A further application of FASE: study liveness properties on Mutual Exclusion Algorithms

speaker: Massimo Callisto De Donato
{massimo.callisto}@unicam.it

Scuola di Scienze e Tecnologie - Università di Camerino
Via Madonna delle Carceri, 9 - 62032 Camerino
http://www.cs.unicam.it

PaCo Meeting - Camerino
September, 15th 2010
FASE (Faster Asynchronous Systems Evaluation):

- Aimed to analyse systems specified in PAFAS algebra
- Different modules loosely-coupled written in Java
- Evaluates the efficiency of a process in the worst-case behaviour
- Checks the presence of catastrophic cycles

Evaluating the Efficiency of Asynchronous Systems with FASE.
In pre-proc. of the 1st Int. Workshop on Quantitative Formal Methods, pp.101-106, Technische Universiteit Eindhoven, 2009
FASE (Faster Asynchronous Systems Evaluation):

- Aimed to analyse systems specified in PAFAS algebra
- Different modules loosely-coupled written in Java
- Evaluates the efficiency of a process in the worst-case behaviour
- Checks the presence of catastrophic cycles

Apply FASE to study new properties about systems:

Evaluating the Efficiency of Asynchronous Systems with FASE.
In pre-proc. of the 1st Int. Workshop on Quantitative Formal Methods, pp.101-106, Technische Universiteit Eindhoven, 2009
FASE (Faster Asynchronous Systems Evaluation):
- Aimed to analyse systems specified in PAFAS algebra
- Different modules loosely-coupled written in Java
- Evaluates the efficiency of a process in the worst-case behaviour
- Checks the presence of catastrophic cycles

Apply FASE to study new properties about systems:
- Checks liveness properties
- Under the assumption of fairness of actions
- And a process algebra with Non-Blocking Reading
- By studying the presence of catastrophic cycles

Evaluating the Efficiency of Asynchronous Systems with FASE.
In pre-proc. of the 1st Int. Workshop on Quantitative Formal Methods, pp.101-106, Technische Universiteit Eindhoven, 2009

F. Corradini, M.R. Di Berardini, W. Vogler
Time and Fairness in a Process Algebra with Non-Blocking Reading
Liveness

*Something good will happen eventually.*
Consider a critical resource that must be accessed from concurrent processes in a mutual way. An algorithm is used to regulate the usage of the resource. Different properties can be studied:

- Is the mutual exclusion preserved?
- Is the algorithm **live**?
Liveness

Something good will happen eventually.

Consider a critical resource that must be accessed from concurrent processes in a mutual way. An algorithm is used to regulate the usage of the resource. Different properties can be studied:

- Is the mutual exclusion preserved?
- Is the algorithm live?

Liveness

Whenever, at some point, a process $P_i$ requests the execution of its critical section, then at some later point it will enter it.

The verification of liveness properties usually requires some fairness assumption.

D.J. Walker

Automated Analysis of Mutual Exclusion algorithms using CCS

A characterization of fair sequences

(Weak) fairness of actions: each action continuously enabled along a computation must eventually proceed.

Theorem (fair traces)

An infinite trace $\alpha_0\alpha_1\alpha_2\ldots$ is fair iff there exists a non-Zeno timed execution sequence

$$P_0 \xrightarrow{v_0} P_1 \xrightarrow{v_1} \ldots P_n \xrightarrow{v_n} P_{n+1}\ldots$$

where $v_0v_1\ldots v_m\ldots = \alpha_0\alpha_1\ldots \alpha_i\ldots$

---

F. Corradini, M.R. Di Berardini, W. Vogler
Fairness of Actions in System Computations

F. Corradini, M.R. Di Berardini, W. Vogler
Time and Fairness in a Process Algebra with Non-Blocking Reading

. Costa, C. Stirling
Weak and Strong Fairness in CCS
Information and Computation 73, pp. 207-244, 1987.
Liveness in mutual exclusion algorithms

If, whenever at any computation a process $P_i$ requests the execution of its critical section (e.g. $req_i$ action), then in any continuation of that computation $P_i$ will perform its critical section (e.g. $cs_i$ action).
If, whenever at any computation a process $P_i$ requests the execution of its critical section (e.g. $\text{req}_i$ action), then in any continuation of that computation $P_i$ will perform its critical section (e.g. $\text{cs}_i$ action).

To establish that an algorithm preserves its liveness property, we check if any occurrence of $\text{req}_i$ in a fair trace is eventually followed by $\text{cs}_i$.

Namely, we check if the process is free from catastrophic cycles.

- F. Corradini and W. Vogler
  Measuring the performance of asynchronous systems with PAFAS

- F. Corradini, M.R. Di Berardini, W. Vogler
  Time and Fairness in a Process Algebra with Non-Blocking Reading
Catastrophic cycles

A cycle in the rRTS(P) is called catastrophic if it contains a positive number of time steps but no in’s and no out’s.
Catastrophic cycles and liveness

In a fair trace, after the request $req_i$ there exists an infinite sequence of actions and time steps without any $cs_i$ action.
Dekker's Algorithm

\[
\begin{align*}
  & \text{non-critical section} \\
  & b_1 := \text{false} \\
  & k := 2 \\
  & \text{critical section} \\
  & b_1 := \text{true} \\
  & b_2 = \text{false?} \\
  & \text{no} \\
  & k = 1? \\
  & \text{no} \\
  & b_1 := \text{false} \\
  & \text{yes} \\
  & k = 1? \\
  & \text{yes} \\
  & b_1 := \text{false} \\
  & \text{no} \\
  & k = 1? \\
  & \text{no} \\
  & b_1 := \text{false} \\
  & \text{yes} \\
  & k = 1? \\
  & \text{yes} \\
  & b_1 := \text{false} \\
  & \text{no} \\
  & k = 1? \\
  & \text{yes} \\
  & b_1 := \text{false} \\
  & \text{no} \\
  & b_1 := \text{false} \\
\end{align*}
\]
Dekker algorithm translated in PAFAS

\[B1T = b1rt.B1T + b1wt.B1T + b1wf.B1F;\]
\[PV = (B1F \mid \| \mid B2F \mid \| \mid K1);\]

\[P1 = \text{in}.b1wt.P11;\]
\[P11 = b2rf.P14 + b2rt.P12;\]
\[P12 = kr1.P11 + kr2.b1wf.P13;\]
\[P13 = kr1.b1wt.P11 + kr2.P13;\]
\[P14 = \text{out}.kw2.b1wf.P1;\]

\[P2 = \text{req}2.b2wt.P21;\]
\[P21 = b1rf.P24 + b1rt.P22;\]
\[P22 = kr2.P21 + kr1.b2wf.P23;\]
\[P23 = kr2.b2wt.P21 + kr1.P23;\]
\[P24 = \text{exit}2.kw1.b2wf.P2;\]

\[\text{DEKKER} = ((P1 \mid \| \mid P2) \mid \| [B] \mid PV) [L \rightarrow \tau];\]

\[B = \text{sort}(PV),\ L = \text{sort}(\text{DEKKER}) \setminus \{\text{in, out}\}\]
Result on Dekker

Dekker is not live meaning that catastrophic cycles are detected.
Dekker is not live meaning that catastrophic cycles are detected.

Some unwanted behaviour: a process reading/writing a variable can indefinitely block another process that tries to read/write it.
Liveness in Dekker - some results

Result on Dekker

Dekker is not live meaning that catastrophic cycles are detected.

- Some unwanted behaviour: a process reading/writing a variable can indefinitely block another process that tries to read/write it.
- Dekker is not live under the assumption of fairness of actions.
- Suffers of Starvation and livelock.

F. Corradini, M.R. Di Berardini, W. Vogler
Checking a Mutex Algorithm in a Process Algebra with Fairness
Proc. of CONCUR'06, pp. 142-157, LNCS 4137, 2006
We could consider the scenario where multiple concurrent processes can read the same variable:

\[
B_i(\text{false}) = \{b_i rf\} > b_i wf. B_i(\text{false}) + b_i wt. B_i(\text{true}) \\
B_i(\text{true}) = \{b_i rt\} > b_i wt. B_i(\text{true}) + b_i wf. B_i(\text{false})
\]
We could consider the scenario where multiple concurrent processes can read the same variable:

\[
B_i(\text{false}) = \{b_{irf}\} > b_{iwf}.B_i(\text{false}) + b_{iwt}.B_i(\text{true})
\]
\[
B_i(\text{true}) = \{b_{irt}\} > b_{iwt}.B_i(\text{true}) + b_{iwf}.B_i(\text{false})
\]

- Dekker is still not live.
- Catastrophic cycles are detected.
- Livelock is still present.
The only ordinary actions are those actions that correspond to the writing of a new value.

These actions can be thought of as non-destructive operations, allowing other potential concurrent accesses.

This way of accessing variables is not new, e.g., database systems.

\[
B_i(\text{false}) = \{b_irf, b_iwf\} > b_iwt.B_i(\text{true}) \\
B_i(\text{true}) = \{b_i rt, b_i wt\} > b_i wf.B_i(\text{false})
\]

Dekker is live and no catastrophic cycle is detected.

F. Corradini, M.R. Di Berardini, W. Vogler
Time and Fairness in a Process Algebra with Non-Blocking Reading
Dekker’s algorithm

\[\begin{align*}
B_1F &= \{b_1rf, b_1wf\} > b_1wt. B_1T; \\
B_1T &= \{b_1rt, b_1wt\} > b_1wf. B_1F; \\
B_2F &= \{b_2rf, b_2wf\} > b_2wt. B_2T; \\
B_2T &= \{b_2rt, b_2wt\} > b_2wf. B_2F; \\
K_1 &= \{kr_1, kw_1\} > kw_2. K_2; \\
K_2 &= \{kr_2, kw_2\} > kw_1. K_1; \\
PV &= (B_1F |[]| B_2F |[]| K_1); \\
P_1 &= \text{req1}. b_1wt. P_{11}; \\
P_{11} &= b_2rf. P_{14} + b_2rt. P_{12}; \\
P_{12} &= kr_1. P_{11} + kr_2. b_1wf. P_{13}; \\
P_{13} &= kr_1. b_1wt. P_{11} + kr_2. P_{13}; \\
P_{14} &= \text{exit1}. kw_2. b_1wf. P_1; \\
P_2 &= \text{req2}. b_2wt. P_{21}; \\
P_{21} &= b_1rf. P_{24} + b_1rt. P_{22}; \\
P_{22} &= kr_2. P_{21} + kr_1. b_2wf. P_{23}; \\
P_{23} &= kr_2. b_2wt. P_{21} + kr_1. P_{23}; \\
P_{24} &= \text{exit2}. kw_1. b_2wf. P_2; \\
\text{DEKKER} &= ((P_1 |[]| P_2)|[B]|PV)[L \rightarrow \tau]; \\
B &= \text{sort}(PV), L &= \text{sort}(\text{DEKKER}) \setminus \{\text{in}, \text{out}\}
\end{align*}\]
Dijkstra’s algorithm

Non critical section

\( b_i := \text{false} \)

\( k \neq i \)

\( c_i := \text{true} \)

\( b[k] = i \)

\( k := i \)

\( c_i := \text{false} \)

For \( j := 1 \) to \( n \)

\( j \neq i \) && \( \neg c[j] \)

\( c_i := \text{true} \)

\( b_i := \text{true} \)

\( c_i := \text{true} \)

Critical section
Dijkstra's algorithm

\[
B_{1F} = b_{1rf}.B_{1F} + b_{1wf}.B_{1F} + b_{1wt}.B_{1T}; \quad B_{2F} = b_{2rf}.B_{2F} + b_{2wf}.B_{2F} + b_{2wt}.B_{2T}; \\
B_{1T} = b_{1rt}.B_{1T} + b_{1wt}.B_{1T} + b_{1wf}.B_{1F}; \quad B_{2T} = b_{2rt}.B_{2T} + b_{2wt}.B_{2T} + b_{2wf}.B_{2F}; \\
C_{1F} = c_{1rf}.C_{1F} + c_{1wf}.C_{1F} + c_{1wt}.C_{1T}; \quad C_{2F} = c_{2rf}.C_{2F} + c_{2wf}.C_{2F} + c_{2wt}.C_{2T}; \\
C_{1T} = c_{1rt}.C_{1T} + c_{1wt}.C_{1T} + c_{1wf}.C_{1F}; \quad C_{2T} = c_{2rt}.C_{2T} + c_{2wt}.C_{2T} + c_{2wf}.C_{2F};
\]

\[
K_{1} = k_{r1}.K_{1} + k_{w1}.K_{1} + k_{w2}.K_{2} + \text{get.}(k_{1r1}.\text{put}.K_{1} + k_{2r1}.\text{put}.K_{1}); \\
K_{2} = k_{r2}.K_{2} + k_{w2}.K_{2} + k_{w1}.K_{1} + \text{get.}(k_{1r2}.\text{put}.K_{2} + k_{2r2}.\text{put}.K_{2});
\]

\[
PV = (B_{1T} || B_{2T} || C_{1T} || C_{2T} || K_{1});
\]

\[
P_{1} = b_{1wf}.\text{in}.P_{11}; \quad P_{1} = b_{1wf}.\text{in}.P_{11}; \\
P_{11} = k_{r1}.P_{13} + k_{r2}.c_{1wt}.P_{12}; \quad P_{11} = k_{r2}.P_{23} + k_{r1}.c_{2wt}.P_{22}; \\
P_{12} = \text{get.}(k_{1r1}.(b_{1rt}.\text{put}.k_{w1}.P_{11} + b_{1rf}.\text{put}.P_{11}) + k_{1r2}.(b_{2rt}.\text{put}.k_{w1}.P_{11} + b_{2rf}.\text{put}.P_{11})); \quad P_{21} = k_{r2}.P_{23} + k_{r1}.c_{2wt}.P_{22}; \\
P_{13} = c_{1wf}.(c_{2rf}.P_{11} + c_{2rt}.P_{14}); \quad P_{14} = P_{23} = c_{2wf}.(c_{1rf}.P_{21} + c_{1rt}.P_{24}); \\
P_{14} = \text{out}.c_{1wt}.b_{1wt}.P_{1}; \quad P_{14} = \text{out}.c_{1wt}.b_{1wt}.P_{1};
\]

\[
\text{DIJKSTRA} = ((P_{1} | [] | P_{2}) | [B] | PV)[L \rightarrow \tau ];
\]

\[
B = \text{sort}(PV), \quad L = \text{sort}((\text{DIJKSTRA}) \setminus \{\text{in}, \text{out}\})
\]
Knuth’s algorithm

- $c_i := 1$
  - $c_i := 1$
  - $c_i := 2$
    - $c_j = 2$
      - $k := i$

Non critical section

Critical section

- $k = i$
  - $c_j \neq 0$

F. Buti, M. Callisto, F. Corradini, M.R. Di Berardini

A further application of FASE: study liveness properties on Mutual Exclusion Algorithms
Knuth’s algorithm

\[
C_{10} = c_{1w0}.C_{10} + c_{1w1}.C_{11} + c_{1w2}.C_{12} + c_{1r0}.C_{10};
\]
\[
C_{11} = c_{1w0}.C_{10} + c_{1w1}.C_{11} + c_{1w2}.C_{12} + c_{1r1}.C_{11};
\]
\[
C_{12} = c_{1w0}.C_{10} + c_{1w1}.C_{11} + c_{1w2}.C_{12} + c_{1r2}.C_{12};
\]
\[
K_{1} = k_{r1}.K_{1} + k_{w1}.K_{1} + k_{w2}.K_{2};
\]
\[
K_{2} = k_{r2}.K_{2} + k_{w2}.K_{2} + k_{w1}.K_{1}
\]
\[
PV = (C_{10} || C_{20} || K_{1});
\]
\[
P_{1} = c_{1w1}.in.P_{11};
\]
\[
P_{11} = k_{r1}.P_{13} + k_{r2}.P_{12};
\]
\[
P_{12} = c_{2r0}.P_{13} + c_{2r1}.P_{11} + c_{2r2}.P_{11};
\]
\[
P_{13} = c_{1w2}.P_{14};
\]
\[
P_{14} = c_{2r0}.P_{15} + c_{2r1}.P_{15} + c_{2r2}.P_{16};
\]
\[
P_{15} = kw_{1}.out.kw_{2}.c_{1w0}.P_{1};
\]
\[
P_{16} = c_{1w1}.P_{11};
\]
\[
P_{2} = c_{2w1}.req2.P_{21};
\]
\[
P_{21} = k_{r2}.P_{23} + k_{r1}.P_{22};
\]
\[
P_{22} = c_{1r0}.P_{23} + c_{1r1}.P_{21} + c_{1r2}.P_{21};
\]
\[
P_{23} = c_{2w2}.P_{24};
\]
\[
P_{24} = c_{1r0}.P_{25} + c_{1r1}.P_{25} + c_{1r2}.P_{26};
\]
\[
P_{25} = kw_{2}.cs2.kw_{1}.c_{2w0}.P_{2};
\]
\[
P_{26} = c_{2w1}.P_{21};
\]
\[
KNUTH = ((P_{1} |[]| P_{2}) |[B]| PV)[L -> tau];
\]
\[
B = sort(PV), L = sort(KNUTH)\{\text{in, out}\}
\]
Peterson’s algorithm

\[ \begin{align*}
\text{critical section:} &
\begin{align*}
&b_i := \text{true} \\
&k := j \\
&\text{while } b_j \land k = j \\
&\quad \text{skip} \\
&b_i := \text{false}
\end{align*}
\end{align*} \]
Peterson’s algorithm

\[
B_1F = b_{1rf}.B_1F + b_{1wf}.B_1F + b_{1wt}.B_1T;
\]
\[
B_1T = b_{1rt}.B_1T + b_{1wt}.B_1T + b_{1wf}.B_1F;
\]
\[
K_1 = kr_1.K_1 + kw_1.K_1 + kw_2.K_2;
\]
\[
K_2 = kr_2.K_2 + kw_2.K_2 + kw_1.K_1;
\]
\[
PV = (B_1F \parallel B_2F \parallel K_1);
\]
\[
P_1 = b_{1wt}.in.kw_2.P_{11};
\]
\[
P_{11} = b_{2rf}.P_{12} + b_{2rt}.(kr_2.P_{11} +
kr_1.P_{12});
\]
\[
P_{12} = out.b_{1wf}.P_1;
\]
\[
P_2 = b_{2wt}.req2.kw_1.P_{21};
\]
\[
P_{21} = b_{1rf}.P_{22} + b_{1rt}.(kr_1.P_{21} +
kr_2.P_{22});
\]
\[
P_{22} = cs_2.b_{2wf}.P_2;
\]

\[
\text{PETE}R\text{SON} = ((P_1 \mid \top) \mid P_2) \mid \{B\} \mid PV)[L \rightarrow \tau];
\]

\[
B = \text{sort}(PV), \quad L = \text{sort}(\text{PETE}R\text{SON}) \setminus \{\text{in, out}\}
\]
Lamport’s algorithm

$\forall i \in \mathbb{N}$

1. $b_i := \text{false}$
2. For $j := 1$ to $i - 1$
   - $b_j$ while $b_j$
     - skip
   - $b_i := \text{false}$
3. For $j := i + 1$ to $n$
   - $b_j$ while $b_j$
     - skip
4. $b_i := \text{true}$
5. Non critical section
6. Critical section

Non critical section

Critical section

$b_i := \text{false}$

$b_i := \text{true}$

$b_j$ while $b_j$

F. Buti, M. Callisto, F. Corradini, M.R. Di Berardini

A further application of FASE: study liveness properties on Mutual Exclusion Algorithms
Lamport’s algorithm

\[ \begin{align*}
B1F &= b1rf.B1F + b1wf.B1F + b1wt.B1T; \\
B1T &= b1rt.B1T + b1wf.B1F + b1wt.B1T; \\
PV &= (B1F || B2F); \\
P1 &= b1wt.in.P1; \\
P11 &= b2rf.P12 + b2rt.P11; \\
P12 &= out.b1wf.P1; \\
P2 &= b2wt.req2.P2; \\
P21 &= b1rf.P23 + b1rt.b2wf.P22; \\
P22 &= b1rf.b2wt.P21 + b1rt.P22; \\
P23 &= cs2.b2wf.P2; \\
LAMPORT &= ((P1 || P2) || PV)[L \rightarrow \tau]; \\
B &= sort(PV), \quad L = sort(LAMPORT)\{\text{in, out}\}
\end{align*} \]
Relating results

- All the algorithms have been coded in PAFAS and automatically checked by the tool FASE.
- Interesting results have been found:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Walker</th>
<th>Without read acts</th>
<th>With read acts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dekker</td>
<td>not live</td>
<td>not live</td>
<td>live</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>not live</td>
<td>not live</td>
<td>not live</td>
</tr>
<tr>
<td>Knuth</td>
<td>live</td>
<td>not live</td>
<td>not live</td>
</tr>
<tr>
<td>Peterson</td>
<td>live</td>
<td>not live</td>
<td>live</td>
</tr>
<tr>
<td>Lamport$_{\text{live}p_1 \land \text{live}p_2}$</td>
<td>not live</td>
<td>not live</td>
<td>not live</td>
</tr>
<tr>
<td>Lamport$_{\text{live}p_1 \land \neg \text{live}p_2}$</td>
<td>live</td>
<td>not live</td>
<td>live</td>
</tr>
</tbody>
</table>

**Table:** Comparing Walker result with catastrophic cycles detection
Further observations

Peterson’s algorithm

- Without using read actions, FASE has detected catastrophic cycles: $P_i$ continuously reads the same value from the same variable while $P_2$ is stuck and waits to write it.
- When read acts are used to models the action of read the same value from a variable, Peterson’s algorithm becomes live.
**Peterson’s algorithm**

- Without using read actions, FASE has detected catastrophic cycles: *\( P_i \) continuously reads the same value from the same variable while *\( P_2 \) *is stuck and waits to write it.*
- When read acts are used to model the action of read the same value from a variable, Peterson’s algorithm becomes live.

**Lamport’s algorithm**

- The same considerations about Peterson’s algorithms are valid if we consider to observe *\( P_1 \).*
- If we observe *\( P_2 \)*, then the catastrophic cycles are detected in both cases as expected (due to the asymmetry of the processes).
Dijkstra’s algorithm

- In Dijkstra’s algorithm, FASE has detected catastrophic cycles in both cases.
- It is known that this algorithm is susceptible to starvation.
- But there are also catastrophic cycles as following:
Further observations

Dijkstra’s algorithm

- In Dijkstra’s algorithm, FASE has detected catastrophic cycles in both cases.
- It is known that this algorithm is susceptible to starvation.
- But there are also catastrophic cycles as following:

\[ K_2 = \{ kr_2, kw_2 \} > (kw_1.K_1 + \text{get}.(k_1r_2.put.K_2 + k_2r_2.put.K_2)) \];

\[ P_{12} = \text{get}.(\ldots); \]

\[ P_{21} = kr_2.P_{23} + kr_1.c_2w.t.P_{22}; \]

\[ P = P_{12} \parallel P_{22} \parallel B \ldots K_2 \xrightarrow{\tau(\text{get})} \tau \ldots 1 \rightarrow P \]
Further observations

Knuth’s algorithm

- FASE has detected catastrophic cycles in both cases.
- When read acts are used, the follow catastrophic cycles is detected:

\[
C_{12} = \{c_{1r2}, c_{1w2}\} > (c_{1w0}.C_{10} + c_{1w1}.C_{11})
\]

\[
C_{21} = \{c_{2r1}, c_{2w1}\} > (c_{2w0}.C_{20} + c_{2w2}.C_{22})
\]

\[
C_{22} = \{c_{2r2}, c_{2w2}\} > (c_{2w0}.C_{20} + c_{2w1}.C_{21})
\]

\[
P_{14} = c_{2r0}.P_{15} + c_{2r1}.P_{15} + c_{2r2}.P_{16};
\]

\[
P_{21} = kr_{2}.P_{23} + kr_{1}.P_{22}
\]

\[
P_{22} = c_{1r0}.P_{23} + c_{1r1}.P_{21} + c_{1r2}.P_{21}
\]

\[
P_{23} = c_{2w2}.P_{24}
\]

\[
P_{24} = c_{1r0}.P_{25} + c_{1r1}.P_{25} + c_{1r2}.P_{26}
\]

\[
P_{26} = c_{2w1}.P_{21}
\]
Further observations

Knuth’s algorithm

- FASE has detected catastrophic cycles in both cases.
- When read acts are used, the follow catastrophic cycles is detected:

\[ C_{12} = \{c1r2, c1w2\} > (c1w0.C10 + c1w1.C11); \]
\[ C_{21} = \{c2r1, c2w1\} > (c2w0.C20 + c2w2.C22); \]
\[ C_{22} = \{c2r2, c2w2\} > (c2w0.C20 + c2w1.C21); \]

\[ P_{14} = c2r0.P15 + c2r1.P15 + c2r2.P16; \]
\[ P_{21} = kr2.P23 + kr1.P22 \]
\[ P_{22} = c1r0.P23 + c1r1.P21 + c1r2.P21 \]
\[ P_{23} = c2w2.P24 \]
\[ P_{24} = c1r0.P25 + c1r1.P25 + c1r2.P26 \]
\[ P_{26} = c2w1.P21 \]

\[ P = P_{14} \parallel P_{24} \parallel_B (C_{12} \parallel C_{22} \parallel K(2)) \xrightarrow{1} \]
Further observations

Knuth’s algorithm

- FASE has detected catastrophic cycles in both cases.
- When read acts are used, the following catastrophic cycles is detected:

\[ C_{12} = \{c_{1r2}, c_{1w2}\} > (c_{1w0}.C_{10} + c_{1w1}.C_{11}) \]
\[ C_{21} = \{c_{2r1}, c_{2w1}\} > (c_{2w0}.C_{20} + c_{2w2}.C_{22}) \]
\[ C_{22} = \{c_{2r2}, c_{2w2}\} > (c_{2w0}.C_{20} + c_{2w1}.C_{21}) \]

\[ P_{14} = c_{2r0}.P_{15} + c_{2r1}.P_{15} + c_{2r2}.P_{16} \]
\[ P_{21} = kr_{2}.P_{23} + kr_{1}.P_{22} \]
\[ P_{22} = c_{1r0}.P_{23} + c_{1r1}.P_{21} + c_{1r2}.P_{21} \]
\[ P_{23} = c_{2w2}.P_{24} \]
\[ P_{24} = c_{1r0}.P_{25} + c_{1r1}.P_{25} + c_{1r2}.P_{26} \]
\[ P_{26} = c_{2w1}.P_{21} \]

\[ P = P_{14} || P_{24} || B (C_{12} || C_{22} || K(2)) \xrightarrow{1} \]
\[ P' = P_{14} || P_{24} || B (C_{12} || C_{22} || K(2)) \xrightarrow{\tau(c_{2w1})} \]
Further observations

Knuth’s algorithm

- FASE has detected catastrophic cycles in both cases.
- When read acts are used, the follow catastrophic cycles is detected:

\[
\begin{align*}
C_{12} &= \{c_1r_2, c_1w_2\} > (c_1w_0.C_{10} + c_1w_1.C_{11}); \\
C_{21} &= \{c_2r_1, c_2w_1\} > (c_2w_0.C_{20} + c_2w_2.C_{22}); \\
C_{22} &= \{c_2r_2, c_2w_2\} > (c_2w_0.C_{20} + c_2w_1.C_{21}); \\
\end{align*}
\]

\[
\begin{align*}
P_{14} &= c_2r_0.P_{15} + c_2r_1.P_{15} + c_2r_2.P_{16}; \\
P_{21} &= kr_2.P_{23} + kr_1.P_{22} \\
P_{22} &= c_1r_0.P_{23} + c_1r_1.P_{21} + c_1r_2.P_{21} \\
P_{23} &= c_2w_2.P_{24} \\
P_{24} &= c_1r_0.P_{25} + c_1r_1.P_{25} + c_1r_2.P_{26} \\
P_{26} &= c_2w_1.P_{21} \\
\end{align*}
\]

\[
\begin{align*}
P &= P_{14} \parallel P_{24} \parallel_B (C_{12} \parallel C_{22} \parallel K(2)) \xrightarrow{1} \\
P' &= P_{14} \parallel P_{24} \parallel_B (C_{12} \parallel C_{22} \parallel K(2)) \xrightarrow{\tau(c_2w_1)} \\
P'' &= P_{14} \parallel P_{21} \parallel_B (C_{12} \parallel C_{21} \parallel K(2))
\end{align*}
\]
Further observations

Knuth’s algorithm

- FASE has detected catastrophic cycles in both cases.
- When read acts are used, the follow catastrophic cycles is detected:

\[ C_{12} = \{c_{1r2}, c_{1w2}\} > (c_{1w0}.C_{10} + c_{1w1}.C_{11}) ; \]
\[ C_{21} = \{c_{2r1}, c_{2w1}\} > (c_{2w0}.C_{20} + c_{2w2}.C_{22}) ; \]
\[ C_{22} = \{c_{2r2}, c_{2w2}\} > (c_{2w0}.C_{20} + c_{2w1}.C_{21}) ; \]

\[ P_{14} = c_{2r0}.P_{15} + c_{2r1}.P_{15} + c_{2r2}.P_{16} ; \]
\[ P_{21} = k_{r2}.P_{23} + k_{r1}.P_{22} \]
\[ P_{22} = c_{1r0}.P_{23} + c_{1r1}.P_{21} + c_{1r2}.P_{21} \]
\[ P_{23} = c_{2w2}.P_{24} \]
\[ P_{24} = c_{1r0}.P_{25} + c_{1r1}.P_{25} + c_{1r2}.P_{26} \]
\[ P_{26} = c_{2w1}.P_{21} \]

\[ P = P_{14} \parallel P_{24} \parallel_B (C_{12} \parallel C_{22} \parallel K(2)) \xrightarrow{1} \]
\[ P' = P_{14} \parallel P_{24} \parallel_B (C_{12} \parallel C_{22} \parallel K(2)) \xrightarrow{\tau(c_{2w1})} \]
\[ P'' = P_{14} \parallel P_{21} \parallel_B (C_{12} \parallel C_{21} \parallel K(2)) \xrightarrow{1} \]
Further observations

Knuth’s algorithm

- FASE has detected catastrophic cycles in both cases.
- When read acts are used, the following catastrophic cycles is detected:

\[ C_{12} = \{c_{1r2}, c_{1w2}\} > (c_{1w0} . C_{10} + c_{1w1} . C_{11}); \]
\[ C_{21} = \{c_{2r1}, c_{2w1}\} > (c_{2w0} . C_{20} + c_{2w2} . C_{22}); \]
\[ C_{22} = \{c_{2r2}, c_{2w2}\} > (c_{2w0} . C_{20} + c_{2w1} . C_{21}); \]

\[ P_{14} = c_{2r0} . P_{15} + c_{2r1} . P_{15} + c_{2r2} . P_{16}; \]
\[ P_{21} = kr_{2} . P_{23} + kr_{1} . P_{22} \]
\[ P_{22} = c_{1r0} . P_{23} + c_{1r1} . P_{21} + c_{1r2} . P_{21} \]
\[ P_{23} = c_{2w2} . P_{24} \]
\[ P_{24} = c_{1r0} . P_{25} + c_{1r1} . P_{25} + c_{1r2} . P_{26} \]
\[ P_{26} = c_{2w1} . P_{21} \]

\[ P = P_{14} \parallel P_{24} \parallel_B (C_{12} \parallel C_{22} \parallel K(2)) \xrightarrow{1} \]
\[ P' = P_{14} \parallel P_{24} \parallel_B (C_{12} \parallel C_{22} \parallel K(2)) \xrightarrow{\tau(c_{2w1})} \]
\[ P'' = P_{14} \parallel P_{21} \parallel_B (C_{12} \parallel C_{21} \parallel K(2)) \xrightarrow{1} \xrightarrow{\tau(kr_{2})} \]
Further observations

Knuth’s algorithm

- FASE has detected catastrophic cycles in both cases.
- When read acts are used, the follow catastrophic cycles is detected:

\[ C_{12} = \{c_{1r2}, c_{1w2}\} > (c_{1w0}.C_{10} + c_{1w1}.C_{11}); \]
\[ C_{21} = \{c_{2r1}, c_{2w1}\} > (c_{2w0}.C_{20} + c_{2w2}.C_{22}); \]
\[ C_{22} = \{c_{2r2}, c_{2w2}\} > (c_{2w0}.C_{20} + c_{2w1}.C_{21}); \]

\[ P_{14} = c_{2r0}.P_{15} + c_{2r1}.P_{15} + c_{2r2}.P_{16}; \]
\[ P_{21} = k_{r2}.P_{23} + k_{r1}.P_{22} \]
\[ P_{22} = c_{1r0}.P_{23} + c_{1r1}.P_{21} + c_{1r2}.P_{21} \]
\[ P_{23} = c_{2w2}.P_{24} \]
\[ P_{24} = c_{1r0}.P_{25} + c_{1r1}.P_{25} + c_{1r2}.P_{26} \]
\[ P_{26} = c_{2w1}.P_{21} \]

\[ P = P_{14} \parallel P_{24} \parallel_B (C_{12} \parallel C_{22} \parallel K(2)) \xrightarrow{1} \]
\[ P' = P_{14} \parallel P_{24} \parallel_B (C_{12} \parallel C_{22} \parallel K(2)) \xrightarrow{\tau(c_{2w1})} \]
\[ P'' = P_{14} \parallel P_{21} \parallel_B (C_{12} \parallel C_{21} \parallel K(2)) \xrightarrow{1} \xrightarrow{\tau(k_{r2})} \xrightarrow{\tau(c_{2w2})} P \]
And...

Thanks!