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

(1) F. Corradini, W. Vogler, L. Jenner. Comparing the Worst-case Efficiency of Asynchronous Systems with PAFAS. **Acta Inf**. 38(11/12): 735-792 (2002)

(2) F. Corradini, W. Vogler. Measuring the performance of asynchronous systems with PAFAS. **Theor. Comput. Sci** 335(2-3): 187-213 (2005).

(3) F. Corradini, M. R. Di Berardini, W. Vogler. PAFAS at Work: Comparing the Worst-Case Efficiency of Three Buffer Implementations. **APAQS 2001**: 231-240.

Timing and Fairness

(4) F. Corradini, M. R. Di Berardini, W. Vogler. Relating Fairness and Timing in Process Algebras, CONCUR'03. Extended Version Fairness of Components in System Computations **Acta Inf**. 43(2): 73-130 (2006)

(5) F. Corradini, M. R. Di Berardini, W. Vogler. Fairness of Components in System Computations, EXPRESS'04. Extended Version **Theor. Comput. Sci**. 356(3): 291-324 (2006)

Liveness Property of Systems

(6) F. Corradini, M. R. Di Berardini, W. Vogler. Checking a Mutex Algorithm in a Process Algebra with Fairness. **CONCUR 2006**: 142-157. Extended Version, to appear on **Acta Inf**.

Part I

PAFAS: A Process Algebra for Faster Asynchronous Systems

A Basic Process Algebra

The set of processes is generated by

$$P ::= \mathsf{nil} \mid x \mid \alpha.P \mid P + P \mid P\|_A P \mid P[\Phi] \mid \mathsf{rec}\, x.P$$

where $\alpha \in \mathbb{A}_\tau$ is a basic action (either visible or internal – in the testing framework we assume that $\mathbb{A}$ contains also ω, the success action), $A \subseteq \mathbb{A}$ and Φ is a relabeling function

$$\textsc{Act}\ \frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \textsc{Sum}\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} + \text{symm.}$$

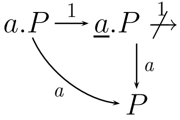
$$\textsc{Synch}\ \frac{\alpha \in A, P \xrightarrow{\alpha} P', Q \xrightarrow{\alpha} Q'}{P\|_A Q \xrightarrow{\alpha} P'\|_A Q'} \qquad \textsc{Par}\ \frac{\alpha \notin A, P \xrightarrow{\alpha} P'}{P\|_A Q \xrightarrow{\alpha} P'\|_A Q} + \text{symm.}$$

other rules are as expected

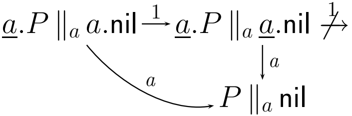
PAFAS

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 α immediately (and then evolve in P) or let pass one time unit and become urgent*
- **urgent prefixes** (time bound 0, denoted by $\underline{\alpha}.P$): *has to perform α before the next time-step*



as a stand-alone process,
$\underline{a}.P$ must perform a immediately



but, as component of a larger system
$\underline{a}.P$ can wait for a synchronization

Transitional Semantics of PAFAS

1. Functional Behaviour

$$Q \xrightarrow{\alpha} Q' \quad Q \text{ evolves into } Q' \text{ by performing the action } \alpha$$

2. Refusal Behaviour

$Q \xrightarrow{X}_r Q'$ 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 action 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 $X = \mathbb{A}$, Q perform a (full) time-step, $Q \xrightarrow{1} Q'$

Initial processes	$\tilde{\mathbb{P}}_1$	ranged over $P, P_1, \ldots, P', \ldots$
General processes	$\tilde{\mathbb{P}}$	ranged over $Q, Q_1, \ldots, Q', \ldots$

The Functional Behaviour of PAFAS-terms

$$\textsc{Act}_1\frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \textsc{Act}_2\frac{}{\underline{\alpha}.P \xrightarrow{\alpha} P}$$

$$\textsc{Sum}\frac{Q_1 \xrightarrow{\alpha} Q'}{Q_1 + Q_2 \xrightarrow{\alpha} Q'} + \text{Symm.}$$

$$\textsc{Synch}\frac{\alpha \in A,\ Q_1 \xrightarrow{\alpha} Q_1',\ Q_2 \xrightarrow{\alpha} Q_2'}{Q_1 \|_A Q_2 \xrightarrow{\alpha} Q_1' \|_A Q_2'}$$

$$\textsc{Par}\frac{\alpha \notin A,\ Q_1 \xrightarrow{\alpha} Q_1'}{Q_1 \|_A Q_2 \xrightarrow{\alpha} Q_1' \|_A Q_2} + \text{Symm.}$$

The other rules are as expected

The Refusal Behaviour of PAFAS-terms

$$\textsc{Nil}_r\,\frac{}{\mathsf{nil} \xrightarrow{X}_r \mathsf{nil}} \qquad \textsc{Act}_{r1}\,\frac{}{\alpha.P \xrightarrow{X}_r \underline{\alpha}.P} \qquad \textsc{Act}_{r2}\,\frac{\alpha \notin X \cup \{\tau\}}{\underline{\alpha}.P \xrightarrow{X}_r \underline{\alpha}.P}$$

$$\textsc{Sum}_r\,\frac{Q_1 \xrightarrow{X}_r Q_1',\, Q_2 \xrightarrow{X}_r Q_2'}{Q_1 + Q_2 \xrightarrow{X}_r Q_1' + Q_2'}$$

$$\textsc{Par}_r\,\frac{Q_1 \xrightarrow{X_1}_r Q_1',\, Q_2 \xrightarrow{X_2}_r Q_2',\, X \subseteq (A \cap (X_1 \cup X_2)) \cup ((X_1 \cap X_2)\backslash A)}{Q_1 \|_A Q_2 \xrightarrow{X}_r Q_1' \|_A Q_2'}$$

Notation:

- The *timed transition system* $\mathrm{TTS}(Q)$ of Q consists of all transitions $R \xrightarrow{\mu} R'$ with $\mu \in \mathbb{A}_\tau$ or $\mu = 1$ where R is reachable from Q via such transitions
- $\mathrm{DL}(Q) = \{v \mid Q \overset{v}{\Longrightarrow}\}$ τ's are abstracted away; it contains the *discrete traces* of Q
- The *refusal transition system* $\mathrm{RTS}(Q)$ of Q consists of all transitions $R \xrightarrow{\alpha} R'$ or $R \xrightarrow{X}_r R'$ where R is reachable from Q via such transitions
- $\mathrm{RT}(Q) = \{v \mid Q \overset{\mu}{\Longrightarrow}_r\}$; it contains the *refusal traces* of Q

Performance Measures

Based on PAFAS, we provide two different performance measures

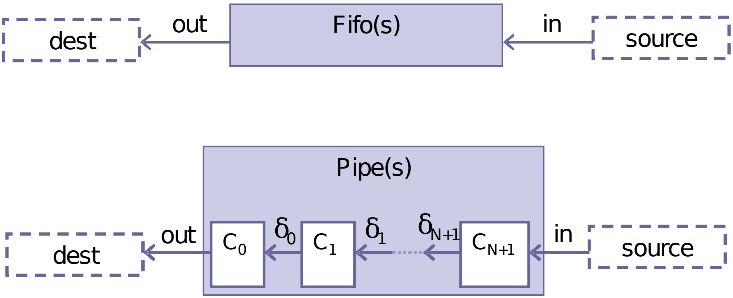
1. A testing-based faster-than (preorder) relation that compares the worst-case effieciency of asynchronous systems (this is a qualitative misure)
2. A performance function that gives for each user behaviour the worst-case time needed to satisfy the user (a quantitative one)

The Testing Preorder

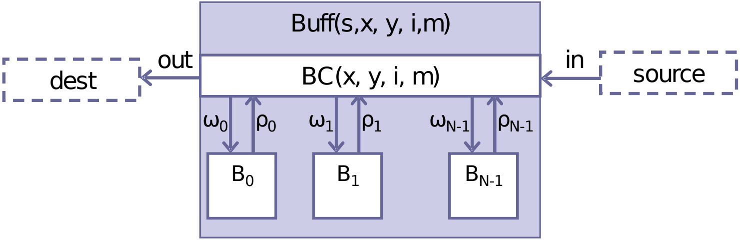
- A timed test is a pair (O, D) where:
  - O is a test process (can perform ω – the success action)
  - $D \in \mathbb{N}_0$ is a time bound
- A testable process Q **satisfies** a test (O, D), i.e. Q must(O, D), if any $v \in \mathrm{DL}(Q \parallel O)$[1] whose duration $\zeta(v) > D$ contains some ω
- Q is **faster than** Q', written $Q \sqsupseteq Q'$, if Q' must(O, D) implies Q must(O, D) for all timed tests (O, D)
- **Theorem** (Characterization of the testing preorder – (1)):
  Let Q, Q' be two testable processes. $Q \sqsupseteq Q'$ iff $\mathrm{RT}(Q) \subseteq \mathrm{RT}(Q)$
- This provides a decidability result for the preorder for finite-state processes

[1] $\parallel$ is a shorthand for $\parallel_{\mathbb{A}-\{\omega\}}$

Three Different Implementations of a Bounded Buffer



Three Different Implementations of a Bounded Buffer



- *Fifo* $\not\sqsupseteq$ *Pipe* and *Pipe* $\not\sqsupseteq$ *Fifo*
- *Fifo* $\sqsupseteq$ *Buff* and *Buff* $\not\sqsupseteq$ *Fifo*
- If $n = 1$ then *Pipe* $\sqsupseteq$ *Buff*, otherwise *Pipe* $\not\sqsupseteq$ *Buff*; *Buff* $\not\sqsupseteq$ *Pipe*

Performance Function

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

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

- The *performance function* p_Q *of* Q is defined by $p_Q(O) = p(Q,O)$
- If $D = p(Q,O)$ then any $v \in \mathrm{DL}(Q \parallel O)$ with $\zeta(v) > D$ contains some ω; 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): *$Q \sqsupseteq Q'$ 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

$$\mathrm{Seq} = \mathrm{rec}\, x.\underline{in}.\tau.out.x$$
$$\mathrm{Pipe} = (\mathrm{rec}\, x.\ \underline{in}.s.x \parallel_{\{s\}} \mathrm{rec}\, x.\ \underline{s}.out.x)/s$$

- One would expect that Pipe is faster than Seq since it allows more parallelism; but it turn out that **this is not true**
- This is because Pipe is not a functional refinement of Seq: the former can perform the sequence *in in* while the latter cannot
- The expectation that Pipe is faster than 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 by defined by

$$U_1 = \underline{in}.\underline{out}.\underline{\omega}$$
$$U_{n+1} = U_n \parallel_\omega \underline{in}.\underline{out}.\underline{\omega}$$

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

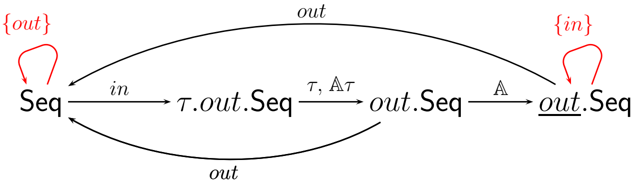
$$rp_Q : \mathbb{N} \longrightarrow \mathbb{N}_0 \cup \{\infty\}$$
$$rp_Q(n) = p_Q(U_u) = p(Q, U_n)$$

- In (2) it is shown how to determine the response performance for the so-calles **response processes**, i.e processes that cannot produce more responses (i.e. *out*) than requests (i.e. *in*)

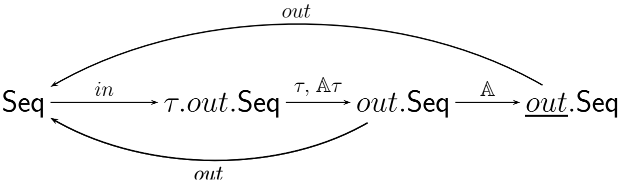
The Reduced Refusal Transition System

- 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 $Q \xrightarrow{X}_r Q'$ 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 partecipate in full time step when considering the behaviour of $Q \parallel U_n$

RTS(Seq)



rRTS(Seq)



Bad-cycle Theorem

Let Q a response process

- A cycle in 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 rRTS(Q)
- **Theorem** (Bad cycles theorem – (2)): *Q has a catastrophic cycle iff its response performance is ∞. 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*

rRTS(Seq)

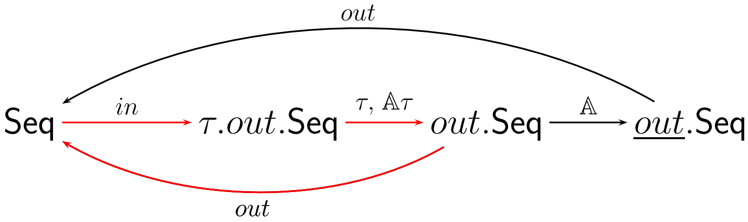


Figure: a cycle with average performance 1

rRTS(Seq)

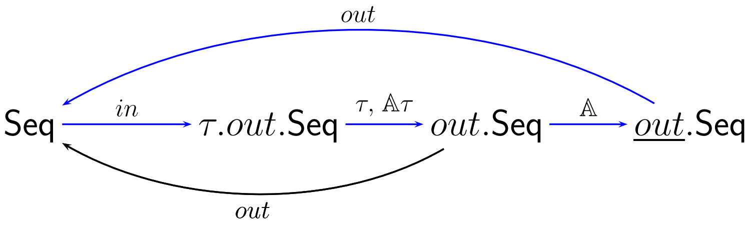


Figure: a cycle with average performance 2 – $rp_{\mathsf{Seq}}(n) = 2n$

rRTS(Pipe)

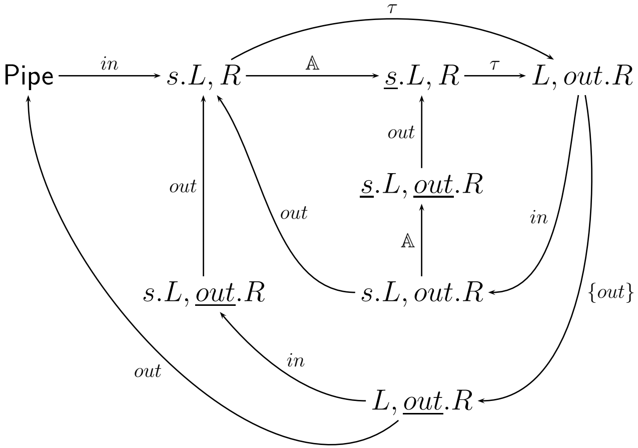


Figure: two bad cycles with average performance $1 - rp_{\text{Pipe}}(n) = n + 1$

Theorem – see (2): *Let Q a finite-state response process Q and n the number os 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

G. Costa, C. Stirling. *Weak and Strong Fairness in CCS*. Inform. and Computation **73**, *pp. 207-244, 1987.*

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 perfomed 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

  $$a.b.\text{nil} \parallel_{\{b\}} b.\text{nil}$$

  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

$$
\begin{aligned}
&\mathsf{L}_u(\mathsf{nil}) = \mathsf{nil}_u, \qquad \mathsf{L}_u(x) = x_u \\
&\mathsf{L}_u(\mu.P) = \{\mu_u.P' \mid P' \in \mathsf{L}_{u1}(P)\} \\
&\mathsf{L}_u(Q_1 + Q_2) = \{Q_1' +_u Q_2' \mid Q_1' \in \mathsf{L}_{u1}(Q_1) \text{ and } Q_2' \in \mathsf{L}_{u2}(Q_2)\} \\
&\mathsf{L}_u(\mathsf{rec}\, x.Q) = \{\mathsf{rec}\, x_u.Q' \mid Q' \in \mathsf{L}_{u1}(Q)\} \ \ldots
\end{aligned}
$$

Example: $\mathsf{L}_\epsilon((a.\mathsf{nil} + b.\mathsf{nil}) + c.\mathsf{nil}) = (a_{11}.\mathsf{nil}_{111} \parallel^{1} b_{12}.\mathsf{nil}_{121}) +_\epsilon c_2.\mathsf{nil}_{21}$

Changes in the Operational Semantics

$$\textsc{Act}_1 \; \frac{}{\alpha_u.P \xrightarrow{\alpha} P} \qquad \textsc{Act}_2 \; \frac{}{\underline{\alpha}_u.P \xrightarrow{\alpha} P}$$

$$\textsc{Rec} \; \frac{Q\{\!|\mathrm{rec}\, x_u.Q/x|\!\} \xrightarrow{\alpha} Q'}{\mathrm{rec}\, x_u.Q \xrightarrow{\alpha} Q'}$$

In $Q\{\!|R/x|\!\}$, each substituted R inherits the label of the x it replaces.
Ex: if $R = \mathrm{rec}\, x_u.a_{u1}.x_{u11}$ then

$$(a_{u1}.x_{u11})\{\!|R/x|\!\} = a_{u1}.\mathrm{rec}\, x_{u11}.a_{u111}.x_{u1111} \xrightarrow{a} \mathrm{rec}\, x_{u11}.a_{u111}.x_{u1111}$$

Thus, labeling is **dynamic**

Live Events

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

$\mathrm{LE}(a_{u1}.\mathrm{nil}) = \{\langle u1 \rangle\}$
a live a-event identified by $\langle u1 \rangle$

$\mathrm{LE}(a_{u21}.\mathrm{nil} +_{u2} b_{u22}.\mathrm{nil}) = \{\langle u21 \rangle, \langle u22 \rangle\}$
two live events (an a-event and a b-event) identified by $\langle u21 \rangle$ and $\langle u22 \rangle$, resp.

$\mathrm{LE}(a_{u1}.\mathrm{nil} \parallel_{\{a\}} (a_{u21}.\mathrm{nil} +_{u2} b_{u22}.\mathrm{nil})) = \{\langle u22 \rangle, \langle u1, u21 \rangle\}$
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

Fair Executions Sequences

Let $P \in \mathrm{L}(\tilde{\mathbb{P}}_1)$ 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 $\gamma_i \in \mathbb{A}_\tau$, for each $i \geq 0$

(*ii*) a timed execution sequence if $\gamma_i \in (\mathbb{A}_\tau \cup \{1\})$, 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

$$\neg(\exists \text{ a tuple } s, \exists\, i \,.\, \forall\, k \geq i \;:\; s \in \mathrm{LE}(Q_k))$$

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
$$\mathsf{LE}(Q_0) \cap \mathsf{LE}(Q_1) \cap \ldots \cap \mathsf{LE}(Q_n) = \emptyset$$
In such a case, we write $Q_0 \xrightarrow{v}_{\mathsf{LE}(Q_0)} Q_n$ 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 $Q_0 \xrightarrow{v_0}_{\mathsf{LE}(Q_0)} Q_1 \xrightarrow{v_1}_{\mathsf{LE}(Q_1)} Q_2 \xrightarrow{v_2}_{\mathsf{LE}(Q_2)} \cdots$
- **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 labes 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

$$R \xrightarrow{a} \text{rec}\, x_{u11}.a_{u111}.x_{u1111} \xrightarrow{a} \text{rec}\, x_{u1111}.a_{u11111}.x_{u111111} \xrightarrow{a} \dots$$

- Thus, cycles in the transition system of a labeled process are not possible as even finite state processes (as $\text{rec}\, x.a.x$) usully 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 $P = \mathrm{rec}\, x.\ a.x \parallel_\emptyset a$ (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 $P \xrightarrow{1} Q = \underline{a}.\mathrm{rec}\, x\ a.x \parallel_\emptyset \underline{a}$

$$Q \xrightarrow{a} \mathrm{rec}\, x.\ a.x \parallel_\emptyset \underline{a} = Q' \xrightarrow{a} \ldots \xrightarrow{a} Q' \xrightarrow{a} \mathrm{rec}\, x.a.x \parallel_\emptyset \mathrm{nil}$$

- Notice that $Q' \stackrel{1}{\not\rightarrow}$ while $\mathrm{rec}\, x.a.x \parallel_\emptyset \mathrm{nil} = P' \xrightarrow{1}$
- Thus, each LE-step of P corresponds to a sequence of timed steps of the form $P \xrightarrow{1} Q \xrightarrow{v} P' \xrightarrow{1}$

LE-steps and 1-1 Transitions – (4)

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

1. If $P_0 \xrightarrow{1} Q_0 \xrightarrow{v} P_1 \xrightarrow{1}$ then $P_0 \xrightarrow{1v}_{\mathsf{LE}(P_0)} P_1$
2. If $P_0 \xrightarrow{v} P_1 \xrightarrow{1} Q_1 \xrightarrow{w} P_2 \xrightarrow{1}$ then $P_0 \xrightarrow{v1w}_{\mathsf{LE}(P_0)} P_2$
3. $P_0 \xrightarrow{v}_{\mathsf{LE}(P_0)} P_1$ 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

- $P_0 = a_1.\mathrm{nil} \parallel_a^\epsilon (a_{21}.\mathrm{nil} +_2 c_{22}.a_{221}.\mathrm{nil})$
- $P_0 \xrightarrow{c}_{\mathrm{LE}(P)} a_1.\mathrm{nil} \parallel_a^\epsilon a_{221}.\mathrm{nil} = P_1$
- $P \xrightarrow{1} \underline{a}_1.\mathrm{nil} \parallel_a^\epsilon (\underline{a}_{21}.\mathrm{nil} +_2 \underline{c}_{22}.\mathrm{nil}) \xrightarrow{c} \underline{a}_1.\mathrm{nil} \parallel_a^\epsilon a_{221}.\mathrm{nil} = Q$
- 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

$$\textsc{Synch}\;\frac{\alpha \in A,\; Q_1 \xrightarrow{\alpha} Q_1',\; Q_2 \xrightarrow{\alpha} Q_2'}{Q_1 \|_A Q_2 \xrightarrow{\alpha} \mathrm{clean}(Q_1' \|_A Q_2')} \qquad \textsc{Par}\;\frac{\alpha \notin A,\; Q_1 \xrightarrow{\alpha} Q_1'}{Q_1 \|_A Q_2 \xrightarrow{\alpha} \mathrm{clean}(Q_1' \|_A Q_2)}$$

$$\textsc{Par}_r\;\frac{Q_1 \xrightarrow{X_1}_r Q_1',\, Q_2 \xrightarrow{X_2}_r Q_2',\, X \subseteq (A \cap (X_1 \cup X_2)) \cup ((X_1 \cap X_2) \backslash A)}{Q_1 \|_A Q_2 \xrightarrow{X}_r \mathrm{clean}(Q_1' \|_A Q_2')}$$

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)

$$\text{clean}(\text{nil}, A) = \text{nil}, \qquad \text{clean}(x, A) = x$$

$$\text{clean}(\alpha.P, A) = \alpha.P \qquad \text{clean}(\underline{\alpha}.P, A) = \begin{cases} \alpha.P & \text{if } \alpha \in A \\ \underline{\alpha}.P & \text{otherwise} \end{cases}$$

$$\text{clean}(Q_1 + Q_2, A) = \text{clean}(Q_1, A) + \text{clean}(Q_2, A)$$

$$\text{clean}(Q_1 \parallel_B Q_2, A) = \text{clean}(Q_1, (B \backslash \mathcal{U}(Q_2)) \cup A) \parallel_B \text{clean}(Q_2, (B \backslash \mathcal{U}(Q_1)) \cup A)$$

$$\text{clean}(Q[\Phi], A) = \text{clean}(Q, \Phi^{-1}(A))[\Phi]$$

$$\text{clean}(\text{rec}\, x.Q, A) = \text{rec}\, x.\text{clean}(Q, A)$$

Unfolding of terms

- $P_0 = \text{rec } x_1.\ a_{11}.x_{111} \parallel_a^{\epsilon} (a_{21}.\text{nil} +_2 c_{22}.a_{221}.\text{nil})$
- $P_0 \xrightarrow{c}_{\text{LE}(P)} \text{rec } x_1.\ a_{11}.x_{111} \parallel_a^{\epsilon} a_{221}.\text{nil} = P_1$
- If $u = 111$ then:

$$\begin{array}{lll} P & \xrightarrow{1} & \underline{a}_{11}.(\text{rec } x_u.\ a_{u1}.x_{u11}) \parallel_a^{\epsilon} (\underline{a}_{21}.\text{nil} +_2 \underline{c}_{22}.a_{221}.\text{nil}) \\ & \xrightarrow{c} & \underline{a}_{11}.(\text{rec } x_u.\ a_{u1}.x_{u11}) \parallel_a^{\epsilon} a_{221}.\text{nil} = Q \end{array}$$

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

$$\text{R}\scriptstyle{\text{EC}}_r\displaystyle\frac{Q \xrightarrow{X}_r Q'}{\text{rec } x_u.Q \xrightarrow{X}_r \text{rec } x_u.Q'} \qquad \text{R}\scriptstyle{\text{EC}}\displaystyle\frac{Q\{\!|\text{rec } x_u.\ \text{unmark}(Q)/x|\!\} \xrightarrow{\alpha} Q'}{\text{rec } x_u.Q \xrightarrow{\alpha}_r Q'}$$

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:

- $P_0 = \text{rec } x_1.\ a_{11}.x_{111} \parallel_a^{\epsilon} (a_{21}.\text{nil} +_2 c_{22}.a_{221}.\text{nil})$
- $P_0 \xrightarrow{c}_{\text{LE}(P)} \text{rec } x_1.\ a_{11}.x_{111} \parallel_a^{\epsilon} .a_{221}.\text{nil} = P_1$
- $$\begin{array}{lcl} P & \xrightarrow{1} & Q = \text{rec } x_1.\ \underline{a}_{11}.x_{111} \parallel_a^{\epsilon} (\underline{a}_{21}.\text{nil} +_2 \underline{c}_{22}.a_{221}.\text{nil}) \\ & \xrightarrow{c} & \text{rec } x_1.\ a_{11}.x_{111} \parallel_a^{\epsilon} a_{221}.\text{nil} = P_1 \end{array}$$
- Moreover:

$$\begin{array}{rcl} (\underline{a}_{11}.x_{111})\{\!|\text{rec } x_1.\ \text{unmark}(\underline{a}_{11}.x_{111})/x|\!\} & = & \\ (\underline{a}_{11}.x_{111})\{\!|\text{rec } x_1.\ a_{11}.x_{111}/x|\!\} & = & \\ \underline{a}_{11}.\text{rec } x_u.\ a_{u1}.x_{u11} & \xrightarrow{a} & \text{rec } x_u.\ a_{u1}.x_{u11} \end{array}$$

(where, again, $u = 111$) and hence, $Q \xrightarrow{a} \text{rec } x_u.\ a_{u1}.x_{u11} \parallel_a^{\epsilon} \text{nil}$

Fairness of everlasting timed execution sequences

Each everlasting timed execution sequence of the form:

$$Q_0 \xrightarrow{v_0} R_1 \xrightarrow{1} Q_1 \xrightarrow{v_1} R_2 \xrightarrow{1} Q_2 \xrightarrow{v_2} R_3 \xrightarrow{1} \ldots$$

where $v_0, v_1, v_2, \ldots \in \mathbb{A}_\tau^*$ is **fair** (because it is associated with a timed fair-step sequence)

Characterization of Fair Executions – The Infinite Case

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

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

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

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

there is a timed execution sequence

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

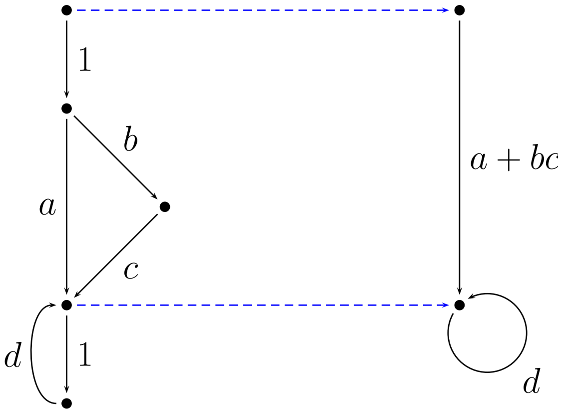
and vice versa

A Transition System for Fair Execution Sequences

- Let P be finite state process (according to the standard operational semantics) and consider the transition system $\mathrm{TTS}(P)$ (all processes reachable from P via $\xrightarrow{\alpha}$ and $\xrightarrow{1}$)
- The **fair timed transition system** of P, written $\mathrm{FairTTS}(P)$, is obtained as follows:

1. the states of $\mathrm{FairTTS}(P)$ are those states Q in $\mathrm{TTS}(P)$ 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 build as described below
   - take $\mathrm{TTS}(P)$ 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



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 $P = \mathrm{rec}\, x.a.x$ can be obtained by considering the non-Zeno run of the form:

$$P \xrightarrow{1} \mathrm{rec}\, x.\underline{a}.x \xrightarrow{a} P \xrightarrow{1}\xrightarrow{a} P \dots$$

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 dene 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

PAFAS^c

Initial process terms are (also in this case) generated by

$$P ::= \mathsf{nil} \mid \alpha.P \mid P + P \mid P \parallel_A P \mid P[\Phi] \mid \mathsf{rec}\ \ x.P$$

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 $\underline{\alpha}.P$ and $P_1 \mathbin{\underline{+}} P_2$
  urgent component has to act in zero time or get disabled

Some Differencies

- 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

$$a \parallel_{\{a\}} (a + c.a) \overset{1}{\longmapsto} \underline{a} \parallel_{\{a\}} (a \underline{+} c.a) \overset{c}{\longmapsto} \underline{a} \parallel_{\{a\}} a \overset{1}{\not\longmapsto}$$

$$\begin{array}{rcl} (a + b) \parallel_{\{a,b\}} (a + c.(b + d)) & \overset{1}{\longmapsto} & (a \underline{+} b) \parallel_{\{a,b\}} (a \underline{+} c.(b + d)) \\ & \overset{c}{\longmapsto} & (a \underline{+} b) \parallel_{\{a,b\}} (b + d) \\ & \overset{d}{\longmapsto} & (a + b) \parallel_{\{a,b\}} \mathsf{nil} \end{array}$$

The Functional Behaviour of PAFAS^c-terms

$$\textsc{Act}_1\frac{}{\alpha.P \stackrel{\alpha}{\longmapsto} P} \qquad \textsc{Act}_2\frac{}{\underline{\alpha}.P \stackrel{\alpha}{\longmapsto} P}$$

$$\textsc{Sum}\frac{Q_1 \stackrel{\alpha}{\longmapsto} Q'}{Q_1 + Q_2 \stackrel{\alpha}{\longmapsto} Q'} + \text{Symm.}$$

$$\textsc{Synch}\frac{\alpha \in A,\ Q_1 \stackrel{\alpha}{\longmapsto} Q_1',\ Q_2 \stackrel{\alpha}{\longmapsto} Q_2'}{Q_1 \|_A Q_2 \stackrel{\alpha}{\longmapsto} \text{clean}(Q_1' \|_A Q_2')}$$

$$\textsc{Par}\frac{\alpha \notin A,\ Q_1 \stackrel{\alpha}{\longmapsto} Q_1'}{Q_1 \|_A Q_2 \stackrel{\alpha}{\longmapsto} \text{clean}(Q_1' \|_A Q_2)}$$

$$\textsc{Rec}\frac{Q\{\!|\text{rec}\, x.\ \text{unmark}(Q)/x|\!\} \stackrel{\alpha}{\longmapsto} Q'}{\text{rec}\, x.Q \stackrel{\alpha}{\longmapsto}_r Q'}$$

Cleaning inactive markings

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

$$\text{clean}(\text{nil}, A) = \text{nil} \qquad \text{clean}(x, A) = x$$

$$\text{clean}(\alpha.P, A) = \alpha.P \qquad \text{clean}(\underline{\alpha}.P, A) = \begin{cases} \alpha.P & \text{if } \alpha \in A \\ \underline{\alpha}.P & \text{otherwise} \end{cases}$$

$$\text{clean}(P_1 + P_2, A) = P_1 + P_2$$

$$\text{clean}(P_1 \underline{+} P_2, A) = \begin{cases} P_1 + P_2 & \text{if } \mathcal{A}(P_1) \cup \mathcal{A}(P_2) \subseteq A \\ P_1 \underline{+} P_2 & \text{otherwise} \end{cases}$$

$$\text{clean}(Q_1 \|_B Q_2, A) = \text{clean}(Q_1, (B \backslash \mathcal{A}(Q_2)) \cup A) \|_B \text{clean}(Q_2, (B \backslash \mathcal{A}(Q_1)) \cup A)$$

$$\text{clean}(Q[\Phi], A) = \text{clean}(Q, \Phi^{-1}(A))[\Phi]$$

$$\text{clean}(\text{rec } x.Q, A) = \text{rec } x.\text{clean}(Q, A)$$

The Timed Behaviour of PAFAS^c-terms

- In order to define the timed behaviour of PAFAS^c-terms, we exploit a function urgent(_) that 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, 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^c-terms

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

$$\begin{array}{lcl}
\text{urgent}(\alpha.P) & = & \underline{\alpha}.P \\
\text{urgent}(P_1 + P_2) & = & P_1 \underline{+} P_2 \\
\text{urgent}(a.P_1 \parallel_{\{a\}} a.P_2) & = & \underline{a}.P_1 \parallel_{\{a\}} \underline{a}.P_2 \\
\text{urgent}(a.P_1 \parallel_{\{a\}} b.P_2) & = & a.P_1 \parallel_{\{a\}} \underline{b}.P_2 \\
\text{urgent}((a.P_1 + c.\text{nil}) \parallel_{\{a\}} b.P_2) & = & (a.P_1 \underline{+} c.\text{nil}) \parallel_{\{a\}} \underline{b}.P_2 \\
\text{urgent}((a.P_1 + c.\text{nil}) \parallel_{\{a,c\}} b.P_2) & = & (a.P_1 + c.\text{nil}) \parallel_{\{a,c\}} \underline{b}.P_2
\end{array}$$

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

$$\mathsf{L}_u(\mathrm{nil}) = \mathrm{nil}_u, \qquad \mathsf{L}_u(x) = x_u$$
$$\mathsf{L}_u(\mu.P) = \{\mu_u.P' \mid P' \in \mathsf{L}_{u1}(P)\}$$

$$\mathsf{L}_u(P_1 + P_2) = \{P_1' +_u P_2' \mid P_1' \in \mathsf{L}_{u1}(P_1) \text{ and } P_2' \in \mathsf{L}_{u2}(P_2)\}$$
$$\mathsf{L}_u(P_1 \underline{+} P_2) = \{P_1' \underline{+}_u P_2' \mid P_1' \in \mathsf{L}_{u1}(P_1) \text{ and } P_2' \in \mathsf{L}_{u2}(P_2)\}$$

$$\mathsf{L}_u(\mathrm{rec}\, x.Q) = \{\mathrm{rec}\, x_u.Q' \mid Q' \in \mathsf{L}_{u1}(Q)\}\ldots$$

- **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:

$P = b_{u1}.a_{u11}.\mathrm{nil} +_{u} a_{u2}$ $\quad$ $\mathrm{LE}(P) = \{\langle u1\rangle, \langle u2\rangle\}$
$\mathrm{LC}(P) = \{u\}$

$P = b_{u1}.a_{u11}.\mathrm{nil} \parallel^{u}_{\{a\}} a_{u2}$ $\quad$ $\mathrm{LE}(P) = \{\langle u1\rangle\}$
$\mathrm{LC}(P) = \{u1\}$

$P = a_{u11}.\mathrm{nil} \parallel^{u}_{\{a\}} a_{u2}$ $\quad$ $\mathrm{LE}(P) = \{\langle u11, u2\rangle\}$
$\mathrm{LC}(P) = \{u11, u2\}$

Fair Executions Sequences

- A (timed) execution sequence $Q_0 \overset{\gamma_0}{\longmapsto} Q_1 \overset{\gamma_1}{\longmapsto} \ldots$ is **fair** if

$$\neg(\exists\, s\, \exists\, i\,.\, \forall\, k \geq i \;:\, s \in \mathrm{LC}(Q_k))$$

- The sequence of transitions $Q_0 \overset{\gamma_0}{\longmapsto} Q_1 \overset{\gamma_1}{\longmapsto} \ldots \overset{\gamma_{n-1}}{\longmapsto} Q_n$ is a **(timed)** LC-**step** if

$$\mathrm{LC}(Q_0) \cap \mathrm{LC}(Q_1) \cap \ldots \cap \mathrm{LC}(Q_n) = \emptyset$$

- **(Timed) fair-step sequences** are maximal sequences of the form $Q_0 \overset{v_0}{\longmapsto}_{\mathrm{LC}(Q_0)} Q_1 \overset{v_1}{\longmapsto}_{\mathrm{LC}(Q_1)} Q_2 \overset{v_2}{\longmapsto}_{\mathrm{LC}(Q_2)} \cdots$
- **Theorem** (Costa & Stirling):

  An execution is **fair** *if and only if it is the sequence associated with a fair-step sequence*

LC-steps and 1-1 Transitions – (5)

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

1. If $P_0 \stackrel{1}{\longmapsto} Q_0 \stackrel{v}{\longmapsto} P_1 \stackrel{1}{\longmapsto}$ then $P_0 \stackrel{1v}{\longmapsto}_{\mathrm{LC}(P_0)} P_1$
2. If $P_0 \stackrel{v}{\longmapsto} P_1 \stackrel{1}{\longmapsto} Q_1 \stackrel{w}{\longmapsto} P_2 \stackrel{1}{\longmapsto}$ then $P_0 \stackrel{v1w}{\longmapsto}_{\mathrm{LC}(P_0)} P_2$
3. $P_0 \stackrel{v}{\longmapsto}_{\mathrm{LC}(P_0)} P_1$ implies $P_0 \stackrel{1}{\longmapsto} Q_0 \stackrel{v}{\longmapsto} P_1 \stackrel{1}{\longmapsto}$

Part IV

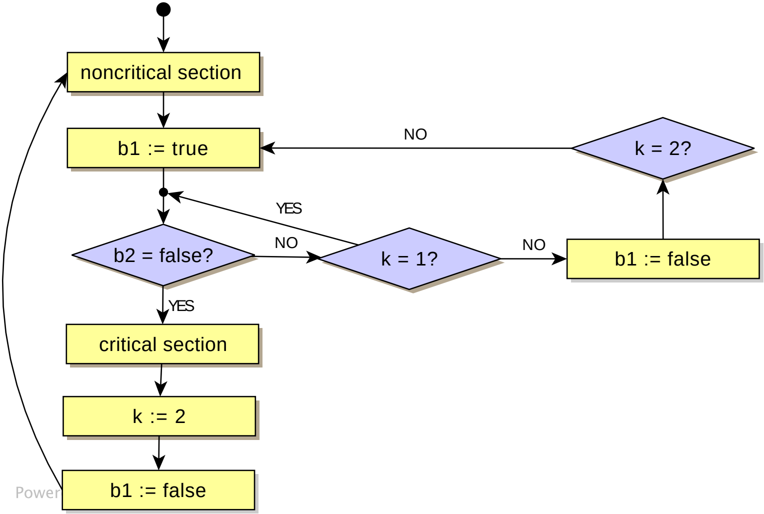
Liveness Property of a MUTEX Algorithm

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
          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



Translating Dekker's algorithm into PAFAS processes

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

$$\begin{array}{lcl} \mathrm{B}_1(\mathit{false}) & = & b_1\mathit{rf}.\mathrm{B}_1(\mathit{false}) + (b_1\mathit{wf}.\mathrm{B}_1(\mathit{false}) + b_1\mathit{wt}.\mathrm{B}_1(\mathit{true})) \\ \mathrm{B}_1(\mathit{true}) & = & b_1\mathit{rt}.\mathrm{B}_1(\mathit{true}) + (b_1\mathit{wf}.\mathrm{B}_1(\mathit{false}) + b_1\mathit{wt}.\mathrm{B}_1(\mathit{true})) \\ \mathrm{K}(1) & = & \mathit{kr1}.\mathrm{K}(1) + (\mathit{kw1}.\mathrm{K}(1) + \mathit{kw2}.\mathrm{K}(2)) \\ \mathrm{K}(2) & = & \mathit{kr2}.\mathrm{K}(1) + (\mathit{kw1}.\mathrm{K}(1) + \mathit{kw2}.\mathrm{K}(2)) \end{array}$$

- Given $b_1, b_2 \in \{\mathit{true}, \mathit{false}\}$ and $k \in \{1, 2\}$, we define

$$\mathrm{PV}(b_1, b_2, k) = (\mathrm{B}_1(b_1) \parallel_\emptyset \mathrm{B}_2(b_2)) \parallel_\emptyset \mathrm{K}(k))$$

Translating Dekker's algorithm into PAFAS processes

- The process $\mathtt{P}_1$ is represented by (the process $\mathtt{P}_2$ has a symmetric representation):

$$
\begin{aligned}
P_1 &= \mathtt{req}_1.b_1wt.P_{11} + \tau.P_1 \\
P_{11} &= b_2rf.P_{14} + b_2rt.P_{12} \\
P_{12} &= kr1.P_{11} + kr2.b_1wf.P_{13} \\
P_{13} &= kr1.b_1wt.P_{11} + kr2.P_{13} \\
P_{14} &= \mathtt{cs}_1.kw2.b_1wf.P_1
\end{aligned}
$$

- The algorithm can be defined as

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

where B contains all reading and writing actions and the relabeling function Φ_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 $\mathtt{P}_i$ requests the execution of its critical section then, in any continuation of that computation, there is a point at which $\mathtt{P}_i$ 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 $\mathtt{req}_i$ in a fair trace is eventually followed by cs_i, $i = 1, 2$.

D. J. Walker. *Automated Analysis of Mutual Exclusion algorithms using CCS.* Formal Aspects of Comp. **1**, *pp. 273-292, 1989.*

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 $\texttt{req}_i$ is **never** followed by the corresponding $\texttt{cs}_i$
- 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

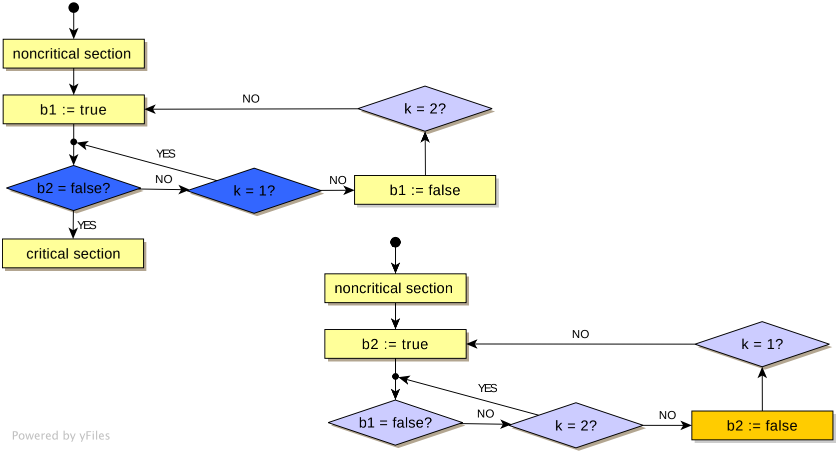
An example

- 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

$$\begin{array}{lll} P & \xrightarrow{1} & Q = (\underline{r}.R \parallel_{\emptyset} \underline{w}.W) \parallel_{\{r,w\}} (\underline{r}.V + \underline{w}.V) \\ & \xrightarrow{r} & (R \parallel_{\emptyset} w.W) \parallel_{\{r,w\}} V = P \end{array}$$

  Each time an r is performed, V offers a new (not urgent) synchronization pattern to $\underline{w}.W$, 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^c timed operational semantics

$$P \xmapsto{1} Q = (\underline{r}.R \parallel_\emptyset \underline{w}.W) \parallel_{\{r,w\}} (r.V \underline{+} w.V)$$

$$Q \xmapsto{r} Q' = (R \parallel_\emptyset \underline{w}.W) \parallel_{\{r,w\}} V \not\xmapsto{1}$$
$$\xmapsto{w} P$$

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