



Concurrency Theory

Winter Semester 2017/18

Lecture 7: Modelling and Analysing Mutual Exclusion Algorithms & Value-Passing CCS

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

<http://moves.rwth-aachen.de/teaching/ws-1718/ct/>

Modelling Mutual Exclusion Algorithms

Outline of Lecture 7

Modelling Mutual Exclusion Algorithms

Evaluating the CCS Model

Model Checking Mutual Exclusion

Alternative Verification Approaches

Syntax of Value-Passing CCS

Semantics of Value-Passing CCS

Translation of Value-Passing into Pure CCS

Modelling Mutual Exclusion Algorithms

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**) — b_i indicates that P_i wants to enter
 - k (in $\{1, 2\}$, arbitrary initial value) — index of prioritised process
- P_i uses local variable $j := 2 - i$ (index of other process)

Modelling Mutual Exclusion Algorithms

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**) — b_i indicates that P_i wants to enter
 - k (in $\{1, 2\}$, arbitrary initial value) — index of prioritised process
- P_i uses local variable $j := 2 - i$ (index of other process)

Algorithm 7.1 (Peterson's algorithm for P_i)

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

Modelling Mutual Exclusion Algorithms

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

Modelling Mutual Exclusion Algorithms

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 7.2 (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)

Modelling Mutual Exclusion Algorithms

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 7.2 (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: $B_{1f} = \overline{b1rf}.B_{1f} + b1wf.B_{1f} + b1wt.B_{1t}$
 $B_{1t} = \overline{b1rt}.B_{1t} + b1wf.B_{1f} + b1wt.B_{1t}$

Modelling Mutual Exclusion Algorithms

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 7.2 (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 Mutual Exclusion Algorithms

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

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

Modelling Mutual Exclusion Algorithms

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

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

CCS representation

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

for $L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$

Modelling Mutual Exclusion Algorithms

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

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

CCS representation

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

for $L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$

Modelling Mutual Exclusion Algorithms

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

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

CCS representation

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

for $L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$

Modelling Mutual Exclusion Algorithms

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

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

CCS representation

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

for $L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$

Modelling Mutual Exclusion Algorithms

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

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

CCS representation

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

for $L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$

Modelling Mutual Exclusion Algorithms

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

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

CCS representation

$$P_1 = \overline{b1wt}.\overline{kw2}.P_{11}$$
$$P_{11} = b2rf.P_{12} + b2rt.(kr1.P_{12} + kr2.P_{11})$$
$$P_{12} = \text{enter}_1.\text{exit}_1.\overline{b1wf}.P_1$$
$$P_2 = \overline{b2wt}.\overline{kw1}.P_{21}$$
$$P_{21} = b1rf.P_{22} + b1rt.(kr1.P_{21} + kr2.P_{22})$$
$$P_{22} = \text{enter}_2.\text{exit}_2.\overline{b2wf}.P_2$$
$$\text{Peterson} = (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L$$

for $L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$

Modelling Mutual Exclusion Algorithms

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

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

CCS representation

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

for $L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$

Evaluating the CCS Model

Outline of Lecture 7

Modelling Mutual Exclusion Algorithms

Evaluating the CCS Model

Model Checking Mutual Exclusion

Alternative Verification Approaches

Syntax of Value-Passing CCS

Semantics of Value-Passing CCS

Translation of Value-Passing into Pure CCS

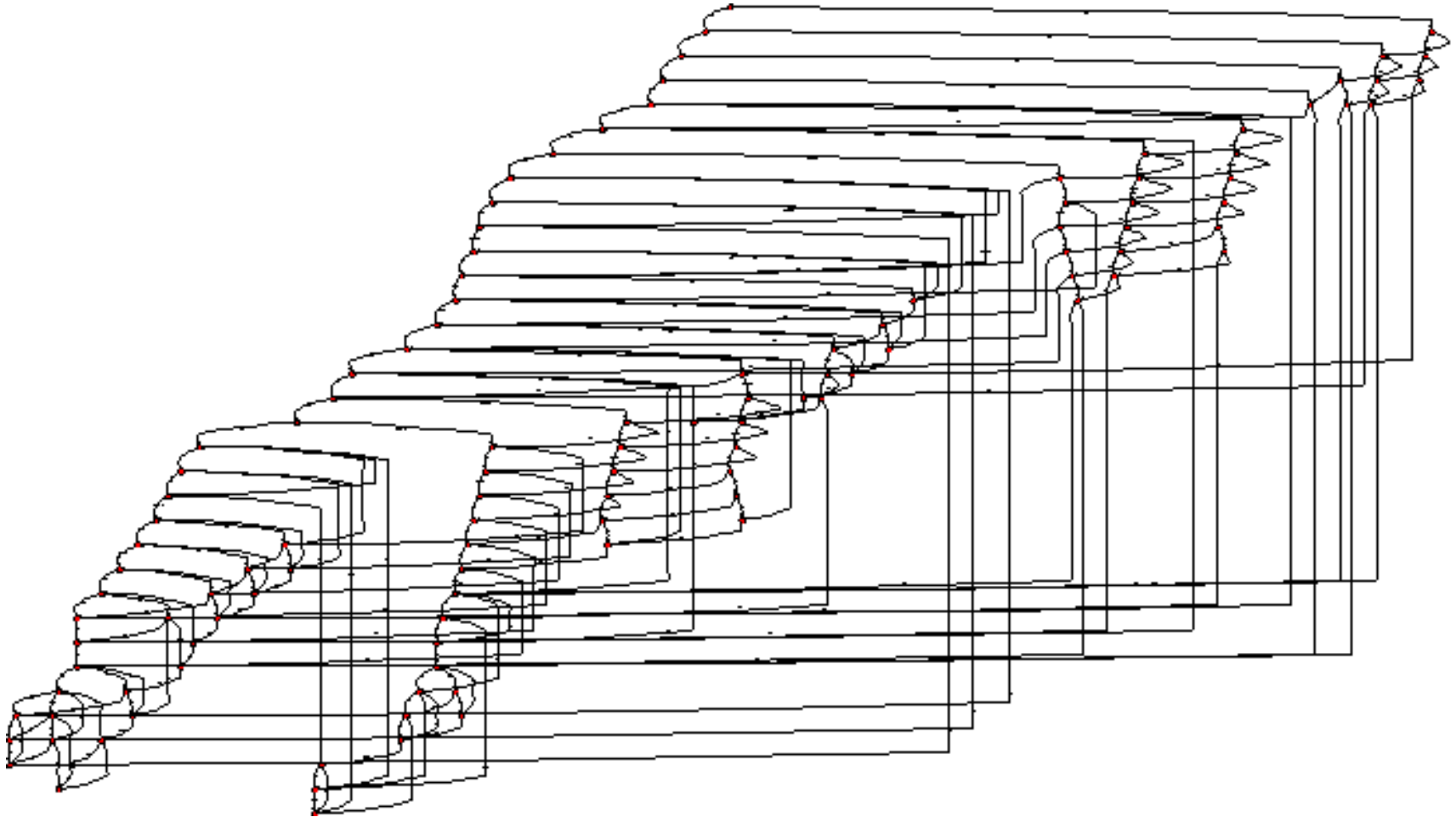
Evaluating the CCS Model

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

Outline of Lecture 7

Modelling Mutual Exclusion Algorithms

Evaluating the CCS Model

Model Checking Mutual Exclusion

Alternative Verification Approaches

Syntax of Value-Passing CCS

Semantics of Value-Passing CCS

Translation of Value-Passing into Pure CCS

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?

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.

Model Checking Mutual Exclusion

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.

Model Checking Mutual Exclusion

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

Model Checking Mutual Exclusion

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

$$\begin{aligned} MutEx &:= Inv(F) \\ Inv(F) &\stackrel{max}{=} F \wedge [Act]Inv(F) && \text{(cf. Theorem 6.1)} \\ F &:= [exit_1]ff \vee [exit_2]ff \end{aligned}$$

Model Checking Mutual Exclusion

Model Checking Mutual Exclusion

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

property MutEx:

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

Model Checking Mutual Exclusion

Model Checking Mutual Exclusion

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

property MutEx:

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

Enable	Property Name	Formula
<input checked="" type="checkbox"/>	MutEx	$\nu x. (([exit1]false \vee [exit2]false) \wedge [*]x)$

Sys	Formula	Result	Time
MutEx	$\nu x. (([exit1]false \vee [exit2]false) \wedge [*]x)$	Yes	0.155 s

Alternative Verification Approaches

Outline of Lecture 7

Modelling Mutual Exclusion Algorithms

Evaluating the CCS Model

Model Checking Mutual Exclusion

Alternative Verification Approaches

Syntax of Value-Passing CCS

Semantics of Value-Passing CCS

Translation of Value-Passing into Pure CCS

Verification by Bisimulation Checking

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

Alternative Verification Approaches

Verification by Bisimulation Checking

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

Example 7.3 (Two-place buffers (cf. Example 2.5))

1. Sequential **specification**:
$$B_0 = in.B_1$$
$$B_1 = \overline{out}.B_0 + in.B_2$$
$$B_2 = \overline{out}.B_1$$
2. Parallel **implementation**:
$$B_{||} = (B[f] || B[g]) \setminus com$$
$$B = in.\overline{out}.B$$

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

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

Alternative Verification Approaches

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 exclusion in CCS

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

Alternative Verification Approaches

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 exclusion in CCS

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

Again: *Peterson* and *MutExSpec* are “weakly bisimilar”

Syntax of Value-Passing CCS

Outline of Lecture 7

Modelling Mutual Exclusion Algorithms

Evaluating the CCS Model

Model Checking Mutual Exclusion

Alternative Verification Approaches

Syntax of Value-Passing CCS

Semantics of Value-Passing CCS

Translation of Value-Passing into Pure CCS

Syntax of Value-Passing CCS

Value-Passing CCS

- **So far:** pure CCS
 - communication = mere synchronisation
 - no (explicit) exchange of data
- **But:** processes usually **do** pass around data

Syntax of Value-Passing CCS

Value-Passing CCS

- **So far:** pure CCS
 - communication = mere synchronisation
 - no (explicit) exchange of data
- **But:** processes usually **do** pass around data

⇒ Value-passing CCS

- Introduced in Robin Milner: *Communication and Concurrency*, Prentice-Hall, 1989
- Assumption (for simplicity): only **integers** as data type

Syntax of Value-Passing CCS

Value-Passing CCS

- **So far:** pure CCS
 - communication = mere synchronisation
 - no (explicit) exchange of data
- **But:** processes usually **do** pass around data

⇒ Value-passing CCS

- Introduced in Robin Milner: *Communication and Concurrency*, Prentice-Hall, 1989
- Assumption (for simplicity): only **integers** as data type

Example 7.4 (One-place buffer with data (cf. Example 2.5))

One-place buffer that outputs successor of stored value:

$$\begin{aligned} B &= in(x).B'(x) \\ B'(x) &= \overline{out}(x + 1).B \end{aligned}$$

Syntax of Value-Passing CCS

Syntax of Value-Passing CCS I

Definition 7.5 (Syntax of value-passing CCS)

- Let A, \bar{A}, Pid (ranked) as in Definition 2.1.

Syntax of Value-Passing CCS

Syntax of Value-Passing CCS I

Definition 7.5 (Syntax of value-passing CCS)

- Let A, \bar{A}, Pid (ranked) as in Definition 2.1.
- Let e and b be integer and Boolean expressions, resp., built from integer variables x, y, \dots

Syntax of Value-Passing CCS

Syntax of Value-Passing CCS I

Definition 7.5 (Syntax of value-passing CCS)

- Let A, \bar{A}, Pid (ranked) as in Definition 2.1.
- Let e and b be integer and Boolean expressions, resp., built from integer variables x, y, \dots
- The set Prc^+ of value-passing process expressions is defined by:

$P ::= \text{nil}$	(inaction)
$a(x).P$	(input prefixing)
$\bar{a}(e).P$	(output prefixing)
$\tau.P$	(τ prefixing)
$P_1 + P_2$	(choice)
$P_1 \parallel P_2$	(parallel composition)
$P \setminus L$	(restriction)
$P[f]$	(relabelling)
if b then P	(conditional)
$C(e_1, \dots, e_n)$	(process call)

where $a \in A, L \subseteq A, C \in Pid$ (of rank $n \in \mathbb{N}$), and $f : A \rightarrow A$.

Syntax of Value-Passing CCS

Syntax of Value-Passing CCS II

Definition 7.5 (Syntax of value-passing CCS; continued)

A **value-passing process definition** is an equation system of the form

$$(C_i(x_1, \dots, x_{n_i}) = P_i \mid 1 \leq i \leq k)$$

where

- $k \geq 1$,
- $C_i \in \text{Pid}$ of rank n_i (pairwise distinct),
- $P_i \in \text{Prc}^+$ (with process identifiers from $\{C_1, \dots, C_k\}$), and
- all occurrences of an integer variable y in each P_i are **bound**, i.e., $y \in \{x_1, \dots, x_{n_i}\}$ or y is in the scope of an input prefix of the form $a(y)$ (to ensure well-definedness of values).

Syntax of Value-Passing CCS

Syntax of Value-Passing CCS II

Definition 7.5 (Syntax of value-passing CCS; continued)

A **value-passing process definition** is an equation system of the form

$$(C_i(x_1, \dots, x_{n_i}) = P_i \mid 1 \leq i \leq k)$$

where

- $k \geq 1$,
- $C_i \in \text{Pid}$ of rank n_i (pairwise distinct),
- $P_i \in \text{Prc}^+$ (with process identifiers from $\{C_1, \dots, C_k\}$), and
- all occurrences of an integer variable y in each P_i are **bound**, i.e., $y \in \{x_1, \dots, x_{n_i}\}$ or y is in the scope of an input prefix of the form $a(y)$ (to ensure well-definedness of values).

Example 7.6

1. $C(x) = \bar{a}(x + 1).b(y).C(y)$ is allowed

Syntax of Value-Passing CCS

Syntax of Value-Passing CCS II

Definition 7.5 (Syntax of value-passing CCS; continued)

A **value-passing process definition** is an equation system of the form

$$(C_i(x_1, \dots, x_{n_i}) = P_i \mid 1 \leq i \leq k)$$

where

- $k \geq 1$,
- $C_i \in \text{Pid}$ of rank n_i (pairwise distinct),
- $P_i \in \text{Prc}^+$ (with process identifiers from $\{C_1, \dots, C_k\}$), and
- all occurrences of an integer variable y in each P_i are **bound**, i.e., $y \in \{x_1, \dots, x_{n_i}\}$ or y is in the scope of an input prefix of the form $a(y)$ (to ensure well-definedness of values).

Example 7.6

1. $C(x) = \bar{a}(x + 1).b(y).C(y)$ is allowed
2. $C(x) = \bar{a}(x + 1).\bar{a}(y + 2).nil$ is disallowed as y is not bound

Semantics of Value-Passing CCS

Outline of Lecture 7

Modelling Mutual Exclusion Algorithms

Evaluating the CCS Model

Model Checking Mutual Exclusion

Alternative Verification Approaches

Syntax of Value-Passing CCS

Semantics of Value-Passing CCS

Translation of Value-Passing into Pure CCS

Semantics of Value-Passing CCS

Semantics of Value-Passing CCS I

Definition 7.7 (Semantics of value-passing CCS)

A value-passing process definition $(C_i(x_1, \dots, x_{n_i}) = P_i \mid 1 \leq i \leq k)$ determines the LTS $(Prc^+, Act, \longrightarrow)$ with $Act := (A \cup \bar{A}) \times \mathbb{Z} \cup \{\tau\}$ whose transitions can be inferred from the following rules ($P, P', Q, Q' \in Prc^+$, $a \in A$, x_i integer variables, e_i/b integer/Boolean expressions, $z \in \mathbb{Z}$, $\alpha \in Act$, $\lambda \in (A \cup \bar{A}) \times \mathbb{Z}$):

$$\begin{array}{c} \text{(In)} \frac{}{a(x).P \xrightarrow{a(z)} P[z/x]} \\ \text{(Out)} \frac{(z \text{ value of } e)}{\bar{a}(e).P \xrightarrow{\bar{a}(z)} P} \\ \text{(Tau)} \frac{}{\tau.P \xrightarrow{\tau} P} \\ \text{(Sum}_1\text{)} \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \\ \text{(Sum}_2\text{)} \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'} \\ \text{(Par}_1\text{)} \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \\ \text{(Par}_2\text{)} \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'} \\ \text{(Com)} \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\bar{\lambda}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'} \end{array}$$

Semantics of Value-Passing CCS

Semantics of Value-Passing CCS II

Definition 7.7 (Semantics of value-passing CCS; continued)

$$\text{(Rel)} \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

$$\text{(Res)} \frac{P \xrightarrow{\alpha} P' \quad (\alpha \notin (L \cup \bar{L}) \times \mathbb{Z})}{P \setminus L \xrightarrow{\alpha} P' \setminus L}$$

$$\text{(If)} \frac{P \xrightarrow{\alpha} P' \quad (b \text{ true})}{\text{if } b \text{ then } P \xrightarrow{\alpha} P'}$$

$$\text{(Call)} \frac{P[z_1/x_1, \dots, z_n/x_n] \xrightarrow{\alpha} P' \quad (C(x_1, \dots, x_n) = P, z_i \text{ value of } e_i)}{C(e_1, \dots, e_n) \xrightarrow{\alpha} P'}$$

Semantics of Value-Passing CCS

Semantics of Value-Passing CCS II

Definition 7.7 (Semantics of value-passing CCS; continued)

$$\text{(Rel)} \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

$$\text{(Res)} \frac{P \xrightarrow{\alpha} P' \quad (\alpha \notin (L \cup \bar{L}) \times \mathbb{Z})}{P \setminus L \xrightarrow{\alpha} P' \setminus L}$$

$$\text{(If)} \frac{P \xrightarrow{\alpha} P' \quad (b \text{ true})}{\text{if } b \text{ then } P \xrightarrow{\alpha} P'}$$

$$\text{(Call)} \frac{P[z_1/x_1, \dots, z_n/x_n] \xrightarrow{\alpha} P' \quad (C(x_1, \dots, x_n) = P, z_i \text{ value of } e_i)}{C(e_1, \dots, e_n) \xrightarrow{\alpha} P'}$$

Remarks:

- $P[z_1/x_1, \dots, z_n/x_n]$ denotes the **substitution** of each free (i.e., unbound) occurrence of x_i by z_i ($1 \leq i \leq n$)
- **Relabelling** functions are extended to actions by letting

$$f(a(z)) := f(a)(z) \quad \text{and} \quad f(\bar{a}(z)) := \overline{f(a)}(z) \quad (\text{and } f(\tau) := \tau)$$

Semantics of Value-Passing CCS III

Further remarks:

- The binding restriction ensures that all integer and Boolean expressions have a **defined value**

Semantics of Value-Passing CCS

Semantics of Value-Passing CCS III

Further remarks:

- The binding restriction ensures that all integer and Boolean expressions have a **defined value**
- The **two-armed conditional** $\text{if } b \text{ then } P \text{ else } Q$ can be defined by

$$(\text{if } b \text{ then } P) + (\text{if } \neg b \text{ then } Q)$$

Semantics of Value-Passing CCS

Semantics of Value-Passing CCS III

Further remarks:

- The binding restriction ensures that all integer and Boolean expressions have a **defined value**
- The **two-armed conditional** $\text{if } b \text{ then } P \text{ else } Q$ can be defined by

$$(\text{if } b \text{ then } P) + (\text{if } \neg b \text{ then } Q)$$

Example 7.8

One-place buffer that outputs non-negative predecessor of stored value:

$$B = in(x).B'(x)$$
$$B'(x) = (\text{if } x = 0 \text{ then } \overline{out}(0).B) + (\text{if } x > 0 \text{ then } \overline{out}(x - 1).B)$$

(processing of value “1”: on the board)

Translation of Value-Passing into Pure CCS

Outline of Lecture 7

Modelling Mutual Exclusion Algorithms

Evaluating the CCS Model

Model Checking Mutual Exclusion

Alternative Verification Approaches

Syntax of Value-Passing CCS

Semantics of Value-Passing CCS

Translation of Value-Passing into Pure CCS

Translation of Value-Passing into Pure CCS

Translation of Value-Passing into Pure CCS I

- **To show:** value-passing process definitions can be represented in pure CCS
- **Idea:** each parametrised construct ($a(x)$, $\bar{a}(e)$, $C(e_1, \dots, e_n)$) corresponds to a **family** of constructs in pure CCS, one for each possible integer value
- Requires extension of pure CCS by **infinite** choices (“ $\sum \dots$ ”), restrictions, and process definitions

Translation of Value-Passing into Pure CCS

Translation of Value-Passing into Pure CCS II

Definition 7.9 (Translation of value-passing into pure CCS)

For each $P \in Prc^+$ without free variables, its **translated form** $\widehat{P} \in Prc$ is given by

$$\begin{array}{ll} \widehat{\text{nil}} := \text{nil} & \widehat{\tau.P} := \tau.\widehat{P} \\ \widehat{a(x).P} := \sum_{z \in \mathbb{Z}} a_z.\widehat{P[z/x]} & \widehat{\bar{a}(e).P} := \bar{a}_z.\widehat{P} \quad (z \text{ value of } e) \\ \widehat{P_1 + P_2} := \widehat{P_1} + \widehat{P_2} & \widehat{P_1 \parallel P_2} := \widehat{P_1} \parallel \widehat{P_2} \\ \widehat{P \setminus L} := \widehat{P} \setminus \{a_z \mid a \in L, z \in \mathbb{Z}\} & \widehat{P[f]} := \widehat{P}[\widehat{f}] \quad (\widehat{f}(a_z) := f(a)_z) \\ \text{if } \widehat{b} \text{ then } P := \begin{cases} \widehat{P} & \text{if } b \text{ true} \\ \text{nil} & \text{otherwise} \end{cases} & \widehat{C(e_1, \dots, e_n)} := C_{z_1, \dots, z_n} \quad (z_i \text{ value of } e_i) \end{array}$$

Translation of Value-Passing into Pure CCS

Translation of Value-Passing into Pure CCS II

Definition 7.9 (Translation of value-passing into pure CCS)

For each $P \in Proc^+$ without free variables, its **translated form** $\widehat{P} \in Proc$ is given by

$$\begin{array}{ll} \widehat{nil} := nil & \widehat{\tau.P} := \tau.\widehat{P} \\ \widehat{a(x).P} := \sum_{z \in \mathbb{Z}} a_z.\widehat{P}[z/x] & \widehat{\bar{a}(e).P} := \bar{a}_z.\widehat{P} \quad (z \text{ value of } e) \\ \widehat{P_1 + P_2} := \widehat{P_1} + \widehat{P_2} & \widehat{P_1 \parallel P_2} := \widehat{P_1} \parallel \widehat{P_2} \\ \widehat{P \setminus L} := \widehat{P} \setminus \{a_z \mid a \in L, z \in \mathbb{Z}\} & \widehat{P[f]} := \widehat{P}[\widehat{f}] \quad (\widehat{f}(a_z) := f(a)_z) \\ \text{if } \widehat{b} \text{ then } P := \begin{cases} \widehat{P} & \text{if } b \text{ true} \\ nil & \text{otherwise} \end{cases} & \widehat{C(e_1, \dots, e_n)} := C_{z_1, \dots, z_n} \quad (z_i \text{ value of } e_i) \end{array}$$

Moreover, each defining equation $C(x_1, \dots, x_n) = P$ of a process identifier is translated into the indexed collection of process definitions

$$\left(C_{z_1, \dots, z_n} = P[z_1/x_1, \dots, z_n/x_n] \mid v_1, \dots, v_n \in \mathbb{Z} \right)$$

Translation of Value-Passing into Pure CCS

Translation of Value-Passing into Pure CCS III

Example 7.10 (cf. Example 7.8)

$$B = in(x).B'(x)$$
$$B'(x) = (\text{if } x = 0 \text{ then } \overline{out}(0).B) + (\text{if } x > 0 \text{ then } \overline{out}(x - 1).B)$$

(on the board)

Translation of Value-Passing into Pure CCS

Translation of Value-Passing into Pure CCS III

Example 7.10 (cf. Example 7.8)

$$B = in(x).B'(x)$$
$$B'(x) = (\text{if } x = 0 \text{ then } \overline{out}(0).B) + (\text{if } x > 0 \text{ then } \overline{out}(x - 1).B)$$

(on the board)

Theorem 7.11 (Correctness of translation)

For all $P, P' \in \text{Prc}^+$ and $\alpha \in \text{Act}$,

$$P \xrightarrow{\alpha} P' \iff \widehat{P} \xrightarrow{\widehat{\alpha}} \widehat{P}'$$

where $\widehat{a}(z) := a_z$, $\widehat{\bar{a}}(z) := \bar{a}_z$, and $\widehat{\tau} := \tau$.

Translation of Value-Passing into Pure CCS

Translation of Value-Passing into Pure CCS III

Example 7.10 (cf. Example 7.8)

$$B = in(x).B'(x)$$
$$B'(x) = (\text{if } x = 0 \text{ then } \overline{out}(0).B) + (\text{if } x > 0 \text{ then } \overline{out}(x - 1).B)$$

(on the board)

Theorem 7.11 (Correctness of translation)

For all $P, P' \in \text{Prc}^+$ and $\alpha \in \text{Act}$,

$$P \xrightarrow{\alpha} P' \iff \widehat{P} \xrightarrow{\widehat{\alpha}} \widehat{P}'$$

where $\widehat{a}(z) := a_z$, $\widehat{\bar{a}}(z) := \bar{a}_z$, and $\widehat{\tau} := \tau$.

Proof.

by induction on the structure of P (omitted) □