Concurrency Theory

Winter Semester 2015/16

Lecture 7: Modelling and Analysing Mutual Exclusion Algorithms

Joost-Pieter Katoen and Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

http://moves.rwth-aachen.de/teaching/ws-1516/ct/

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Syntax of Mutually Recursive Equational Systems

Definition (Syntax of mutually recursive equational systems)

Let $\mathcal{X} = \{X_1, \ldots, X_n\}$ be a set of variables. The set $HMF_{\mathcal{X}}$ of Hennessy-Milner formulae over $\mathcal{X}$ is defined by the following syntax:

$$
\begin{array}{rlll}
F ::= & X_i & & \text{(variable)} \\
\mid & \mathrm{tt} & & \text{(true)} \\
\mid & \mathrm{ff} & & \text{(false)} \\
\mid & F_1 \wedge F_2 & & \text{(conjunction)} \\
\mid & F_1 \vee F_2 & & \text{(disjunction)} \\
\mid & \langle\alpha\rangle F & & \text{(diamond)} \\
\mid & [\alpha]F & & \text{(box)}
\end{array}
$$

where $1 \leq i \leq n$ and $\alpha \in Act$. A mutually recursive equational system has the form

$$(X_i = F_{X_i} \mid 1 \leq i \leq n)$$

where $F_{X_i} \in HMF_{\mathcal{X}}$ for every $1 \leq i \leq n$.

Semantics of Recursive Equational Systems I

As before: semantics of formula depends on states satisfying the variables

Definition (Semantics of mutually recursive equational systems)

Let $(S, Act, \longrightarrow)$ be an LTS and $E = (X_i = F_{X_i} \mid 1 \leq i \leq n)$ a mutually recursive equational system. The semantics of E, $[\![E]\!] : (2^S)^n \to (2^S)^n$, is defined by

$$[\![E]\!](T_1, \dots, T_n) := ([\![F_{X_1}]\!](T_1, \dots, T_n), \dots, [\![F_{X_n}]\!](T_1, \dots, T_n))$$

where

$$\begin{aligned}
[\![X_i]\!](T_1, \dots, T_n) &:= T_i \\
[\![\mathrm{tt}]\!](T_1, \dots, T_n) &:= S \\
[\![\mathrm{ff}]\!](T_1, \dots, T_n) &:= \emptyset \\
[\![F_1 \wedge F_2]\!](T_1, \dots, T_n) &:= [\![F_1]\!](T_1, \dots, T_n) \cap [\![F_2]\!](T_1, \dots, T_n) \\
[\![F_1 \vee F_2]\!](T_1, \dots, T_n) &:= [\![F_1]\!](T_1, \dots, T_n) \cup [\![F_2]\!](T_1, \dots, T_n) \\
[\![\langle \alpha \rangle F]\!](T_1, \dots, T_n) &:= \langle \cdot \alpha \cdot \rangle([\![F]\!](T_1, \dots, T_n)) \\
[\![[\alpha] F]\!](T_1, \dots, T_n) &:= [\cdot \alpha \cdot]([\![F]\!](T_1, \dots, T_n))
\end{aligned}$$

Semantics of Recursive Equational Systems II

Lemma

Let $(S, Act, \longrightarrow)$ *be a finite LTS and* $E = (X_i = F_{X_i} \mid 1 \leq i \leq n)$ *a mutually recursive equational system. Let* $(D, \sqsubseteq)$ *be given by* $D := (2^S)^n$ *and*

$$(T_1, \ldots, T_n) \sqsubseteq (T'_1, \ldots, T'_n)$$

iff $T_i \subseteq T'_i$ *for every* $1 \leq i \leq n$*.*

1. $(D, \sqsubseteq)$ *is a complete lattice with*
$$\bigsqcup\{(T^i_1, \ldots, T^i_n) \mid i \in I\} = \left(\bigcup\{T^i_1 \mid i \in I\}, \ldots, \bigcup\{T^i_n \mid i \in I\}\right)$$
$$\bigsqcap\{(T^i_1, \ldots, T^i_n) \mid i \in I\} = \left(\bigcap\{T^i_1 \mid i \in I\}, \ldots, \bigcap\{T^i_n \mid i \in I\}\right)$$
2. $[\![E]\!]$ *is monotonic w.r.t.* $(D, \sqsubseteq)$
3. $\text{fix}([\![E]\!]) = [\![E]\!]^m(\emptyset, \ldots, \emptyset)$ *for some* $m \in \mathbb{N}$
4. $\text{FIX}([\![E]\!]) = [\![E]\!]^M(S, \ldots, S)$ *for some* $M \in \mathbb{N}$

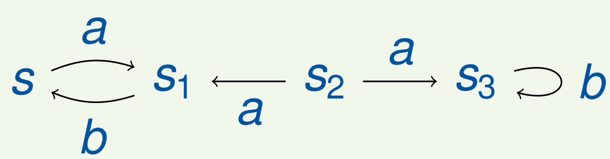
Proof.

omitted □

An Example

A Mutually Recursive Specification

Example 7.1



Let $S := \{s, s_1, s_2, s_3\}$ and E given by

$$X \overset{max}{=} \langle a \rangle Y \wedge [a]Y \wedge [b]\mathrm{ff}$$
$$Y \overset{max}{=} \langle b \rangle X \wedge [b]X \wedge [a]\mathrm{ff}$$

Computation of $\mathrm{FIX}(\llbracket E \rrbracket)$: on the board

Mixing Least and Greatest Fixed Points I

- **So far:** least/greatest fixed point of overall system
- **But:** too restrictive

Example 7.2

> *"It is possible for the system to reach a state which has a livelock (i.e., an infinite sequence of internal steps)."*

can be specified by

$$Pos(Livelock)$$

where

$$Pos(F) \stackrel{min}{=} F \vee \langle Act \rangle Pos(F) \qquad \text{(cf. Example 4.6)}$$
$$Livelock \stackrel{max}{=} \langle \tau \rangle Livelock$$

(thus, $Livelock \equiv Forever(\tau)$ [cf. Example 6.3])

Mixing Least and Greatest Fixed Points II

Caveat: arbitrary mixing can entail non-monotonic behaviour

Example 7.3

$$E : X \overset{min}{=} Y$$
$$Y \overset{max}{=} X$$

Fixed-point iteration:

$$(\bot, \top) = (\emptyset, S) \overset{[\![E]\!]}{\mapsto} (S, \emptyset) \overset{[\![E]\!]}{\mapsto} (\emptyset, S) \overset{[\![E]\!]}{\mapsto} \ldots$$

Solution: nesting of specifications by partitioning equations into a sequence of blocks such that all equations in one block

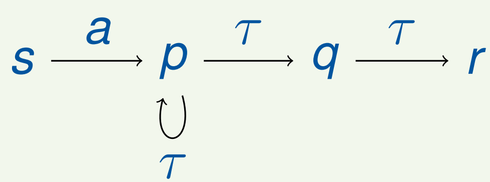
- are of same type (either *min* or *max*) and
- use only variables defined in the same or subsequent blocks

$\Longrightarrow$ bottom-up, block-wise evaluation by fixed-point iteration

Mixing Least and Greatest Fixed Points III

Example 7.4 (cf. Example 7.2)

$$\begin{aligned} PosLL &\overset{min}{=} Livelock \vee \langle Act \rangle PosLL \\ Livelock &\overset{max}{=} \langle \tau \rangle Livelock \end{aligned}$$



1. Fixed-point iteration for $Livelock : T \mapsto \langle \cdot \tau \cdot \rangle(T)$:
$$S = \{s, p, q, r\} \mapsto \{p, q\} \mapsto \{p\} \mapsto \{p\}$$
2. Fixed-point iteration for $PosLL : T \mapsto \{p\} \cup \langle \cdot Act \cdot \rangle(T)$:
$$\emptyset \mapsto \{p\} \mapsto \{s, p\} \mapsto \{s, p\}$$

Mixing Least and Greatest Fixed Points

The Modal μ-Calculus

- Logic that supports free mixing of least and greatest fixed points:
  - D. Kozen: *Results on the Propositional μ-Calculus*, Theoretical Computer Science 27, 1983, 333–354
- HML variants are fragments thereof
- Expressivity increases with alternation of least and greatest fixed points:
  - J.C. Bradfield: *The Modal Mu-Calculus Alternation Hierarchy is Strict*, Theoretical Computer Science 195(2), 1998, 133–153
- Decidable model-checking problem for finite LTSs
  (in NP $\cap$ co-NP; linear for HML with one variable)
- Generally undecidable for infinite LTSs and HML with one variable (CCS, Petri nets, ...)
- Overview paper:
  - O. Burkart, D. Caucal, F. Moller, B. Steffen: *Verification on Infinite Structures*, Chapter 9 of *Handbook of Process Algebra*, Elsevier, 2001, 545–623

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 7.5 (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 7.6 (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: $$B_{1f} = \overline{b1rf}.B_{1f} + b1wf.B_{1f} + b1wt.B_{1t}$$ $$B_{1t} = \overline{b1rt}.B_{1t} + b1wf.B_{1f} + b1wt.B_{1t}$$
- Similarly for b_2 and k: $$B_{2f} = \overline{b2rf}.B_{2f} + b2wf.B_{2f} + b2wt.B_{2t}$$ $$B_{2t} = \overline{b2rt}.B_{2t} + b2wf.B_{2f} + b2wt.B_{2t}$$

$$K_1 = \overline{kr1}.K_1 + kw1.K_1 + kw2.K_2$$
$$K_2 = \overline{kr2}.K_2 + kw1.K_1 + kw2.K_2$$

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

$$\begin{aligned}
P_1 &= \overline{b1wt}.\overline{kw2}.P_{11} \\
P_{11} &= b2rf.P_{12} + \\
&\quad b2rt.(kr1.P_{12} + kr2.P_{11}) \\
P_{12} &= enter_1.exit_1.\overline{b1wf}.P_1 \\
P_2 &= \overline{b2wt}.\overline{kw1}.P_{21} \\
P_{21} &= b1rf.P_{22} + \\
&\quad b1rt.(kr1.P_{21} + kr2.P_{22}) \\
P_{22} &= enter_2.exit_2.\overline{b2wf}.P_2 \\
Peterson &= (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L \\
\text{for } L &= \{b1rf, b1rt, b1wf, b1wt, \\
&\quad b2rf, b2rt, b2wf, b2wt, \\
&\quad kr1, kr2, kw1, kw2\}
\end{aligned}$$

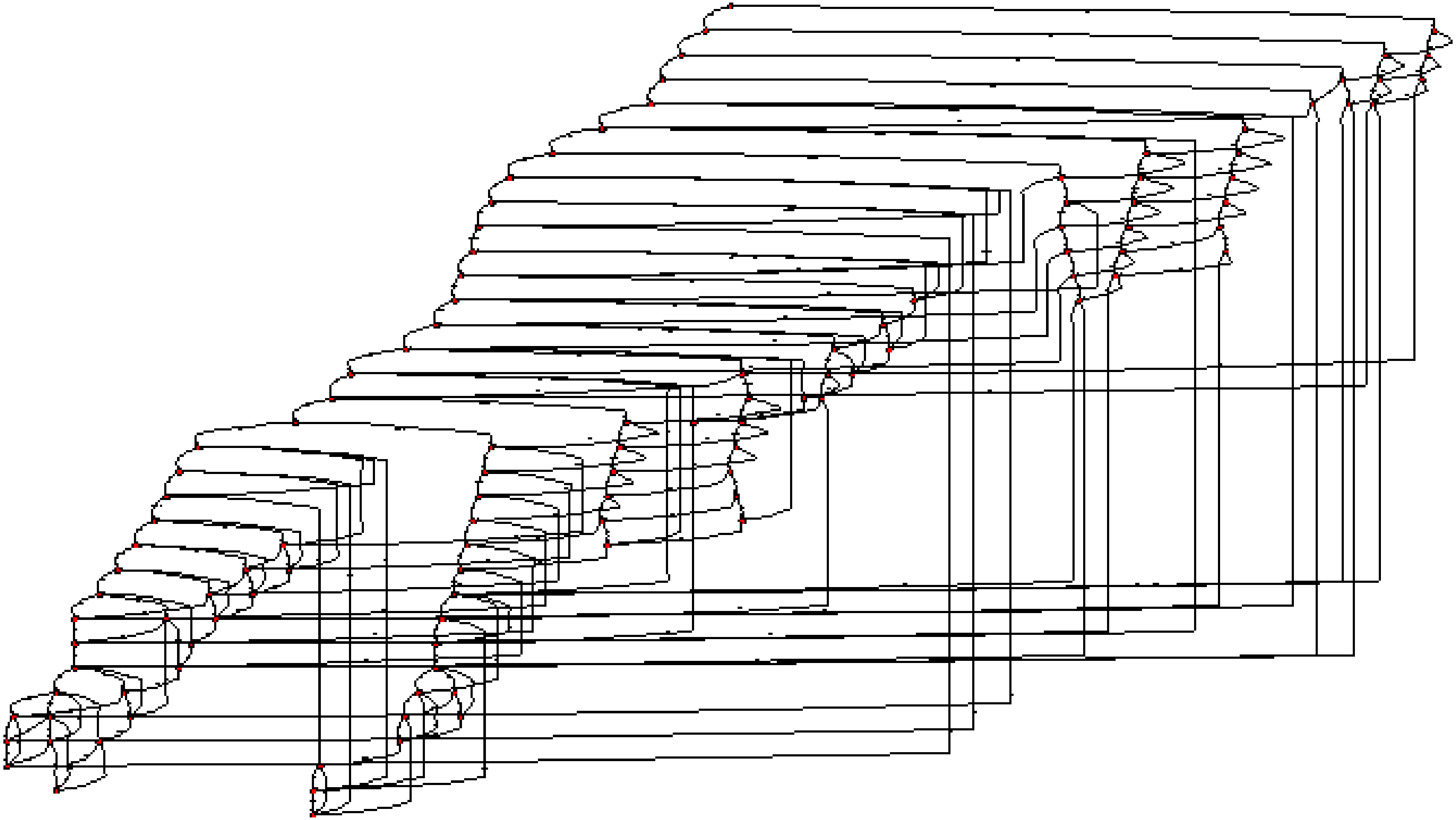
Evaluating the CCS Model

Obtaining the LTS I

Alternatives:

- By hand (really painful)
- By tools:
  - CAAL (Concurrency Workbench, Aalborg Edition): http://caal.cs.aau.dk
    - smart editor
    - visualisation of generated LTS
    - equivalence checking w.r.t. several bisimulation, simulation and trace equivalences
    - generation of distinguishing formulae for nonequivalent processes
    - model checking of recursive HML formulae
    - (bi)simulation and model checking games.
    - 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)
  - CWB (Edinburgh Concurrency Workbench): http://homepages.inf.ed.ac.uk/perdita/cwb/
    - somewhat outdated

Obtaining the LTS II



Model Checking Mutual Exclusion

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

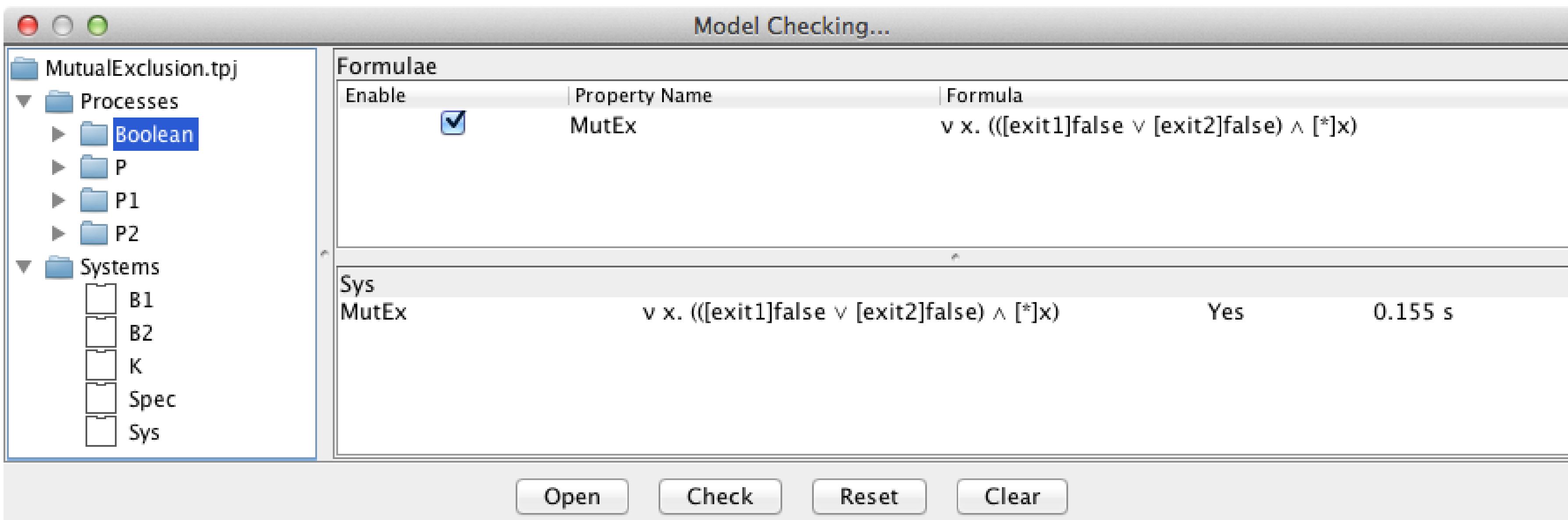
Mutual exclusion in HML

$$\begin{aligned} MutEx &:= Inv(F) \\ Inv(F) &\stackrel{max}{=} F \wedge [Act]Inv(F) \qquad \text{(cf. Theorem 6.2)} \\ F &:= [exit_1]\mathrm{ff} \vee [exit_2]\mathrm{ff} \end{aligned}$$

Model Checking Mutual Exclusion

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

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



Alternative Verification Approaches

Verification by Bisimulation Checking

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

Example 7.7 (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”