_1' \quad Q_1 \parallel_A Q_2 \xrightarrow{\alpha} Q_1 \parallel_A Q_2' \\
\end{align*}
\]

The other rules are as expected
The Refusal Behaviour of PAFAS-terms

\[
\begin{align*}
\text{NIL}_r & \quad \text{nil} \xrightarrow{X} r \text{nil} \\
\text{ACT}_1 & \quad \alpha.P \xrightarrow{X} r \alpha.P \\
\text{ACT}_2 & \quad \alpha \notin X \cup \{\tau\} \\
\end{align*}
\]

\[
\begin{align*}
\text{SUM}_r & \quad Q_1 \xrightarrow{X} r Q_1', Q_2 \xrightarrow{X} r Q_2' \\
\quad & \quad Q_1 + Q_2 \xrightarrow{X} r Q_1' + Q_2' \\
\text{PAR}_r & \quad Q_1 \xrightarrow{x_1} r Q_1', Q_2 \xrightarrow{x_2} r Q_2', X \subseteq (A \cap (X_1 \cup X_2)) \cup ((X_1 \cap X_2) \setminus A) \\
\text{PAR}_r & \quad Q_1 \parallel_A Q_2 \xrightarrow{X} r Q_1' \parallel_A Q_2'
\end{align*}
\]
Notation:

- The *timed transition system* $\text{TTS}(Q)$ of $Q$ consists of all transitions $R \xrightarrow{\mu} R'$ with $\mu \in A_\tau$ or $\mu = 1$ where $R$ is reachable from $Q$ via such transitions.

- $\text{DL}(Q) = \{ v \mid Q \xrightarrow{v} \}$ $\tau$’s are abstracted away; it contains the *discrete traces* of $Q$.

- The *refusal transition system* $\text{RTS}(Q)$ of $Q$ consists of all transitions $R \xrightarrow{\alpha} R'$ or $R \xrightarrow{X} R'$ where $R$ is reachable from $Q$ via such transitions.

- $\text{RT}(Q) = \{ v \mid Q \xrightarrow{\mu} \}$; it contains the *refusal traces* of $Q$. 
Performance Measures

Based on PAFAS, we provide two different performance measures:

1. A testing-based faster-than (preorder) relation that compares the worst-case efficiency of asynchronous systems (this is a qualitative measure).

2. A performance function that gives for each user behaviour the worst-case time needed to satisfy the user (a quantitative one).
The Testing Preorder

- A **timed test** is a pair $(O, D)$ where:
  - $O$ is a test process (can perform $\omega$ – the success action)
  - $D \in \mathbb{N}_0$ is a time bound

- A testable process $Q$ **satisfies** a test $(O, D)$, i.e. $Q \text{ must}(O, D)$, if any $v \in DL(Q \parallel O)^1$ whose duration $\zeta(v) > D$ contains some $\omega$

- $Q$ is **faster than** $Q'$, written $Q \preceq Q'$, if $Q'$ must$(O, D)$ implies $Q \text{ must}(O, D)$ for all timed tests $(O, D)$

- **Theorem** (Characterization of the testing preorder – (1)): Let $Q, Q'$ be two testable processes. $Q \preceq Q'$ iff $\text{RT}(Q) \subseteq \text{RT}(Q)$

This provides a decidability result for the preorder for finite-state processes

---

$^1 \parallel$ is a shorthand for $\parallel_{A \setminus \{\omega\}}$
Three Different Implementations of a Bounded Buffer
Three Different Implementations of a Bounded Buffer

- $Fifo \nsubseteq Pipe$ and $Pipe \nsubseteq Fifo$
- $Fifo \subseteq Buff$ and $Buff \nsubseteq Fifo$
- If $n = 1$ then $Pipe \subseteq Buff$, otherwise $Pipe \nsubseteq Buff$; $Buff \nsubseteq Pipe$
Performance Function

- For a testable process $Q$ and a test process $O$, the *performance function* $p$ is defined by

$$p(Q, O) = \sup \{ n \in \mathbb{N}_0 \mid \exists v \in DL(Q \parallel O) : \zeta(v) = n \text{ and } v \text{ does not contain } \omega \}$$

- The *performance function* $p_Q$ of $Q$ is defined by $p_Q(O) = p(Q, O)$

- If $D = p(Q, O)$ then any $v \in DL(Q \parallel O)$ with $\zeta(v) > D$ contains some $\omega$; in other terms, $p(Q, O)$ gives the worst-case time to reach the satisfaction of $Q$

- **Proposition** – Quantitative formulation of the faster-than preorder (2): $Q \sqsubseteq Q'$ iff $p(Q, O) \leq p(Q', O)$ for all tests $O$, i.e. iff $p_Q \leq p_{Q'}$
Response Performance

Consider the following specifications

\[
\text{Seq} = \text{rec } x. \overline{\text{in}}.\tau.\text{out}.x \\
\text{Pipe} = (\text{rec } x. \overline{\text{in}}.s.x \parallel \{s\} \text{ rec } x. \overline{s}.\text{out}.x) / s
\]

One would expect that \text{Pipe} is faster than \text{Seq} since it allows more parallelism; but it turn out that this is not true.

This is because \text{Pipe} is not a functional refinement of \text{Seq}: the former can perform the sequence \text{in in} while the latter cannot.

The expectation that \text{Pipe} is faster than \text{Seq} is based on some assumption about the users.

We want to compare these processes w.r.t. their ability to answer a given number of requests as fast as possible.
Response Performance

- This class $\mathcal{U}$ of user behaviours can be defined by

$$U_1 = \textbf{in} \cdot \textbf{out} \cdot \omega$$
$$U_{n+1} = U_n \| \omega \cdot \textbf{in} \cdot \textbf{out} \cdot \omega$$

- With this assumption on the class of users, one can turn the function $p_Q$ into a function that we call response performance

$$rp_Q : \mathbb{N} \rightarrow \mathbb{N}_0 \cup \{\infty\}$$
$$rp_Q(n) = p_Q(U_u) = p(Q, U_n)$$

- In (2) it is shown how to determine the response performance for the so-called response processes, i.e. processes that cannot produce more responses (i.e. $\textbf{out}$) than requests (i.e. $\textbf{in}$)
The Reduced Refusal Transition System

- $p_Q$ (and hence $rp_Q$) can be determined from the $TTS(Q \parallel O)$, which in turn can be determined from the $RTS(Q)$ and $RTS(O)$.

- Due to our assumption on users, some interesting fact about $rp_Q$ can be deduced by considering only $RTS(Q)$.

- For a response process $Q$ the *reduced refusal transition system* $rRTS(Q)$ of $Q$ is obtained from the $RTS(Q)$ as follows:
  - we keep all actions transitions,
  - we keep a time step $Q \xrightarrow{X} Q'$ iff either (i) $X = \mathbb{A}$ or (ii) $X = \{out\}$ and $Q$ has a positive number of the pending $out$ actions,
  - we delete all processes that are not reachable anymore.

- Basically, we remove time steps that cannot participate in full time step when considering the behaviour of $Q \parallel U_n$. 

---

**Performance Measures**

- A quantitative approach to performance
Performance Measures

A quantitative approach to performance

\[ \text{RTS}(\text{Seq}) \]

\[ \text{Seq} \xrightarrow{\text{in}} \tau\text{.out}.\text{Seq} \xrightarrow{\tau, \text{in}} \text{out}.\text{Seq} \xrightarrow{\text{out}} \text{out}.\text{Seq} \]

\[ \text{seq} \xrightarrow{\text{in}} \tau\text{.out}.\text{seq} \xrightarrow{\tau, \text{in}} \text{out}.\text{seq} \xrightarrow{\text{out}} \text{out}.\text{seq} \]

\[ \text{rRTS}(\text{Seq}) \]

\[ \text{Seq} \xrightarrow{\text{in}} \tau\text{.out}.\text{Seq} \xrightarrow{\tau, \text{in}} \text{out}.\text{Seq} \xrightarrow{\text{out}} \text{out}.\text{Seq} \]

\[ \text{seq} \xrightarrow{\text{in}} \tau\text{.out}.\text{seq} \xrightarrow{\tau, \text{in}} \text{out}.\text{seq} \xrightarrow{\text{out}} \text{out}.\text{seq} \]
Bad-cycle Theorem

Let $Q$ a response process

- A cycle in $\text{rRTS}(Q)$ is **catastrophic** if it contains a positive number of time steps but no *in’s* and no *out’s* (along this cycle time increases without limits, but no ‘useful’ actions are performed).
- For a $Q$ without catastrophic cycles, we consider cycles that may be reached from $Q$ by a path where all time steps are full and which themselves contains only full time steps.
- The **average performance** of such a cycle as the number of its time steps divided by the number of the *in’s* in this cycle.
- We call a cycle bad if it is a cycle of maximal average performance in $\text{rRTS}(Q)$.
- **Theorem** (Bad cycles theorem – (2)): $Q$ has a catastrophic cycle iff its response performance is $\infty$. For $Q$ without catastrophic cycles, the response performance of $Q$ is asymptotically linear and its asymptotic factor is the average performance of a bad cycle.
Performance Measures

A quantitative approach to performance

\[ rRTS(\text{Seq}) \]

Figure: a cycle with average performance 1
Figure: a cycle with average performance $2 - r_{p_{Seq}}(n) = 2n$
rRTS(Pipe)

Figure: two bad cycles with average performance \( 1 - r_{PP}(n) = n + 1 \)
Theorem – see (2): Let $Q$ a finite-state response process and $n$ the number of states of the rRTS($Q$)

- It can be decided in $O(n^3)$ time whether $Q$ has a catastrophic cycle
- If no catastrophic cycles exist, the average performance of $Q$ can be computed in $O(n^3)$ time

FastAsy is an automated tool that allows us to

- compare processes w.r.t. the testing preorder
- check whether a process has a catastrophic cycle, and if this is not the case. to compute its average performance
Part II

Timing and Fairness
Timing and Fairness

**Timing** gives information on when actions are performed and can serve as a basis for considering efficiency.

**Fairness** requires that a system activity which is continuously enabled along a computation will proceed

- **Weak Fairness of Actions/Components**: an action/a component continuously enabled along a computation must eventually proceed

- **Strong Fairness of Actions/Components**: an action/a component enabled infinitely often along a computation proceed infinitely often

We relate:

Weak Fairness of Actions as defined by Costa & Stirling

and the

PAFAS timed operational semantics

Our main results:

1. all non-Zeno (or everlasting) timed process executions are fair
2. a characterization of fair executions of untimed processes in terms of timed process executions
3. a finite representation of fair executions using regular expressions.
Costa & Stirling (Weak) Fairness of Actions

The main ingredients of this theory are:

- **A Labeling for process terms:** this allows us to detect, during a transition, which action is actually performed as, for instance, in $P = \text{rec } x.a.x \parallel \emptyset \text{ rec } x.a.x \xrightarrow{\ a \ } P$

- **Live events:** an action/event of a process term is **live** if it can currently be performed, as action $a$ in
  
  $$a.b.nil \parallel \{b\} \ b.nil$$

  $b$ is not live ... at the moment

- **Fair sequences:** a maximal sequence is fair when no event becomes live and then remains live throughout
Labeling for process terms

We need a labeling function $L$ that attaches labels (strings in $\{1, 2\}^*$) to process terms. It must satisfy the following properties

- **Unicity of Labels:** no label occurs more than once in a term
- **Persistence and Disappearance of Labels under derivations:** once a label disappears it can never reappear

\[
\begin{align*}
L_u(nil) &= nil_u, \quad L_u(x) = x_u \\
L_u(\mu.P) &= \{\mu_u.P' \mid P' \in L_{u1}(P)\} \\
L_u(Q_1 + Q_2) &= \{Q'_1 + u Q'_2 \mid Q'_1 \in L_{u1}(Q_1) \text{ and } Q'_2 \in L_{u2}(Q_2)\} \\
L_u(\text{rec } x.Q) &= \{\text{rec } x_u.Q' \mid Q' \in L_{u1}(Q)\} \ldots
\end{align*}
\]

Example: $L_\epsilon((a.nil + b.nil) + c.nil) = (a_{11}.nil_{111} \parallel b_{12}.nil_{121}) + c_{2}.nil_{21}$
Changes in the Operational Semantics

\[
\begin{align*}
\text{ACT}_1 & \quad \alpha \cdot P \xrightarrow{\alpha} P \\
\text{ACT}_2 & \quad \alpha \cdot P \xrightarrow{\alpha} P \\
\text{REC} & \quad Q\{\text{rec } x \cdot Q/x\} \xrightarrow{\alpha} Q' \\
& \quad \text{rec } x \cdot Q \xrightarrow{\alpha} Q'
\end{align*}
\]

In \( Q\{R/x\} \), each substituted \( R \) inherits the label of the \( x \) it replaces. Ex: if \( R = \text{rec } x \cdot a_{u1}.x_{u11} \) then

\[
(a_{u1}.x_{u11})\{R/x\} = a_{u1}.\text{rec } x_{u11}.a_{u111}.x_{u1111} \xrightarrow{a} \text{rec } x_{u11}.a_{u111}.x_{u1111}
\]

Thus, labeling is \textbf{dynamic}
Live Events

Tuples of labels associated with enabled actions, i.e. actions that can be immediately performed:

\[
\text{LE}(a_{u1}.\text{nil}) = \{\langle u1 \rangle\}
\]
a live \( a \)-event identified by \( \langle u1 \rangle \)

\[
\text{LE}(a_{u21}.\text{nil} + u_2 \ b_{u22}.\text{nil}) = \{\langle u21 \rangle, \langle u22 \rangle\}
\]
two live events (an \( a \)-event and a \( b \)-event) identified by \( \langle u21 \rangle \) and \( \langle u22 \rangle \), resp.

\[
\text{LE}(a_{u1}.\text{nil} \parallel_{\{a\}} (a_{u21}.\text{nil} + u_2 \ b_{u22}.\text{nil})) = \{\langle u22 \rangle, \langle u1, u21 \rangle\}
\]
a \( b \)-event identified by \( \langle u22 \rangle \), and
an \( a \)-event identified by \( \langle u1, u21 \rangle = \langle u1 \rangle \times \langle u21 \rangle \)

The tuple of a synchronized event (as the \( a \)-event) is obtained by composing
the tuples of the events in the left-hand and in the right-hand side
Let $P \in L(\tilde{P}_1)$ an initial and labeled process term. A maximal sequence of transitions $P = Q_0 \xrightarrow{\gamma_0} Q_1 \xrightarrow{\gamma_1} \ldots$ is:

(i) an execution sequence if $\gamma_i \in A_\tau$, for each $i \geq 0$

(ii) a timed execution sequence if $\gamma_i \in (A_\tau \cup \{1\})$, for each $i \geq 0$.

It is everlasting or non-Zeno if it contains an infinite number of 1.

We say that a (timed) execution sequence $Q_0 \xrightarrow{\gamma_0} Q_1 \xrightarrow{\gamma_1} \ldots$ is fair if

$\neg (\exists$ a tuple $s, \exists i . \forall k \geq i : s \in LE(Q_k))$
A Local Characterization of Fair Sequences

- The sequence of transitions $Q_0 \xrightarrow{\gamma_0} Q_1 \xrightarrow{\gamma_1} \ldots \xrightarrow{\gamma_{n-1}} Q_n$ is a (timed) LE-step if
  \[ \text{LE}(Q_0) \cap \text{LE}(Q_1) \cap \ldots \cap \text{LE}(Q_n) = \emptyset \]
  In such a case, we write $Q_0 \xrightarrow{v_{\text{LE}(Q_0)}} Q_n$ where $v = \gamma_0 \gamma_1 \ldots \gamma_{n-1}$

- An LE-step is a **locally fair step**: all events that are live in $Q_0$ lose their liveness at some point during the computation

- *(Timed) fair-step sequences* are maximal sequences of the form
  \[ Q_0 \xrightarrow{v_0}_{\text{LE}(Q_0)} Q_1 \xrightarrow{v_1}_{\text{LE}(Q_1)} Q_2 \xrightarrow{v_2}_{\text{LE}(Q_2)} \ldots \]

- **Theorem** (Costa & Stirling):
  
  *An execution is fair if and only if it is the sequence associated with a fair-step sequence*
Drawbacks of this approach

- To keep track of the different instances of system activities along a system execution, Costa and Stirling associate labels to actions
- They obtain all fair computations of $P$ by means of a criterion that considers labels along maximal runs
- But, new labels are created dynamically during the system evolution with the immediate effect of changing the syntax of the terms. Ex: if $R = \text{rec } x.u.a_{u1}.x_{u11}$ then
  
  $$R \xrightarrow{a} \text{rec } x_{u11}.a_{u111}.x_{u1111} \xrightarrow{a} \text{rec } x_{u1111}.a_{u11111}.x_{u111111} \xrightarrow{a} \ldots$$

- Thus, cycles in the transition system of a labeled process are not possible as even finite state processes (as $\text{rec } x.a.x$) usually become infinite-state
Our idea

- Instead of labels, we can use the timing information attached to a PAFAS-term to decide if a certain sequence of actions is a locally fair step.

- Let \( P = \text{rec } x. \ a.x \parallel \emptyset \ a \) (for simplicity, here \( P \) is unlabeled). Each LE-step from \( P \) consists of a number of actions \( a \) (also infinite); the last of them is the one performed by the right-hand side component.

- By our operational semantics \( P \xrightarrow{1} Q = a.\text{rec } x. \ a.x \parallel \emptyset \ a \)

\[
Q \xrightarrow{a} \text{rec } x. \ a.x \parallel \emptyset \ a = Q' \xrightarrow{a} \ldots \xrightarrow{a} Q' \xrightarrow{a} \text{rec } x.a.x \parallel \emptyset \text{ nil}
\]

- Notice that \( Q' \xrightarrow{1} \) while \( \text{rec } x.a.x \parallel \emptyset \text{ nil} = P' \xrightarrow{1} \)

- Thus, each LE-step of \( P \) corresponds to a sequence of timed steps of the form \( P \xrightarrow{1} Q \xrightarrow{\nu} P' \xrightarrow{1} \)
LE-steps and 1-1 Transitions – (4)

Let $P_0 \in L(\tilde{P}_1)$ and $v, w \in A^*_\tau$.

1. If $P_0 \xrightarrow{1} Q_0 \xrightarrow{v} P_1 \xrightarrow{1} \text{ then } P_0 \xrightarrow{1v}_{\text{LE}(P_0)} P_1$

2. If $P_0 \xrightarrow{v} P_1 \xrightarrow{1} Q_1 \xrightarrow{w} P_2 \xrightarrow{1} \text{ then } P_0 \xrightarrow{v1w}_{\text{LE}(P_0)} P_2$

3. $P_0 \xrightarrow{v}_{\text{LE}(P_0)} P_1$ implies $P_0 \xrightarrow{1} Q_0 \xrightarrow{v} P_1 \xrightarrow{1}$

These results required some modification to the (original) PAFAS timed operational semantics
Cleaning inactive markings

- $P_0 = a_1.\text{nil} \parallel^\epsilon a (a_{21}.\text{nil} + 2 c_{22}.a_{221}.\text{nil})$
- $P_0 \xrightarrow{c}_{\text{LE}(P)} a_1.\text{nil} \parallel^\epsilon a_{221}.\text{nil} = P_1$
- $P \xrightarrow{1} a_1.\text{nil} \parallel^\epsilon (a_{21}.\text{nil} + 2 c_{22}.\text{nil}) \xrightarrow{c} a_1.\text{nil} \parallel^\epsilon a_{221}.\text{nil} = Q$
- $Q$ is different from $P_1$, but such processes have the same behaviour because the marking on the left-hand side is not “active”
- We have defined a function clean( ) that removes such markings

\[
\begin{align*}
\text{SYNCH} & \quad \alpha \in A, \quad Q_1 \xrightarrow{\alpha} Q_1', \quad Q_2 \xrightarrow{\alpha} Q_2' \quad \Rightarrow \quad Q_1 \parallel_A Q_2 \xrightarrow{\alpha} \text{clean}(Q_1' \parallel_A Q_2') \\
\text{PAR} & \quad \alpha \notin A, \quad Q_1 \xrightarrow{\alpha} Q_1' \quad \Rightarrow \quad Q_1 \parallel_A Q_2 \xrightarrow{\alpha} \text{clean}(Q_1' \parallel_A Q_2) \\
\text{PAR}_r & \quad Q_1 \xrightarrow{X_1 \cap r} Q_1', \quad Q_2 \xrightarrow{X_2 \cap r} Q_2', \quad X \subseteq (A \cap (X_1 \cup X_2)) \cup ((X_1 \cap X_2) \setminus A) \quad \Rightarrow \quad Q_1 \parallel_A Q_2 \xrightarrow{X \cap r} \text{clean}(Q_1' \parallel_A Q_2')
\end{align*}
\]
Cleaning inactive markings

\[ \text{clean}(Q) = \text{clean}(Q, \emptyset) \] where \( \text{clean}(Q, A) \) is defined by (\( A \) represents the set of actions that have to lose their urgency)

\[
\begin{align*}
\text{clean}(\text{nil}, A) &= \text{nil}, & \text{clean}(x, A) &= x \\
\text{clean}(\alpha.P, A) &= \alpha.P & \text{clean}(\underline{\alpha}.P, A) &= \begin{cases} 
\alpha.P & \text{if } \alpha \in A \\
\underline{\alpha}.P & \text{otherwise}
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\text{clean}(Q_1 + Q_2, A) &= \text{clean}(Q_1, A) + \text{clean}(Q_2, A) \\
\text{clean}(Q_1 \parallel_B Q_2, A) &= \text{clean}(Q_1, (B \setminus \mathcal{U}(Q_2)) \cup A) \parallel_B \text{clean}(Q_2, (B \setminus \mathcal{U}(Q_1)) \cup A) \\
\text{clean}(Q[\Phi], A) &= \text{clean}(Q, \Phi^{-1}(A))[\Phi] \\
\text{clean}(\text{rec } x.Q, A) &= \text{rec } x.\text{clean}(Q, A)
\end{align*}
\]
An alternative Characterization of Fair Traces

Changes to the original PAFAS timed operational semantics

Unfolding of terms

- \( P_0 = \text{rec } x_1. a_{11}.x_{111} \parallel^\varepsilon a_21.\text{nil} + 2 c_22.a_{221}.\text{nil} \)

- \( P_0 \xrightarrow{c_{\text{LE}(P)}} \text{rec } x_1. a_{11}.x_{111} \parallel^\varepsilon a_{221}.\text{nil} = P_1 \)

- If \( u = 111 \) then:
  \[
  P \xrightarrow{1} a_{11}.(\text{rec } x_u. a_{u1}.x_{u11}) \parallel^\varepsilon (a_{21}.\text{nil} + 2 c_22.a_{221}.\text{nil})
  \]
  \[
  \xrightarrow{c} a_{11}.(\text{rec } x_u. a_{u1}.x_{u11}) \parallel^\varepsilon a_{221}.\text{nil} = Q
  \]

- Up to unfolding, \( Q \) and \( P_1 \) have exactly the same behaviour

\[
\begin{align*}
\text{REC}_r & \quad Q \xrightarrow{X} r \quad Q' \\
\text{REC} & \quad Q \{ \text{rec } x_u. \text{unmark}(Q)/x \} \xrightarrow{\alpha} r \quad Q'
\end{align*}
\]

where \( \text{unmark}(Q) \) is the process we obtain from \( Q \) by removing all markings (inactive or not)
Unfolding of terms

With these new rules:

- $P_0 = \text{rec } x_1 \cdot a_{11}.x_{111} \parallel^\varepsilon a \cdot (a_{21}.\text{nil} + 2 \cdot c_{22} \cdot a_{221}.\text{nil})$
- $P_0 \xrightarrow{c}_{\text{LE}(P)} \text{rec } x_1 \cdot a_{11}.x_{111} \parallel^\varepsilon a \cdot a_{221}.\text{nil} = P_1$
- $P \xrightarrow{1} Q = \text{rec } x_1 \cdot a_{11}.x_{111} \parallel^\varepsilon a \cdot (a_{21}.\text{nil} + 2 \cdot c_{22} \cdot a_{221}.\text{nil})$
- $P \xrightarrow{c} \text{rec } x_1 \cdot a_{11}.x_{111} \parallel^\varepsilon a \cdot a_{221}.\text{nil} = P_1$

Moreover:

\[
(a_{11}.x_{111})\{\text{rec } x_1 \cdot \text{unmark}(a_{11}.x_{111})/x\} = (a_{11}.x_{111})\{\text{rec } x_1 \cdot a_{11}.x_{111}/x\} = a_{11}\cdot\text{rec } x_u \cdot a_{u1}.x_{u11} \xrightarrow{a} \text{rec } x_u \cdot a_{u1}.x_{u11}
\]

(where, again, $u = 111$) and hence, $Q \xrightarrow{a} \text{rec } x_u \cdot a_{u1}.x_{u11} \parallel^\varepsilon a \cdot \text{nil}$
An alternative Characterization of Fair Traces

Main results

Fairness of everlasting timed execution sequences

Each everlasting timed execution sequence of the form:

\[ Q_0 \xrightarrow{\nu_0} R_1 \xrightarrow{1} Q_1 \xrightarrow{\nu_1} R_2 \xrightarrow{1} Q_2 \xrightarrow{\nu_2} R_3 \xrightarrow{1} \ldots \]

where \( \nu_0, \nu_1, \nu_2, \ldots \in \mathbb{A}_T^* \) is fair (because it is associated with a timed fair-step sequence)
Characterization of Fair Executions – The Infinite Case

Let $P \in L(\tilde{P}_1)$ and $\nu_0, \nu_1, \nu_2, \ldots \in A^*$. For any infinite fair-step sequence from $P$

$$P = P_0 \xrightarrow{v_0} LE(P_0) \xrightarrow{v_1} LE(P_1) \xrightarrow{v_2} LE(P_2) \ldots$$

there is a timed execution sequence

$$P_0 \xrightarrow{1} Q_0 \xrightarrow{v_0} P_1 \xrightarrow{1} Q_1 \xrightarrow{v_1} P_2 \xrightarrow{1} Q_2 \xrightarrow{v_2} P_2 \ldots$$

and vice versa
Characterization of Fair Executions – The Infinite Case

Let $P \in L(\tilde{P}_1)$ and $v_0, v_1, v_2, \ldots \in A^*$. For any infinite fair-step sequence from $P$

$$P = P_0 \xrightarrow{v_0} \text{LE}(P_0) P_1 \xrightarrow{v_1} \text{LE}(P_1) P_2 \xrightarrow{v_2} \text{LE}(P_2) \ldots$$

there is a timed execution sequence

$$R(P_0) \xrightarrow{1} S_0 \xrightarrow{v_0} R(P_1) \xrightarrow{1} S_1 \xrightarrow{v_1} R(P_2) \xrightarrow{1} S_2 \xrightarrow{v_2} R(P_2) \ldots$$

and vice versa
A Transition System for Fair Execution Sequences

Let $P$ be finite state process (according to the standard operational semantics) and consider the transition system $\text{TTS}(P)$ (all processes reachable from $P$ via $\alpha \rightarrow$ and $1 \rightarrow$).

The fair timed transition system of $P$, written $\text{FairTTS}(P)$, is obtained as follows:

1. the states of $\text{FairTTS}(P)$ are those states $Q$ in $\text{TTS}(P)$ with $Q \xrightarrow{1}$
2. if $Q$ and $R$ are two of such states, add an arc between them labeled with a regular expression $e$. If $Q \xrightarrow{1} Q'$, this expression is built as described below

- take $\text{TTS}(P)$ with $Q'$ as initial state and $R$ as the only final one
- delete all transitions $\xrightarrow{1}$ (and all states do not reachable any more)
- apply the Kleen construction to get a regular expression from a NFA
FairTTS – An Example

\[ a \rightarrow b \rightarrow c \rightarrow d \]

\[ a + bc \]

\[ d \]
Advantages of our approach

- We also change the syntax of processes, in our case by adding timing information, but this is much simpler that the syntax of labels and leaves finite-state processes finite state
- Then we apply a simple filter that does not consider processes: we simply require that infinitely many time steps occur in a run.
- As a small price, we have to project away these time steps in the end
- EX: all fair runs of $P = \text{rec } x.a.x$ can be obtained by considering the non-Zeno run of the form:

$$P \xrightarrow{1} \text{rec } x.a.x \xrightarrow{a} P \xrightarrow{1} \xrightarrow{a} P \ldots$$
Part III

From Fairness of Actions to Fairness of Components
PAFAS and Fairness of Components

- PAFAS is not a suitable abstraction for **Fairness of Components** as it is for fairness of actions.

- We have found a variation of PAFAS with slightly different terms and operational semantics (this is called PAFAS\(^c\)) that allows us to characterize Costa & Stirling Fairness of Components.

- The results we have obtained are conceptually the same as those for fairness of actions (also in this case we can characterize fair runs in terms on timed non-Zeno runs, ...), but a number of changes were needed to define the new semantics.
Costa & Stirling (Weak) Fairness of Components

It closely follows the theory of Fairness of Actions:

- **A Labeling for process terms**: this labeling allows us to detect which component actually moves during a transition.
- **Live Components**: an component of a process term is live if it can currently contribute to a move.
- **Fair sequences**: a maximal sequence is fair when no component becomes live and then remains live throughout.
Initial process terms are (also in this case) generated by

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

but now upper time bounds (again 0 or 1) are associated with parallel components of a process term. We distinguish between:

- **patient components** (time bound 1) denoted by \( \alpha.P \) and \( P_1 + P_2 \)
  - can perform some action within time 1

- **urgent components** (time bound 0) denoted by \( \alpha.P \) and \( P_1 \parallel P_2 \)
  - urgent component has to act in zero time or get disabled
Some Differences

- Time passes marking as urgent all enabled components, i.e. all components that can currently contribute to a move.

- Components can lose their urgency only if their actions are no longer enabled due to changes of context.

- The next time step will be only possible if no components are marked as urgent.

\[
\begin{align*}
a \parallel \{a\}(a + c.a) & \overset{1}{\rightarrow} a \parallel \{a\}(a + c.a) \\
(a + b) \parallel \{a,b\}(a + c.(b + d)) & \overset{1}{\rightarrow} (a + b) \parallel \{a,b\}(a + c.(b + d)) \\
& \overset{c}{\rightarrow} (a + b) \parallel \{a,b\}(b + d) \\
& \overset{d}{\rightarrow} (a + b) \parallel \{a,b\}\text{nil}
\end{align*}
\]
The Functional Behaviour of PAFAS\textsuperscript{c}-terms

\[
\begin{align*}
\text{Act}_1: & \quad \alpha.P \xrightarrow{\alpha} P \\
\text{Act}_2: & \quad \alpha.P \xrightarrow{\alpha} P \\
\text{Sum}: & \quad Q_1 \xrightarrow{\alpha} Q' \\
& \quad Q_1 + Q_2 \xrightarrow{\alpha} Q' \\
\text{Synch}: & \quad \alpha \in A, \ Q_1 \xrightarrow{\alpha} Q'_1, \ Q_2 \xrightarrow{\alpha} Q'_2 \\
& \quad Q_1 \parallel_A Q_2 \xrightarrow{\alpha} \text{clean}(Q'_1 \parallel_A Q'_2) \\
\text{Par}: & \quad \alpha \notin A, \ Q_1 \xrightarrow{\alpha} Q'_1 \\
& \quad Q_1 \parallel_A Q_2 \xrightarrow{\alpha} \text{clean}(Q'_1 \parallel_A Q_2) \\
\text{Rec}: & \quad Q\{\text{rec } x. \ \text{unmark}(Q)/x\} \xrightarrow{\alpha} Q' \\
& \quad \text{rec } x.Q \xrightarrow{\alpha_r} Q'
\end{align*}
\]
Cleaning inactive markings

\[
\text{clean}(Q) = \text{clean}(Q, \emptyset) \text{ where } \text{clean}(Q, A) \text{ is defined below (A represent the set of actions that have to lose their urgency)}
\]

\[
\begin{align*}
\text{clean}(\text{nil}, A) &= \text{nil} & \text{clean}(x, A) &= x \\
\text{clean}(\alpha. P, A) &= \alpha. P & \text{clean}(\overline{\alpha}. P, A) &= \begin{cases} 
\alpha. P & \text{if } \alpha \in A \\
\overline{\alpha}. P & \text{otherwise}
\end{cases} \\
\text{clean}(P_1 + P_2, A) &= P_1 + P_2 \\
\text{clean}(P_1 \pm P_2, A) &= \begin{cases} 
P_1 + P_2 & \text{if } A(P_1) \cup A(P_2) \subseteq A \\
P_1 \pm P_2 & \text{otherwise}
\end{cases} \\
\text{clean}(Q_1 \parallel_B Q_2, A) &= \text{clean}(Q_1, (B \setminus A(Q_2)) \cup A) \parallel_B \text{clean}(Q_2, (B \setminus A(Q_1)) \cup A) \\
\text{clean}(Q[\Phi], A) &= \text{clean}(Q, \Phi^{-1}(A))[\Phi] \\
\text{clean}(\text{rec } x. Q, A) &= \text{rec } x. \text{clean}(Q, A)
\end{align*}
\]
The Timed Behaviour of PAFAS\textsuperscript{c}-terms

- In order to define the timed behaviour of PAFAS\textsuperscript{c}-terms, we exploit a function \texttt{urgent(\_)} that marks the \textit{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, and, afterwards the marked components have to act in zero time.

- The next time step will only be possible, if no component is marked as urgent.
The Timed Behaviour of PAFAS\textsuperscript{c}-terms

Let $P \in \tilde{P}_1$ be an initial term, then: $P \xrightarrow{1} \text{urgent}(P)$

\[
\begin{align*}
\text{urgent}(\alpha.P) &= \alpha.P \\
\text{urgent}(P_1 + P_2) &= P_1 + P_2 \\
\text{urgent}(a.P_1 \parallel \{a\} a.P_2) &= a.P_1 \parallel \{a\} a.P_2 \\
\text{urgent}(a.P_1 \parallel \{a\} b.P_2) &= a.P_1 \parallel \{a\} b.P_2 \\
\text{urgent}((a.P_1 + c.nil) \parallel \{a\} b.P_2) &= (a.P_1 + c.nil) \parallel \{a\} b.P_2 \\
\text{urgent}((a.P_1 + c.nil) \parallel \{a,c\} b.P_2) &= (a.P_1 + c.nil) \parallel \{a,c\} b.P_2
\end{align*}
\]
Costa & Stirling (Weak) Fairness of Components

It closely follows the theory of Fairness of Actions:

- **A Labeling for process terms:** this labeling allows us to detect which component actually moves during a transition

  \[ L_u(\text{nil}) = \text{nil}_u, \quad L_u(x) = x_u \]

  \[ L_u(\mu. P) = \{ \mu_u. P' \mid P' \in L_{u1}(P) \} \]

  \[ L_u(P_1 + P_2) = \{ P'_1 + u P'_2 \mid P'_1 \in L_{u1}(P_1) \text{ and } P'_2 \in L_{u2}(P_2) \} \]

  \[ L_u(P_1 \pm P_2) = \{ P'_1 \pm u P'_2 \mid P'_1 \in L_{u1}(P_1) \text{ and } P'_2 \in L_{u2}(P_2) \} \]

  \[ L_u(\text{rec } x.Q) = \{ \text{rec } x_u.Q' \mid Q' \in L_{u1}(Q) \} \ldots \]

- **Live Components:** an component of a process term is live if it can currently contribute to a move

- **Fair sequences:** a maximal sequence is fair when no component becomes live and then remains live throughout
Live Components

Labels associated with components that can immediately contribute to the execution of an action:

\[ P = b_{u1}.a_{u11}.nil +_u a_{u2} \quad \text{LE}(P) = \{\langle u1 \rangle, \langle u2 \rangle\} \]
\[ \text{LC}(P) = \{u\} \]

\[ P = b_{u1}.a_{u11}.nil u_{\{a\}} a_{u2} \quad \text{LE}(P) = \{\langle u1 \rangle\} \]
\[ \text{LC}(P) = \{u1\} \]

\[ P = a_{u11}.nil u_{\{a\}} a_{u2} \quad \text{LE}(P) = \{\langle u11, u2 \rangle\} \]
\[ \text{LC}(P) = \{u11, u2\} \]
Fair Executions Sequences

- A (timed) execution sequence $Q_0 \xrightarrow{\gamma_0} Q_1 \xrightarrow{\gamma_1} \ldots$ is **fair** if
  \[ \neg \left( \exists s \exists i . \forall k \geq i : s \in \text{LC}(Q_k) \right) \]

- The sequence of transitions $Q_0 \xrightarrow{\gamma_0} Q_1 \xrightarrow{\gamma_1} \ldots \xrightarrow{\gamma_{n-1}} Q_n$ is a **(timed) LC-step** if
  \[ \text{LC}(Q_0) \cap \text{LC}(Q_1) \cap \ldots \cap \text{LC}(Q_n) = \emptyset \]

- **(Timed) fair-step sequences** are maximal sequences of the form
  
  $Q_0 \xrightarrow{v_0, \text{LC}(Q_0)} Q_1 \xrightarrow{v_1, \text{LC}(Q_1)} Q_2 \xrightarrow{v_2, \text{LC}(Q_2)} \ldots$

- **Theorem** (Costa & Stirling):
  
  An execution is **fair** if and only if it is the sequence associated with a fair-step sequence
Let $P_0 \in L(\tilde{P}_1)$ and $v, w \in A^*_T$.

1. If $P_0 \xrightarrow{1} Q_0 \xrightarrow{v} P_1 \xrightarrow{1} \text{ then } P_0 \xrightarrow{1v}_{LC(P_0)} P_1$

2. If $P_0 \xrightarrow{v} P_1 \xrightarrow{1} Q_1 \xrightarrow{w} P_2 \xrightarrow{1} \text{ then } P_0 \xrightarrow{v1w}_{LC(P_0)} P_2$

3. $P_0 \xrightarrow{v}_{LC(P_0)} P_1$ implies $P_0 \xrightarrow{1} Q_0 \xrightarrow{v} P_1 \xrightarrow{1}$
Part IV

Liveness Property of a MUTEX Algorithm
Dekker’s Algorithm

There are two processes $P_1$ and $P_2$ that compete for enter their critical sections, two request variables $b_1$ and $b_2$ (boolean-valued) and a turn variable $k$ which may take value from $\{1, 2\}$

while true do
begin
  ⟨noncritical section⟩;
  $b_i = true$;
  while $b_j$ do
    while $b_j$ do
      if $k = j$ then begin
        $b_i := false$;
        while $k = j$ do skip;
        $b_i := true$;
      end;
  ⟨critical section⟩; $k := j$; $b_i := false$;
end;
Dekker’s Algorithm

1. \( b_1 := false \)
2. \( k = 1? \)
3. \( b_1 := false \)
4. \( k := 2 \)
5. critical section
6. \( b_2 = false? \)
7. \( b_1 := true \)
8. noncritical section
9. \( k = 2? \)
10. \( NO \)
11. \( b_1 := false \)
12. \( NO \)
13. \( YES \)
14. \( k = 1? \)
15. \( NO \)
16. \( YES \)
17. \( k := 2 \)
18. \( b_1 := false \)
Translating Dekker’s algorithm into PAFAS processes

- Each program variable is represented as a family of processes:

\[
\begin{align*}
B_1(\text{false}) &= b_1 rf \cdot B_1(\text{false}) + (b_1 wf \cdot B_1(\text{false}) + b_1 wt \cdot B_1(\text{true})) \\
B_1(\text{true}) &= b_1 rt \cdot B_1(\text{true}) + (b_1 wf \cdot B_1(\text{false}) + b_1 wt \cdot B_1(\text{true})) \\
K(1) &= kr1 \cdot K(1) + (kw1 \cdot K(1) + kw2 \cdot K(2)) \\
K(2) &= kr2 \cdot K(1) + (kw1 \cdot K(1) + kw2 \cdot K(2))
\end{align*}
\]

- Given \( b_1, b_2 \in \{\text{true}, \text{false}\} \) and \( k \in \{1, 2\} \), we define

\[
PV(b_1, b_2, k) = (B_1(b_1) \parallel \emptyset B_2(b_2)) \parallel \emptyset K(k))
\]
Translating Dekker’s algorithm into PAFAS processes

- The process $P_1$ is represented by (the process $P_2$ has a symmetric representation):

$$
\begin{align*}
P_1 &= \text{req}_1.b_1 wt.P_{11} + \tau.P_1 \\
P_{11} &= b_2 rf.P_{14} + b_2 rt.P_{12} \\
P_{12} &= kr1.P_{11} + kr2.b_1 wf.P_{13} \\
P_{13} &= kr1.b_1 wt.P_{11} + kr2.P_{13} \\
P_{14} &= cs_1.kw2.b_1 wf.P_1
\end{align*}
$$

- The algorithm can be defined as

$$
\text{Dekker} = (((P_1 \parallel \emptyset P_2) \parallel_B \text{PV}(\text{false}, \text{false}, 1)))[\Phi_B]
$$

where $B$ contains all reading and writing actions and the relabeling function $\Phi_B$ makes all actions in $B$ internal.
Liveness Property

- Dekker’s algorithm and its properties have been studied by Walker in a CCS framework (automated analysis with the CWB).
- He was able to prove that the algorithm preserves mutual exclusion, but w.r.t. liveness he was less successful.
- The algorithm is live if whenever at any point in any computation a process $P_i$ requests the execution of its critical section then, in any continuation of that computation, there is a point at which $P_i$ will eventually enter the critical section.
- We expect this property to hold only under a fairness assumption; so we replace ‘computation’ by ‘fair trace’.
- A MUTEX algorithm satisfies its liveness property if any occurrence of req$_i$ in a fair trace is eventually followed by cs$_i$, $i = 1, 2$.

Which Kind of Fairness

- **Theorem – (6):**

  *Each fair trace (w.r.t. fairness of components) of Dekker is live*

- Vice versa, fairness of actions is **not** sufficiently strong to ensure the liveness property

- There are computations fair (w.r.t. fairness of actions) but not live, i.e. along these computation a given \( req_i \) is **never** followed by the corresponding \( cs_i \)

- The proof of this negative result is provided by means of examples

- **Intuition:** fairness of actions still allows computations where a process that tries to **write** a variable (in our case, one of those we use to manage the entry and exit protocol) can indefinitely be blocked by another process that **reads** it

- This is not the case for fairness of components
An example

Consider:
- a program variable \( V = r.V + w.V \)
- a reading activity \( R = r.R \)
- a writing activity \( W = w.W \)

A run from \( P = (R \parallel \emptyset W) \parallel \{r, w\} \) \( V \) consisting of infinitely many \( r \)'s is fair w.r.t. fairness of actions.

Indeed, according to the PAFAS timed operational semantics:

\[
P \xrightarrow{1} Q = (r.R \parallel \emptyset w.W) \parallel \{r, w\} (r.V + w.V) \\
\xrightarrow{r} (R \parallel \emptyset w.W) \parallel \{r, w\} V = P
\]

Each time an \( r \) is performed, \( V \) offers a new (not urgent) synchronization pattern to \( w.W \), i.e. a new instance of the action \( w \) is produced.

Thus:

\[
P \xrightarrow{1} Q \xrightarrow{r} P \xrightarrow{1} Q \xrightarrow{r} P \ldots
\]
In the case of Dekker's Algorithm
An example

- Vice versa, a run from $P$ consisting of infinitely many $r$’s is **not fair w.r.t. fairness of components**
- According to the PAFAS\textsuperscript{c} timed operational semantics
  
  $P \xrightarrow{1} Q = (r.R \parallel \emptyset w.W) \parallel \{r,w\} (r.V \perp w.V)$

  $Q \xrightarrow{r} Q' = (R \parallel \emptyset w.W) \parallel \{r,w\} V \xrightarrow{1} P$

  $\xrightarrow{w} \quad \xrightarrow{w} \quad P$

  This is because the writing component is always enabled (and hence never lose its urgency) while we perform an arbitrary sequence of $r$-actions
Expressiveness of “non-blocking” readings in PA

- Fairness of actions is not sufficiently strong to ensure the liveness property of Dekker’s algorithm.
- Is this problem specific to fairness of actions or it somehow related to the way we represent program variables?
- Non-blocking readings are a special kind of actions used to represent “read” with consuming operations that allow multiple (non-exclusive) concurrent uses of the same resource.
- This kind of non-consuming operations has been successfully studied in the Petri Nets setting.
- Study the impact of such kind of operations in the timing, fairness and liveness properties of systems.
Thank you for your attention