Concurrency Theory

Lecture 8: Modelling and Analysing Mutual Exclusion Algorithms

Joost-Pieter Katoen Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH AACHEN UNIVERSITY

{katoen,noll}@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/ct13/

Winter Semester 2013/14

Outline

Peterson's Mutual Exclusion Algorithm

- **Goal:** ensuring exclusive access to non-shared resources
- Here: two competing processes P_1, P_2 and shared variables
  - b_1, b_2 (Boolean, initially `false`)
  - k (in $\{1, 2\}$, arbitrary initial value)
- P_i uses local variable $j := 2 - i$ (index of other process)

Algorithm (Peterson's algorithm for P_i)

while true do
  "non-critical section";
  $b_i :=$ true;
  $k := j$;
  while $b_j \wedge k = j$ do skip;
  "critical section";
  $b_i :=$ false;
end

Representing Shared Variables in CCS

- Not directly expressible in CCS (communication by message passing)
- Idea: consider variables as processes that communicate with environment by processing read/write requests

Example (Shared variables in Peterson's algorithm)

- Encoding of b_1 with two (process) states B_{1t} (value tt) and B_{1f} (ff)
- Read access along ports $b1rt$ (in state B_{1t}) and $b1rf$ (in state B_{1f})
- Write access along ports $b1wt$ and $b1wf$ (in both states)
- Possible behaviours:

$$\begin{aligned} B_{1f} &= \overline{b1rf}.B_{1f} + b1wf.B_{1f} + b1wt.B_{1t} \\ B_{1t} &= \overline{b1rt}.B_{1t} + b1wf.B_{1f} + b1wt.B_{1t} \end{aligned}$$

- Similarly for b_2 and k:

$$\begin{aligned} B_{2f} &= \overline{b2rf}.B_{2f} + b2wf.B_{2f} + b2wt.B_{2t} \\ B_{2t} &= \overline{b2rt}.B_{2t} + b2wf.B_{2f} + b2wt.B_{2t} \end{aligned}$$

$$\begin{aligned} K_1 &= \overline{kr1}.K_1 + kw1.K_1 + kw2.K_2 \\ K_2 &= \overline{kr2}.K_2 + kw1.K_1 + kw2.K_2 \end{aligned}$$

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

while true do
  "non-critical section";
  b_i := true;
  $k := j$;
  while $b_j \wedge k = j$ **do skip**;
  "critical section";
  b_i := false;
end

CCS representation

$$
\begin{aligned}
P_1 &= \overline{b1wt}.\overline{kw2}.P_{11} \\
P_{11} &= b2rf.P_{12} + \\
&\quad b2rt.(kr1.P_{12} + kr2.P_{11}) \\
P_{12} &= enter_1.exit_1.\overline{b1wf}.P_1 \\
P_2 &= \overline{b2wt}.\overline{kw1}.P_{21} \\
P_{21} &= b1rf.P_{22} + \\
&\quad b1rt.(kr1.P_{21} + kr2.P_{22}) \\
P_{22} &= enter_2.exit_2.\overline{b2wf}.P_2
\end{aligned}
$$

$$Peterson = (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L$$

where

$$
\begin{aligned}
L = \{&b1rf, b1rt, b1wf, b1wt, \\
&b2rf, b2rt, b2wf, b2wt, \\
&kr1, kr2, kw1, kw2\}
\end{aligned}
$$

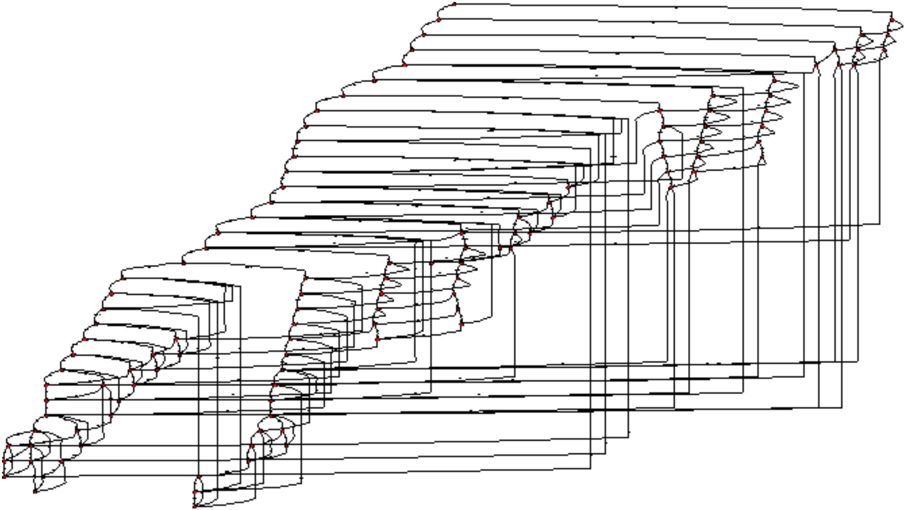
Outline

Obtaining the LTS I

Alternatives:

- By hand (really painful)
- By tools:
  - Edinburgh Concurrency Workbench
    - http://homepages.inf.ed.ac.uk/perdita/cwb/
    - see exercises
  - TAPAs ("Tool for the Analysis of Process Algebras")
    - http://rap.dsi.unifi.it/tapas/
    - CCS specification of Peterson's algorithm available as example
    - yields LTS with 115 states (see next slide)

Obtaining the LTS II



Outline

The Mutual Exclusion Property

- **Done:** formal description of Peterson's algorithm
- **To do:** analysing its behaviour (manually or with tool support)
- **Question:** what does "ensuring mutual exclusion" formally mean?

Mutual exclusion

At no point in the execution of the algorithm, processes P_1 and P_2 will both be in their critical section at the same time.

Alternatively:
It is always the case that either P_1 or P_2 or both are not in their critical section.

Specifying Mutual Exclusion in HML

Mutual exclusion

It is always the case that either P_1 or P_2 or both are not in their critical section.

Observations:

- Mutual exclusion is an invariance property ("always")
- P_i is in its critical section iff action $exit_i$ is enabled

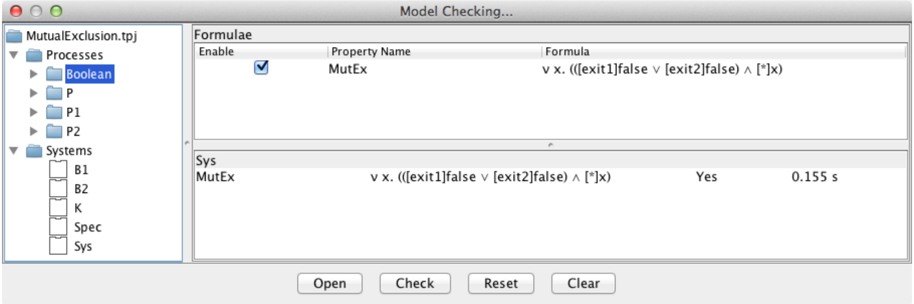
Mutual exclusion in HML

$$\begin{aligned} MutEx &:= Inv(F) \\ Inv(F) &\overset{max}{=} F \wedge [Act]Inv(F) \qquad \text{(cf. Theorem 6.5)} \\ F &:= [exit_1]\mathrm{ff} \vee [exit_2]\mathrm{ff} \end{aligned}$$

Model Checking Mutual Exclusion

- Using TAPAs Tool
- Supports property specifications in μ-calculus:

```
property MutEx:
max x. (([exit1] false | [exit2] false) & ([*] x))
end
```



Outline

Verification by Bisimulation Checking

- Alternative to logic-based approaches
- **Idea:** establish equivalence between (concrete) "implementation" and (abstract) "specification"

Example 8.1 (Two-place buffers (cf. Example 2.5))

1. Sequential specification:

$$\begin{aligned} B_0 &= in.B_1 \\ B_1 &= \overline{out}.B_0 + in.B_2 \\ B_2 &= \overline{out}.B_1 \end{aligned}$$

2. Parallel implementation:

$$\begin{aligned} B_\parallel &= (B[f] \parallel B[g]) \setminus com \\ B &= in.\overline{out}.B \end{aligned}$$

where $f := [out \mapsto com]$ and $g := [in \mapsto com]$

Later: (1) and (2) are "weakly bisimilar" (i.e., bisimilar up to τ-transitions)

Specifying Mutual Exclusion in CCS

- **Goal:** express desired behaviour of mutual exclusion algorithm as an "abstract" CCS process
- Intuitively:
  1. initially, either P_1 or P_2 can enter its critical section
  2. once this happened, the other process cannot enter the critical section before the first has exited it

Mutual exlusion in CCS

$$MutExSpec = enter_1.exit_1.MutExSpec + enter_2.exit_2.MutExSpec$$

Again: *Peterson* and *MutExSpec* are "weakly bisimilar"