Lezione 23 MSC
Peterson's mutual exclusion algorithm

Roberto Gorrieri

Peterson's algorithm

- Ingredients: Two processes P_1 and P_2, two boolean variables b_1 and b_2 (initialized to *false*), an integer variable k that can assume value 1 or 2 (arbitrarily initialized)
- Algorithm of P_i

(j is the index of the other)

- Is it correct? Does it ensure mutual exclusion?
- Formal proof?

while true do
begin
  'noncritical section';
  b_i := **true**;
  $k := j$;
  while (b_j **and** $k = j$) **do skip**;
  'critical section';
  b_i := **false**;
end

How to prove the algorithm correct?

- Build a CCS model for the entities involved in the algorithm:
  - Modeling variables b1, b2 and k
  - Modeling processes P1 and P2
- Use model-checking, by first specifying a logic formula that captures the property of interest.
- Alternatively, use equivalence-checking, by first specifying a self-evidently correct CCS specification of mutual exclusion

Modeling the variables

Variable b1 can be modeled as a two-state specification, offering in output (read operation r) the stored value (f false, t true) and accepting in input any write (w) request:

$$B_{1f} \stackrel{\text{def}}{=} \overline{\text{b1rf}}.B_{1f} + \text{b1wf}.B_{1f} + \text{b1wt}.B_{1t},$$
$$B_{1t} \stackrel{\text{def}}{=} \overline{\text{b1rt}}.B_{1t} + \text{b1wf}.B_{1f} + \text{b1wt}.B_{1t}.$$

Similarly b2

$$B_{2f} \stackrel{\text{def}}{=} \overline{\text{b2rf}}.B_{2f} + \text{b2wf}.B_{2f} + \text{b2wt}.B_{2t},$$
$$B_{2t} \stackrel{\text{def}}{=} \overline{\text{b2rt}}.B_{2t} + \text{b2wf}.B_{2f} + \text{b2wt}.B_{2t}.$$

And k

$$K_1 \stackrel{\text{def}}{=} \overline{\text{kr1}}.K_1 + \text{kw1}.K_1 + \text{kw2}.K_2,$$
$$K_2 \stackrel{\text{def}}{=} \overline{\text{kr2}}.K_2 + \text{kw1}.K_1 + \text{kw2}.K_2.$$

Modeling the two processes

Assumptions:

- Ignore "non critical section"
- Ignore "critical section"
- Focus only on the mutual exclusion mechanism
- The execution of "critical section" is always performed in a finite amount of time (no deadlock, no divergence)

P_i

while true do
begin
  'noncritical section';
  b_i := **true**;
  $k := j$;
  while (b_j **and** $k = j$) **do skip**;
  'critical section';
  b_i := **false**;
end

Modelling P_1

$$P_1 \stackrel{\text{def}}{=} \overline{\text{b1wt}}.\overline{\text{kw2}}.\text{P}_{11}$$

- The boolean test is evaluated by a lazy left-to-right rule (esterna sinistra)

while true do
begin
  'noncritical section';
  b_i := **true**;
  $k := j$;
  while $(b_j$ **and** $k = j)$ **do skip**;
  'critical section';
  b_i := **false**;
end

$$\text{P}_{11} \stackrel{\text{def}}{=} \text{b2rf}.\text{P}_{12} + \text{b2rt}.(\text{kr2}.\text{P}_{11} + \text{kr1}.\text{P}_{12})$$

$$P_{12} \stackrel{\text{def}}{=} \text{enter1}.\text{exit1}.\overline{\text{b1wf}}.\text{P}_1$$

Modeling Peterson's algorithm

$$P_2 \stackrel{\text{def}}{=} \overline{\mathrm{b2wt}}.\overline{\mathrm{kw1}}.\mathrm{P}_{21},$$

$$\mathrm{P}_{21} \stackrel{\text{def}}{=} \mathrm{b1rf}.\mathrm{P}_{22} + \mathrm{b1rt}.(\mathrm{kr1}.\mathrm{P}_{21} + \mathrm{kr2}.\mathrm{P}_{22}),$$

$$\mathrm{P}_{22} \stackrel{\text{def}}{=} \mathrm{enter2}.\mathrm{exit2}.\overline{\mathrm{b2wf}}.P_2.$$

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

where

- L = {b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2}
- that is: L = Act \ {enter1, enter2, exit1, exit2}, which are the only observable actions!

Mutual exclusion in HML with recursion

- Safety property: "it never happens that both processes are in the critical section at the same time"
- Inv(F) $X \stackrel{\max}{=} F \wedge [\mathrm{Act}]X$ where F should be the property "the two processes are not both in the critical section now"
- Note that P_i is in the critical section if P_i –exit$_i$->
- Then

$$F \stackrel{\text{def}}{=} [\mathrm{exit}_1]ff \vee [\mathrm{exit}_2]ff$$ (weak modalities [[-]])

which is true in a state if it is not possible to perform both exit1 and exit2 (possibly with some tau's in between – weak modalities).

Model-checking mutual exclusion

- By hand, computing the greatest fixpoint of

$$S \mapsto [\![F]\!] \cap [\cdot\mathbf{Act}\cdot]S$$

- Or Checking that Peterson ⊨ Inv(F) with CWB, using the command >checkprop

- Exercise: is Inv(G), where G =

$$([\text{enter}_1][\text{enter}_2]\mathit{ff}) \wedge ([\text{enter}_2][\text{enter}_1]\mathit{ff})$$

a good specification of mutual exclusion? What about H = $((\langle\text{enter}_1\rangle[\text{enter}_2]\mathit{ff}) \wedge ((\langle\text{enter}_2\rangle[\text{enter}_1]\mathit{ff})$

Hyman's algorithm

- Under the same hypothesis of Peterson about initial values of variables, build its CCS model
- Is it correct? No. Why?
- Check with CWB that the CCS model for Hyman does not satisfy Inv(F)

P_i

while true do
begin
    'noncritical section';
    b_i := **true**;
    while $k \neq i$ **do begin**
        while b_j **do skip**;
        $k := i$
    end;
    'critical section';
    b_i := **false**;
end

CCS specification of mutual exclusion

- Equivalence-checking between the implementation (Peterson) and a suitable, self-evidently correct, CCS specification
- What specification? What equivalence?
- Possible specification:

$$\mathrm{MutexSpec} \stackrel{\mathrm{def}}{=} \mathrm{enter}_1.\mathrm{exit}_1.\mathrm{MutexSpec} + \mathrm{enter}_2.\mathrm{exit}_2.\mathrm{MutexSpec}.$$

- Nobody can tell us that MutexSpec is correct! We have to get convinced by ourselves.

Which equivalence?

- Not strong bisimilarity: Peterson performs tau's while MutexSpec doesn't
- Unfortunately, not weak bisimilarity! Peterson is not weak bisimilar to MutexSpec:

Consider P = (vL)(P12|P21|B1t|B2t|K1), i.e., a state in which the race for entering the critical section has been won by P1 (state P12), while P2 is testing the variables, trying to enter (P21). Then:

Peterson ==tau=> P –enter1->

where P cannot perform enter2 not even weakly.

MutexSpec can reply to this tau-step by idling but the reached state can perform both enter1 and enter2: so Peterson and MutexSpec are not weakly bisimilar.

Which equivalence? (2)

- Mutual exclusion is a Safety property ➔ no need to use bisimulation-based equivalences. Use instead trace preorder! Every weak trace of Peterson is also a trace of MutexSpec!

- Wtr(Peterson) $\subseteq$ Tr(MutexSpec) means that all Peterson's executions are conformant to mutual exclusion (SAFETY CONDITION)
- Note that also **0** satisfies the safety condition! Hence, we need one further condition to ensure that mutual exclusion is indeed a possible behavior of Peterson.

Equivalence-checking

- Tr(MutexSpec) $\subseteq$ Wtr(Peterson) means that any mutual exclusion behavior is possible for Peterson!
- Hence, use weak trace equivalence: WTr(MutexSpec) = Wtr(Peterson)
- In CWB there is the command >mayeq that can be used to check weak trace equivalence