Formale Spezifikation und Verifikation

Mirco Tribastone

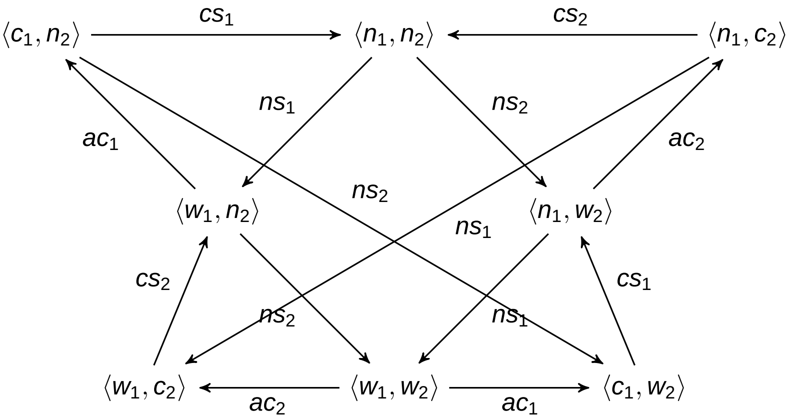
Institut für Informatik
Ludwig-Maximilians-Universität München
`tribastone@pst.ifi.lmu.de`

Verification with CCS

Overview

- CCS model of an exclusion algorithm
- Verification using *weak simulation*
- Verification of the model using CCS itself
- Check with ECW

Previously. . .



- Concurrent access was taken care of at a higher level of abstraction
- In no state are the two processes in the critical section

Peterson's Exclusion Algorithm[1]

while true do
  noncrititical section
  $b_i :=$ **true**
  $k := j$
  while $b_j \wedge k = j$ **do**
    skip
  end while
  critical section
  $b_i :=$ **false**
end while

- $i, j \in \{1, 2\}$ (process ids)
- b_1, b_2, k are shared variables
- $b_i =$ **true** means that process i is trying to enter the critical section
- k is the id of the process in the critical section
- Initially, $b_1 :=$ **false** and $b_2 :=$ **false**
- The initial value of k is left unspecified

[1] G.L. Peterson, Myths About the Mutual Exclusion Problem, *Information Processing Letters*, **12**(3), 115–116, 1981.

CCS Model of Peterson's Algorithm

1. Identify the collection of *communicating systems* and their channels
   - Obviously, process 1 and 2
   - Also, the shared variables can be seen as *passive agents* that react to actions performed by the processes
   - Processes communicate with variables through read and write operations
   - Processes do not communicate with each other explicitly
2. Describe the behaviour of each agent
3. Compose agents through parallel composition and restriction

Communicating Boolean Variables

- For each process i there is a boolean variable b_i
- b_i has two *local states* (i.e., **true** and **false**)

$$B_{1f} \triangleq \overline{b1rf}.B_{1f} + b1wf.B_{1f} + b1wt.B_{1t} \ ,$$
$$B_{1t} \triangleq \overline{b1rt}.B_{1t} + b1wf.B_{1f} + b1wt.B_{1t} \ .$$

Similarly,

$$B_{2f} \triangleq \overline{b2rf}.B_{2f} + b2wf.B_{2f} + b2wt.B_{2t} \ ,$$
$$B_{2t} \triangleq \overline{b2rt}.B_{2t} + b2wf.B_{2f} + b2wt.B_{2t} \ ,$$

where the pattern for the channel name is $b\langle i\rangle\langle x\rangle\langle y\rangle$, with

- $i \in \{1, 2\}$ the process id
- $x \in \{r, w\}$ the kind of operation
- $y \in \{f, t\}$ the variable value to be written or read

Model of the *turn* Variable k

In the case of a protocol with only two concurrent processes, k may only take values 1 and 2, respectively denoted by K_1 and K_2 in

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

where the pattern for the channel name is $k\langle x\rangle\langle n\rangle$, with

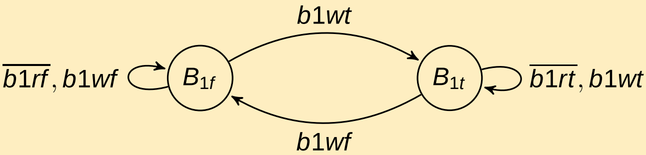
- $x \in \{r, w\}$ the kind of operation
- $n \in \{1, 2\}$ the value to be written or read

Exercise

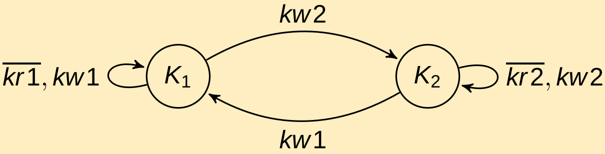
How does this model generalise to a variable v taking values over a data domain D?

Labelled Transition Systems So Far

LTS of B_{1f} (B_{2f} is similar)



LTS of K_1



Model of Process 1

while true do
noncrititical section
$b_i := \textbf{true}$
$k := j$
while $b_j \wedge k = j$ **do**
skip
end while
critical section
$b_i := \textbf{false}$
end while

- Abstraction: we ignore the process behaviour outside the critical section
- The process tries to enter:

$$P_1 \triangleq \overline{b1wt}.\overline{kw2}.P_{11}$$

- P_{11} models the **while** loop (with short-circuit evaluation):

$$P_{11} \triangleq b2rf.P_{12} + b2rt.(kr2.P_{11} + kr1.P_{12})$$

- P_{12} models the critical section:

$$P_{12} \triangleq enter_1.exit_1.\overline{b1wf}.P_1$$

Process 1 and Process 2

Process 1

$$P_1 \triangleq \overline{b1wt}.\overline{kw2}.P_{11}$$

$$P_{11} \triangleq b2rf.P_{12} + b2rt.(kr2.P_{11} + kr1.P_{12})$$

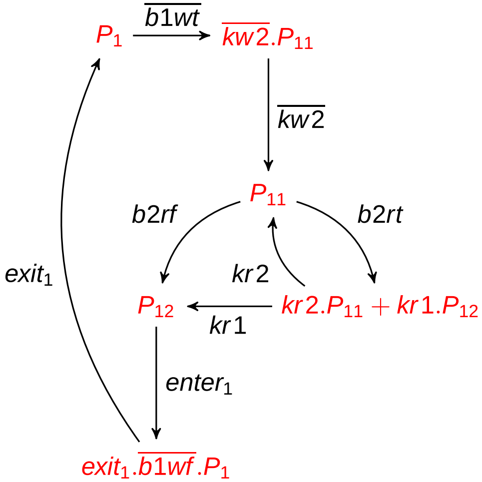
$$P_{12} \triangleq enter_1.exit_1.\overline{b1wf}.P_1$$

Process 2

$$P_2 \triangleq \overline{b2wt}.\overline{kw1}.P_{21}$$

$$P_{21} \triangleq b1rf.P_{22} + b1rt.(kr1.P_{21} + kr2.P_{22})$$

$$P_{22} \triangleq enter_2.exit_2.\overline{b2wf}.P_2$$



LTS of P_1

Peterson's System

$$
\begin{aligned}
B_{1f} &\triangleq \overline{b1rf}.B_{1f} + b1wf.B_{1f} + b1wt.B_{1t} \\
B_{1t} &\triangleq \overline{b1rt}.B_{1t} + b1wf.B_{1f} + b1wt.B_{1t} \\
B_{2f} &\triangleq \overline{b2rf}.B_{2f} + b2wf.B_{2f} + b2wt.B_{2t} \\
B_{2t} &\triangleq \overline{b2rt}.B_{2t} + b2wf.B_{2f} + b2wt.B_{2t} \\
K_1 &\triangleq \overline{kr1}.K_1 + kw1.K_1 + kw2.K_2 \\
K_2 &\triangleq \overline{kr2}.K_2 + kw1.K_1 + kw2.K_2 \\
P_1 &\triangleq \overline{b1wt}.\overline{kw2}.P_{11} \\
P_{11} &\triangleq b2rf.P_{12} + b2rt.(kr2.P_{11} + kr1.P_{12}) \\
P_{12} &\triangleq enter_1.exit_1.\overline{b1wf}.P_1 \\
P_2 &\triangleq \overline{b2wt}.\overline{kw1}.P_{21} \\
P_{21} &\triangleq b1rf.P_{22} + b1rt.(kr1.P_{21} + kr2.P_{22}) \\
P_{22} &\triangleq enter_2.exit_2.\overline{b2wf}.P_2 \\
Peterson &\triangleq \left(B_{1f} \mid B_{2f} \mid K_1 \mid P_1 \mid P_2\right)\backslash L \ , \\
L &= \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kw1, kr2, kw2\}
\end{aligned}
$$

Verification with CCS Itself

Informal Verification Criterion

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

A Variant

If one process, say P_1, is in its critical section, the other process P_2 may enter only after P_1 has exited its critical section.

How can we verify this property?

- Check if *Peterson* is *strongly* bisimilar to some other specification.
- Check if *Peterson* is *weakly* bisimilar to some other specification.
- Check if *Peterson weakly simulates* some other specification.
- Combine *Peterson* with an observer process that emits a *bad* action if the critical section is not accessed correctly.

Verification with Equivalence Relations

Informal Verification Criterion

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

Strong Bisimulation

$$MutexSpec \triangleq enter_1.exit_1.MutexSpec + enter_2.exit_2.MutexSpec \ .$$

Is $MutexSpec \sim Peterson$?

Using the game characterisation of strong bisimulation:

- Attacker says: $MutexSpec \xrightarrow{enter_1} exit_1.MutexSpec$
- Defender loses because no $enter_1$ action is enabled by *Peterson*:

$$Peterson \xrightarrow{\tau} (B_{1t} \mid B_{2f} \mid K_1 \mid \overline{kw2}.P_{11} \mid P_2)\backslash L \text{ , and}$$

$$Peterson \xrightarrow{\tau} (B_{1f} \mid B_{2t} \mid K_1 \mid P_1 \mid \overline{kw1}.P_{21})\backslash L \ .$$

Using Weak Bisimulation?

$$MutexSpec \triangleq enter_1.exit_1.MutexSpec + enter_2.exit_2.MutexSpec \ .$$

Is $MutexSpec \approx Peterson$?

1. Attacker chooses right for a sufficient number of times to have

$$Peterson \stackrel{\tau}{\Rightarrow} (B_{1t} \mid B_{2t} \mid P_{12} \mid P_{21} \mid K_1)\backslash L \ ,$$

2. to which the defender responds by

$$MutexSpec \stackrel{\tau}{\Rightarrow} MutexSpec \ .$$

3. Now, the attacher chooses left and says

$$MutexSpec \xrightarrow{enter_2} exit_2.MutexSpec$$

4. but the defender does not afford any $enter_2$-transitions.

Weak Simulation

Weak Traces and Weak Trace Equivalence

A *weak trace* of a process P is a sequence $a_1 \cdots a_k$, $k \geq 1$, of observable actions such that there exists a sequence of transitions

$$P = P_0 \stackrel{a_1}{\Longrightarrow} P_1 \stackrel{a_2}{\Longrightarrow} \cdots \stackrel{a_k}{\Longrightarrow} P_k \ ,$$

for some $P_1, \ldots, P_k$. Process P is a *weak trace approximation* of process Q if the set of weak traces of P is included in that of Q.
Two processes are *weak trace equivalent* if the afford the same weak traces.

Property

If a process does not afford internal transitions then its set of weak traces coincides with its set of traces.

Weak Simulation

- It is possible to show that *Peterson* is weak trace equivalent to *MutexSpec*.
- However, this is a stronger condition than we need to prove the correctness of the algorithm.
- We may be content with just verifying that *Peterson* is a weak trace approximation of *MutexSpec*.

Weak Simulation

A binary relation $\mathcal{R}$ over the set of states of an LTS is called a *weak simulation* iff, whenever $s_1 \mathcal{R} s_2$ and α is an action (including τ), if $s_1 \xrightarrow{\alpha} s_1'$ then there is a transition $s_2 \stackrel{\alpha}{\Rightarrow} s_2'$ such that $s_1' \mathcal{R} s_2'$.
We say that s' *weakly simulates* s iff there is a weak simulation $\mathcal{R}$ with $s \mathcal{R} s'$.

Proposition If s' weakly simulates s then each weak trace of s is also a weak trace of s'.

Verification Using CCS Itself: Observer Process

Informal Verification Criterion

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

- Once the observer has seen an enter action, say $enter_1$, it goes to a state where it may see the corresponding $exit_1$ action.
- However, in this new state it must not see $enter_2$. If it does observe $enter_2$, it must emit a *bad* action highlighting the breach of the protocol.
- Analogously, if the observer sees $enter_2$, it goes to a state where it may see $exit_2$.
- If it sees $enter_1$ instead, then it will emit the *bad* action.
- Once the correct exit action is observed, the observer goes back to a state where it may see either enter action.

Formalisation of the Verification Criterion

Informal Verification Criterion

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

Consider the following CCS process:

$$
\begin{aligned}
\mathit{MutexTest} &\triangleq \overline{\mathit{enter}_1}.\mathit{MutexTest}_1 + \overline{\mathit{enter}_2}.\mathit{MutexTest}_2 \\
\mathit{MutexTest}_1 &\triangleq \overline{\mathit{exit}_1}.\mathit{MutexTest} + \overline{\mathit{enter}_2}.\overline{\mathit{bad}}.\mathbf{0} \\
\mathit{MutexTest}_2 &\triangleq \overline{\mathit{exit}_2}.\mathit{MutexTest} + \overline{\mathit{enter}_1}.\overline{\mathit{bad}}.\mathbf{0}
\end{aligned}
$$

and combine it as follows

$$(\mathit{Peterson} \mid \mathit{MutexTest}) \backslash M \ ,$$

where $M = \{\mathit{enter}_1, \mathit{enter}_2, \mathit{exit}_1, \mathit{exit}_2\}$.

Verification with Observers

- Indeed, the LTS of $(Peterson \mid MutexTest)\backslash M$ does not have states which afford *bad* transitions.
- The observer does not affect the communicating behaviour inside *Peterson*, i.e., the process *MutexTest* is left unaltered if $Peterson \xrightarrow{\tau} Peterson'$.

$$\dfrac{\dfrac{Peterson \xrightarrow{\tau} Peterson'}{Peterson \mid MutexTest \xrightarrow{\tau} Peterson' \mid MutexTest}}{(Peterson \mid MutexTest)\backslash M \xrightarrow{\tau} (Peterson' \mid MutexTest)\backslash M}$$

Commands for ECW – 1

Model Declarations

```
agent B1f='b1rf.B1f + b1wf.B1f + b1wt.B1t;
agent B1t='b1rt.B1t + b1wf.B1f + b1wt.B1t;
...
set L={b1rf,b1rt,b1wf,b1wt,b2rf,b2rt,b2wf,b2wt,kr1,kw1,kr2,kw2};
agent Peterson = ( B1f | B2f | K1 | P1 | P2)  L;
```

Specification Declarations

```
agent MutexSpec= enter1.exit1.MutexSpec+enter2.exit2.MutexSpec;
```

Visualise All Declarations

```
print;
```

Commands for ECW – 2

Strong Bisimilarity

```
strongeq(Peterson, MutexSpec);
```

Weak Bisimilarity

```
eq(Peterson, MutexSpec);
```

Weak Trace Equivalence

```
mayeq(Peterson, MutexSpec);
```

Weak Simulation

```
agent Div = tau.Div;
pre(Peterson | Div, MutexSpec);
```