Concurrency Theory

Winter Semester 2019/20

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

`https://moves.rwth-aachen.de/teaching/ws-19-20/ct/`

Software Modeling and Verification Chair | RWTH AACHEN UNIVERSITY

Outline of Lecture 7

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 critical section
  - k (in $\{1, 2\}$, arbitrary initial value) – index of prioritised process
- P_i uses local variable $j := 2 - i$ (index of other process)

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

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)

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

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

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 end;
    "critical section";
    $b_i :=$ false;
end

CCS representation

$$
\begin{aligned}
P_1 &= \overline{b1wt}.\overline{kw2}.P_{11}\\
P_{11} &= b2rf.P_{12} + \\
&\quad\; b2rt.(kr1.P_{12} + kr2.P_{11})\\
P_{12} &= enter1.exit1.\overline{b1wf}.P_1\\
P_2 &= \overline{b2wt}.\overline{kw1}.P_{21}\\
P_{21} &= b1rf.P_{22} + \\
&\quad\; b1rt.(kr1.P_{21} + kr2.P_{22})\\
P_{22} &= enter2.exit2.\overline{b2wf}.P_2\\
Peterson &= (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L\\
\text{for } L &= \{b1rf, b1rt, b1wf, b1wt,\\
&\quad\; b2rf, b2rt, b2wf, b2wt,\\
&\quad\; kr1, kr2, kw1, kw2\}
\end{aligned}
$$

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 ∧ k = j do skip end;
    "critical section";
    b_i := false;
end
```

CCS representation

$$
\begin{aligned}
P_1 &= \overline{b1wt}.\overline{kw2}.P_{11}\\
P_{11} &= b2rf.P_{12} + \\
&\quad b2rt.(kr1.P_{12} + kr2.P_{11})\\
P_{12} &= enter1.exit1.\overline{b1wf}.P_1\\
P_2 &= \overline{b2wt}.\overline{kw1}.P_{21}\\
P_{21} &= b1rf.P_{22} + \\
&\quad b1rt.(kr1.P_{21} + kr2.P_{22})\\
P_{22} &= enter2.exit2.\overline{b2wf}.P_2\\
Peterson &= (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L\\
\text{for } L &= \{b1rf, b1rt, b1wf, b1wt,\\
&\quad b2rf, b2rt, b2wf, b2wt,\\
&\quad kr1, kr2, kw1, kw2\}
\end{aligned}
$$

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm	CCS representation

while true do
"non-critical section";
$b_i :=$ true;
$k := j$;
while $b_j \wedge k = j$ do skip end;
"critical section";
$b_i :=$ false;
end
	$$\begin{aligned} P_1 &= \overline{b1wt}.\overline{kw2}.P_{11} \\ P_{11} &= b2rf.P_{12} + \\ &\ b2rt.(kr1.P_{12} + kr2.P_{11}) \\ P_{12} &= enter1.exit1.\overline{b1wf}.P_1 \\ P_2 &= \overline{b2wt}.\overline{kw1}.P_{21} \\ P_{21} &= b1rf.P_{22} + \\ &\ b1rt.(kr1.P_{21} + kr2.P_{22}) \\ P_{22} &= enter2.exit2.\overline{b2wf}.P_2 \\ Peterson &= (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L \\ \text{for } L &= \{b1rf, b1rt, b1wf, b1wt, \\ &\ b2rf, b2rt, b2wf, b2wt, \\ &\ kr1, kr2, kw1, kw2\} \end{aligned}$$

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm	CCS representation

while true do
"non-critical section";
$b_i :=$ true;
$k := j$;
while $b_j \wedge k = j$ do skip end;
"critical section";
$b_i :=$ false;
end
	$P_1 = \overline{b1wt}.\overline{kw2}.P_{11}$
	$P_{11} = b2rf.P_{12} + b2rt.(kr1.P_{12} + kr2.P_{11})$
	$P_{12} = enter1.exit1.\overline{b1wf}.P_1$
	$P_2 = \overline{b2wt}.\overline{kw1}.P_{21}$
	$P_{21} = b1rf.P_{22} + b1rt.(kr1.P_{21} + kr2.P_{22})$
	$P_{22} = enter2.exit2.\overline{b2wf}.P_2$
	$Peterson = (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L$
	for $L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm	CCS representation

while true do
"non-critical section";
$b_i :=$ true;
$k := j$;
while $b_j \wedge k = j$ do skip end;
"critical section";
$b_i :=$ false;
end
	$P_1 = \overline{b1wt}.\overline{kw2}.P_{11}$
	$P_{11} = b2rf.P_{12} + b2rt.(kr1.P_{12} + kr2.P_{11})$
	$P_{12} = enter1.exit1.\overline{b1wf}.P_1$
	$P_2 = \overline{b2wt}.\overline{kw1}.P_{21}$
	$P_{21} = b1rf.P_{22} + b1rt.(kr1.P_{21} + kr2.P_{22})$
	$P_{22} = enter2.exit2.\overline{b2wf}.P_2$
	$Peterson = (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L$
	for $L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm	CCS representation

while true do
    "non-critical section";
    $b_i :=$ true;
    $k := j$;
    while $b_j \wedge k = j$ do skip end;
    "critical section";
    $b_i :=$ false;
end
	$P_1 = \overline{b1wt}.\overline{kw2}.P_{11}$
	$P_{11} = b2rf.P_{12} + b2rt.(kr1.P_{12} + kr2.P_{11})$
	$P_{12} = enter1.exit1.\overline{b1wf}.P_1$
	$P_2 = \overline{b2wt}.\overline{kw1}.P_{21}$
	$P_{21} = b1rf.P_{22} + b1rt.(kr1.P_{21} + kr2.P_{22})$
	$P_{22} = enter2.exit2.\overline{b2wf}.P_2$
	$Peterson = (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L$
	for $L = \{b1rf, b1rt, b1wf, b1wt, b2rf, b2rt, b2wf, b2wt, kr1, kr2, kw1, kw2\}$

Software Modeling and Verification Chair

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 end;
  "critical section";
  $b_i :=$ false;
end

CCS representation

$$
\begin{aligned}
P_1 &= \overline{b1wt}.\overline{kw2}.P_{11}\\
P_{11} &= b2rf.P_{12} + \\
&\quad\; b2rt.(kr1.P_{12} + kr2.P_{11})\\
P_{12} &= enter1.exit1.\overline{b1wf}.P_1\\
P_2 &= \overline{b2wt}.\overline{kw1}.P_{21}\\
P_{21} &= b1rf.P_{22} + \\
&\quad\; b1rt.(kr1.P_{21} + kr2.P_{22})\\
P_{22} &= enter2.exit2.\overline{b2wf}.P_2\\
Peterson &= (P_1 \parallel P_2 \parallel B_{1f} \parallel B_{2f} \parallel K_1) \setminus L\\
\text{for } L &= \{b1rf, b1rt, b1wf, b1wt,\\
&\quad\; b2rf, b2rt, b2wf, b2wt,\\
&\quad\; kr1, kr2, kw1, kw2\}
\end{aligned}
$$

Outline of Lecture 7

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.
    - CCS specification of Peterson's algorithm available as example
    - yields LTS with 50 states (see following slides)
  - [e]TAPAs (Tool for the Analysis of Process Algebras): http://etapas.sourceforge.net/
    - Eclipse plug-in
    - stand-alone version not supported any more
  - CWB (Edinburgh Concurrency Workbench): http://homepages.inf.ed.ac.uk/perdita/cwb/
    - somewhat outdated

Obtaining the LTS II

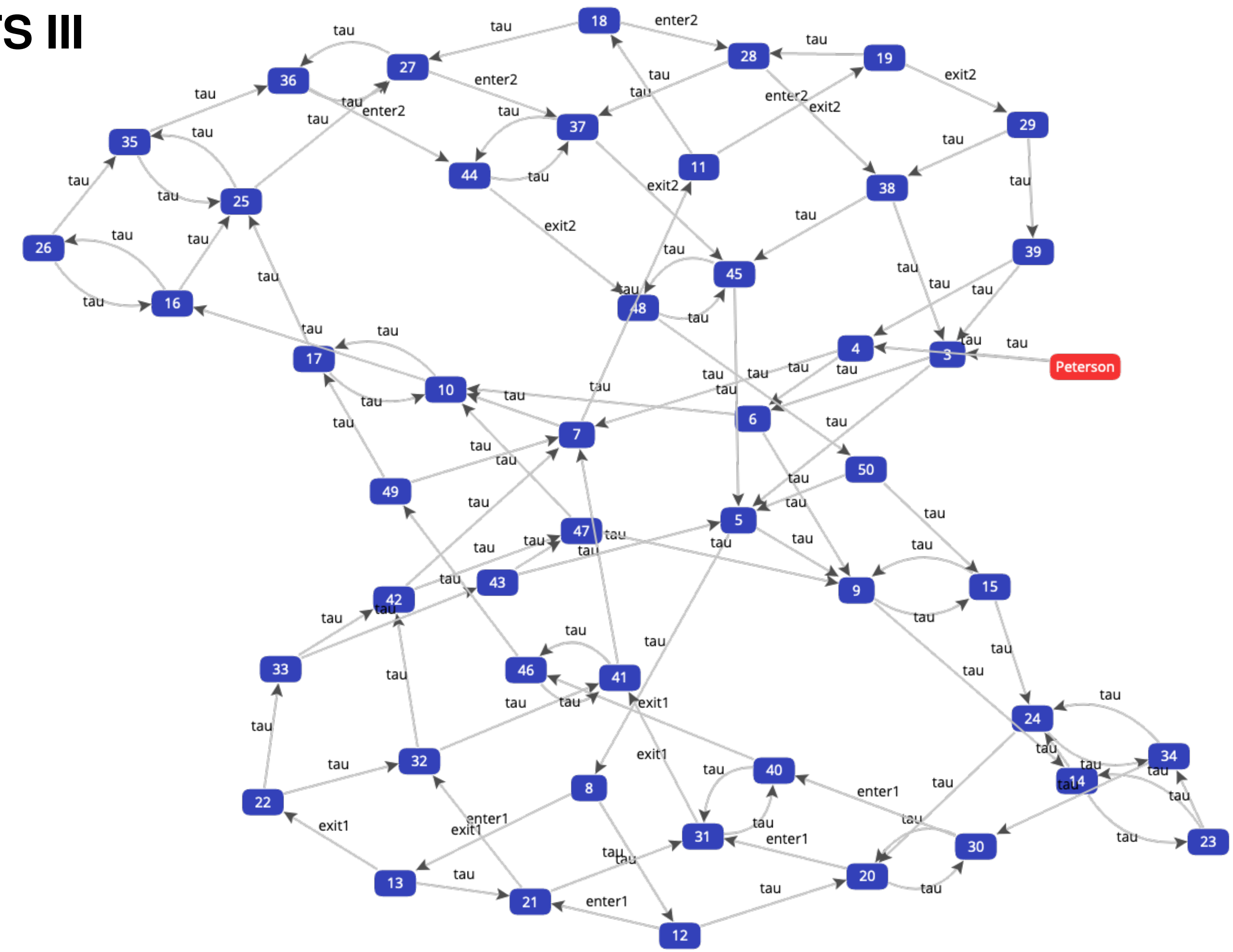
CAAL | Project | Edit | Explore | Verify | Games | About | Syntax

Peterson's Algorithm

Parse | CCS | TCCS | 16

```
* Peterson's algorithm for mutual exclusion.
* See Chapter 7 of "Reactive Systems" for a full description.

B1f = 'b1rf.B1f + b1wf.B1f + b1wt.B1t;
B1t = 'b1rt.B1t + b1wf.B1f + b1wt.B1t;

B2f = 'b2rf.B2f + b2wf.B2f + b2wt.B2t;
B2t = 'b2rt.B2t + b2wf.B2f + b2wt.B2t;

K1 = 'kr1.K1 + kw1.K1 + kw2.K2;
K2 = 'kr2.K2 + kw1.K1 + kw2.K2;

P1 = 'b1wt.'kw2.P11;
P11 = b2rf.P12 + b2rt.(kr2.P11 + kr1.P12);
P12 = enter1.exit1.'b1wf.P1;

P2 = 'b2wt.'kw1.P21;
P21 = b1rf.P22 + b1rt.(kr1.P21 + kr2.P22);
P22 = enter2.exit2.'b2wf.P2;

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

Spec = enter1.exit1.Spec + enter2.exit2.Spec;
```

Obtaining the LTS III



Outline of Lecture 7

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?

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

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.

Equivalently:
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.

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

$$
\begin{aligned}
MutEx &:= Inv(F) \\
Inv(F) &\stackrel{max}{=} F \wedge [Act]Inv(F) \qquad \text{(cf. Theorem 6.1)} \\
F &:= [exit1]\mathrm{ff} \vee [exit2]\mathrm{ff}
\end{aligned}
$$

Model Checking Mutual Exclusion

- Using CAAL Tool
- Supports property specifications by recursive HML formulae:

```
MutEx max= ([[exit1]]ff or [[exit2]]ff) and [-]MutEx;
```

Outline of Lecture 7

Alternative Verification Approaches

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

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

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

$$\mathit{MutExSpec} = \mathit{enter1}.\mathit{exit1}.\mathit{MutExSpec} + \mathit{enter2}.\mathit{exit2}.\mathit{MutExSpec}$$

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

$$\mathit{MutExSpec} = \mathit{enter1}.\mathit{exit1}.\mathit{MutExSpec} + \mathit{enter2}.\mathit{exit2}.\mathit{MutExSpec}$$

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

Outline of Lecture 7

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

$\Rightarrow$ 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

$\Rightarrow$ 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:

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

Syntax of Value-Passing CCS I

Definition 7.5 (Syntax of value-passing CCS)

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

Syntax of Value-Passing CCS I

Definition 7.5 (Syntax of value-passing CCS)

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

Syntax of Value-Passing CCS I

Definition 7.5 (Syntax of value-passing CCS)

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

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

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

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

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

where

- $k \geq 1$,
- $C_i \in Pid$ of rank n_i (pairwise distinct),
- $P_i \in Prc^+$ (with process identifiers from $\{C_1, \ldots, C_k\}$), and
- all occurrences of an integer variable y in each P_i are bound, i.e., $y \in \{x_1, \ldots, x_{n_i}\}$ 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 II

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

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

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

where

- $k \geq 1$,
- $C_i \in Pid$ of rank n_i (pairwise distinct),
- $P_i \in Prc^+$ (with process identifiers from $\{C_1, \ldots, C_k\}$), and
- all occurrences of an integer variable y in each P_i are bound, i.e., $y \in \{x_1, \ldots, x_{n_i}\}$ 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. $C(x) = \overline{a}(x+1).b(y).C(y)$ is allowed

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

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

where

- $k \geq 1$,
- $C_i \in Pid$ of rank n_i (pairwise distinct),
- $P_i \in Prc^+$ (with process identifiers from $\{C_1, \ldots, C_k\}$), and
- all occurrences of an integer variable y in each P_i are bound, i.e., $y \in \{x_1, \ldots, x_{n_i}\}$ 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. $C(x) = \overline{a}(x + 1).b(y).C(y)$ is allowed
2. $C(x) = \overline{a}(x + 1).\overline{a}(y + 2).\mathrm{nil}$ is disallowed as y is not bound

Outline of Lecture 7

Semantics of Value-Passing CCS I

Definition 7.7 (Semantics of value-passing CCS)

A value-passing process definition $(C_i(x_1, \ldots, x_{n_i}) = P_i \mid 1 \leq i \leq k)$ determines the LTS $(Prc^+, Act, \longrightarrow)$ with $Act := (A \cup \overline{A}) \times \mathbb{Z} \cup \{\tau\}$ 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, $z \in \mathbb{Z}$, $\alpha \in Act$, $\lambda \in (A \cup \overline{A}) \times \mathbb{Z}$):

$$\text{(In)}\ \frac{}{a(x).P \xrightarrow{a(z)} P[z/x]} \qquad \text{(Out)}\ \frac{(z \text{ value of } e)}{\overline{a}(e).P \xrightarrow{\overline{a}(z)} P} \qquad \text{(Tau)}\ \frac{}{\tau.P \xrightarrow{\tau} P}$$

$$\text{(Sum}_1\text{)}\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \text{(Sum}_2\text{)}\ \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

$$\text{(Par}_1\text{)}\ \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \qquad \text{(Par}_2\text{)}\ \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'} \qquad \text{(Com)}\ \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

Semantics of Value-Passing CCS II

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

$$\text{(Rel)}\ \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]} \qquad \text{(Res)}\ \frac{P \xrightarrow{\alpha} P' \ \ (\alpha \notin (L \cup \overline{L}) \times \mathbb{Z})}{P \setminus L \xrightarrow{\alpha} P' \setminus L}$$

$$\text{(If)}\ \frac{P \xrightarrow{\alpha} P' \ \ (b \text{ true})}{\text{if } b \text{ then } P \xrightarrow{\alpha} P'} \qquad \text{(Call)}\ \frac{\begin{array}{c} P[z_1/x_1, \ldots, z_n/x_n] \xrightarrow{\alpha} P' \\ (C(x_1, \ldots, x_n) = P, z_i \text{ value of } e_i) \end{array}}{C(e_1, \ldots, e_n) \xrightarrow{\alpha} P'}$$

Semantics of Value-Passing CCS II

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

$$\text{(Rel)}\ \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]} \qquad \text{(Res)}\ \frac{P \xrightarrow{\alpha} P' \quad (\alpha \notin (L \cup \overline{L}) \times \mathbb{Z})}{P \setminus L \xrightarrow{\alpha} P' \setminus L}$$

$$\text{(If)}\ \frac{P \xrightarrow{\alpha} P' \quad (b \text{ true})}{\text{if } b \text{ then } P \xrightarrow{\alpha} P'} \qquad \text{(Call)}\ \frac{\begin{array}{c} P[z_1/x_1, \ldots, z_n/x_n] \xrightarrow{\alpha} P' \\ (C(x_1, \ldots, x_n) = P, z_i \text{ value of } e_i) \end{array}}{C(e_1, \ldots, e_n) \xrightarrow{\alpha} P'}$$

Remarks:

- $P[z_1/x_1, \ldots, z_n/x_n]$ denotes the substitution of each free (i.e., unbound) occurrence of x_i by z_i $(1 \leq i \leq n)$
- Operations on actions ignore values:

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

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 III

Further remarks:

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

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

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 if b then P 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:

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

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

Outline of Lecture 7

Translation of Value-Passing into Pure CCS I

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

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

$$\widehat{\mathrm{nil}} := \mathrm{nil}$$

$$\widehat{a(x).P} := \sum_{z \in \mathbb{Z}} a_z.\widehat{P[z/x]}$$

$$\widehat{P_1 + P_2} := \widehat{P}_1 + \widehat{P}_2$$

$$\widehat{P \setminus L} := \widehat{P} \setminus \{a_z \mid a \in L, z \in \mathbb{Z}\}$$

$$\widehat{\text{if } b \text{ then } P} := \begin{cases} \widehat{P} & \text{if } b \text{ true} \\ \mathrm{nil} & \text{otherwise} \end{cases}$$

$$\widehat{\tau.P} := \tau.\widehat{P}$$

$$\widehat{\overline{a}(e).P} := \overline{a_z}.\widehat{P} \quad (z \text{ value of } e)$$

$$\widehat{P_1 \parallel P_2} := \widehat{P}_1 \parallel \widehat{P}_2$$

$$\widehat{P[f]} := \widehat{P}[\widehat{f}] \quad (\widehat{f}(a_z) := f(a)_z)$$

$$C(\widehat{e_1, \ldots, e_n}) := C_{z_1,\ldots,z_n} \quad (z_i \text{ value of } e_i)$$

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

$$\widehat{\text{nil}} := \text{nil} \qquad \widehat{\tau.P} := \tau.\widehat{P}$$

$$\widehat{a(x).P} := \sum_{z \in \mathbb{Z}} a_z.\widehat{P[z/x]} \qquad \widehat{\overline{a}(e).P} := \overline{a_z}.\widehat{P} \quad (z \text{ value of } e)$$

$$\widehat{P_1 + P_2} := \widehat{P}_1 + \widehat{P}_2 \qquad \widehat{P_1 \parallel P_2} := \widehat{P}_1 \parallel \widehat{P}_2$$

$$\widehat{P \setminus L} := \widehat{P} \setminus \{a_z \mid a \in L, z \in \mathbb{Z}\} \qquad \widehat{P[f]} := \widehat{P}[\widehat{f}] \quad (\widehat{f}(a_z) := f(a)_z)$$

$$\widehat{\text{if } b \text{ then } P} := \begin{cases} \widehat{P} & \text{if } b \text{ true} \\ \text{nil} & \text{otherwise} \end{cases} \qquad \widehat{C(e_1, \ldots, e_n)} := C_{z_1,\ldots,z_n} \quad (z_i \text{ value of } e_i)$$

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

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

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 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 $P, P' \in Prc^+$ and $\alpha \in Act$,

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

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

Translation of Value-Passing into Pure CCS III

Example 7.10 (cf. Example 7.8)

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

(on the board)

Theorem 7.11 (Correctness of translation)

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

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

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

Proof.

by induction on the structure of P (omitted) □