Concurrency Theory

Lecture 7: Mutually Recursive Equational Systems

Joost-Pieter Katoen   Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH AACHEN UNIVERSITY

{katoen,noll}@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/ct13/

Winter Semester 2013/14

Outline

Application to HML with Recursion

Lemma

Let $(S, Act, \longrightarrow)$ *be an LTS and* $F \in HMF_X$. *Then*

1. $[\![F]\!] : 2^S \to 2^S$ *is monotonic w.r.t.* $(2^S, \subseteq)$
2. $\mathrm{fix}([\![F]\!]) = \bigcap\{T \subseteq S \mid [\![F]\!](T) \subseteq T\}$
3. $\mathrm{FIX}([\![F]\!]) = \bigcup\{T \subseteq S \mid T \subseteq [\![F]\!](T)\}$

If, in addition, S *is finite, then*

4. $\mathrm{fix}([\![F]\!]) = [\![F]\!]^m(\emptyset)$ *for some* $m \in \mathbb{N}$
5. $\mathrm{FIX}([\![F]\!]) = [\![F]\!]^M(S)$ *for some* $M \in \mathbb{N}$

Proof.

1. by induction on the structure of F (details omitted)
2. by Lemma 5.9 and Theorem 5.14
3. by Lemma 5.9 and Theorem 5.14
4. by Lemma 5.9 and Theorem 6.1
5. by Lemma 5.9 and Theorem 6.1

$\square$

Outline

Introducing Several Variables

Sometimes useful: using more than one variable

Example 7.1

"It is always the case that a process can perform an a-labelled transition leading to a state where b-transitions can be executed forever."

Sometimes useful: using more than one variable

Example 7.1

"It is always the case that a process can perform an a-labelled transition leading to a state where b-transitions can be executed forever."

can be specified by

$$Inv(\langle a \rangle Forever(b))$$

where

$$\begin{aligned} Inv(F) &\overset{max}{=} F \wedge [Act]F \qquad \text{(cf. Theorem 6.5)} \\ Forever(b) &\overset{max}{=} \langle b \rangle Forever(b) \end{aligned}$$

Definition 7.2 (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 7.3 (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

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

where

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

Lemma 7.4

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* $(T_1, \ldots, T_n) \sqsubseteq (T_1', \ldots, T_n')$ *if* $T_i \subseteq T_i'$ *for every* $1 \leq i \leq n$.

Lemma 7.4

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* $(T_1, \ldots, T_n) \sqsubseteq (T'_1, \ldots, T'_n)$ *if* $T_i \subseteq T'_i$ *for every* $1 \leq i \leq n$.

1. $(D, \sqsubseteq)$ *is a complete lattice with*

$$\bigsqcup\{(T_1^i, \ldots, T_n^i) \mid i \in I\} = \left(\bigcup\{T_1^i \mid i \in I\}, \ldots, \bigcup\{T_n^i \mid i \in I\}\right)$$
$$\bigsqcap\{(T_1^i, \ldots, T_n^i) \mid i \in I\} = \left(\bigcap\{T_1^i \mid i \in I\}, \ldots, \bigcap\{T_n^i \mid i \in I\}\right)$$

Lemma 7.4

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* $(T_1, \ldots, T_n) \sqsubseteq (T'_1, \ldots, T'_n)$ *if* $T_i \subseteq T'_i$ *for every* $1 \leq i \leq n$*.*

1. $(D, \sqsubseteq)$ *is a complete lattice with*

$$\bigsqcup\{(T_1^i, \ldots, T_n^i) \mid i \in I\} = \left(\bigcup\{T_1^i \mid i \in I\}, \ldots, \bigcup\{T_n^i \mid i \in I\}\right)$$
$$\bigsqcap\{(T_1^i, \ldots, T_n^i) \mid i \in I\} = \left(\bigcap\{T_1^i \mid i \in I\}, \ldots, \bigcap\{T_n^i \mid i \in I\}\right)$$

2. $[\![E]\!]$ *is monotonic w.r.t.* $(D, \sqsubseteq)$

Semantics of Recursive Equational Systems II

Lemma 7.4

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* $(T_1, \ldots, T_n) \sqsubseteq (T'_1, \ldots, T'_n)$ *if* $T_i \subseteq T'_i$ *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. $\mathrm{fix}([\![E]\!]) = [\![E]\!]^m(\emptyset, \ldots, \emptyset)$ *for some* $m \in \mathbb{N}$
4. $\mathrm{FIX}([\![E]\!]) = [\![E]\!]^M(S, \ldots, S)$ *for some* $M \in \mathbb{N}$

Proof.

omitted □

Example 7.5

$$\begin{tikzcd} s \arrow[r, bend left, "a"] & s_1 \arrow[l, bend left, "b"] & s_2 \arrow[l, "a"] \arrow[r, "a"] & s_3 \arrow[loop right, "b"] \end{tikzcd}$$

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

Outline

1. Recap: Fixed-Point Theory for HML
2. Mutually Recursive Equational Systems
3. Mixing Least and Greatest Fixed Points
4. Modelling Mutual Exclusion Algorithms

Mixing Least and Greatest Fixed Points I

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

Mixing Least and Greatest Fixed Points I

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

Example 7.6

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

Mixing Least and Greatest Fixed Points I

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

Example 7.6

"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

$$\begin{aligned} Pos(F) &\overset{min}{=} F \vee \langle Act \rangle Pos(F) \qquad \text{(cf. Example \textbf{??})} \\ Livelock &\overset{max}{=} \langle \tau \rangle Livelock \end{aligned}$$

(thus, $Livelock \equiv Forever(\tau)$ [cf. Example 7.1])

Mixing Least and Greatest Fixed Points II

Caveat: arbitrary mixing can entail non-monotonic behaviour

Mixing Least and Greatest Fixed Points II

Caveat: arbitrary mixing can entail non-monotonic behaviour

Example 7.7

$$\begin{aligned} E : X &\overset{min}{=} Y \\ Y &\overset{max}{=} X \end{aligned}$$

Mixing Least and Greatest Fixed Points II

Caveat: arbitrary mixing can entail non-monotonic behaviour

Example 7.7

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

Fixed-point iteration:

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

Mixing Least and Greatest Fixed Points II

Caveat: arbitrary mixing can entail non-monotonic behaviour

Example 7.7

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

Fixed-point iteration:

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

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

Mixing Least and Greatest Fixed Points II

Caveat: arbitrary mixing can entail non-monotonic behaviour

Example 7.7

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

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

Example 7.8 (cf. Example 7.6)

$$\mathit{PosLL} \overset{min}{=} \mathit{Livelock} \vee \langle \mathit{Act} \rangle \mathit{PosLL}$$
$$\mathit{Livelock} \overset{max}{=} \langle \tau \rangle \mathit{Livelock}$$

$$\begin{tikzcd} s \arrow[r, "a"] & p \arrow[r, "\tau"] \arrow[loop below, "\tau"] & q \arrow[r, "\tau"] & r \end{tikzcd}$$

Example 7.8 (cf. Example 7.6)

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

$$
\begin{tikzcd}
s \arrow[r, "a"] & p \arrow[r, "\tau"] \arrow[loop below, "\tau"] & q \arrow[r, "\tau"] & r
\end{tikzcd}
$$

1. Fixed-point iteration for $Livelock : T \mapsto \langle \cdot\tau\cdot \rangle(T)$:

$$S = \{s, p, q, r\}$$

Example 7.8 (cf. Example 7.6)

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

$$\begin{tikzcd} s \arrow[r, "a"] & p \arrow[r, "\tau"] \arrow[loop below, "\tau"] & q \arrow[r, "\tau"] & r \end{tikzcd}$$

1. Fixed-point iteration for $Livelock : T \mapsto \langle \cdot \tau \cdot \rangle (T)$:

$$S = \{s, p, q, r\} \mapsto \{p, q\}$$

Mixing Least and Greatest Fixed Points III

Example 7.8 (cf. Example 7.6)

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

$$
\begin{tikzcd}
s \arrow[r, "a"] & p \arrow[r, "\tau"] \arrow[loop below, "\tau"] & q \arrow[r, "\tau"] & r
\end{tikzcd}
$$

1. Fixed-point iteration for $Livelock : T \mapsto \langle \cdot \tau \cdot \rangle (T)$:

$$S = \{s, p, q, r\} \mapsto \{p, q\} \mapsto \{p\}$$

Mixing Least and Greatest Fixed Points III

Example 7.8 (cf. Example 7.6)

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

$$
\begin{tikzcd}
s \arrow[r, "a"] & p \arrow[r, "\tau"] \arrow[loop below, "\tau"] & q \arrow[r, "\tau"] & r
\end{tikzcd}
$$

1. Fixed-point iteration for $Livelock : T \mapsto \langle \cdot \tau \cdot \rangle (T)$:

$$S = \{s, p, q, r\} \mapsto \{p, q\} \mapsto \{p\} \mapsto \{p\}$$

Example 7.8 (cf. Example 7.6)

$$PosLL \overset{min}{=} Livelock \vee \langle Act \rangle PosLL$$
$$Livelock \overset{max}{=} \langle \tau \rangle Livelock$$

$$s \xrightarrow{a} p \xrightarrow{\tau} q \xrightarrow{\tau} r \qquad p \xrightarrow{\tau} p$$

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 $PosLL : T \mapsto \{p\} \cup \langle \cdot Act \cdot \rangle (T)$:

$$\emptyset$$

Mixing Least and Greatest Fixed Points III

Example 7.8 (cf. Example 7.6)

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

$$\begin{tikzcd} s \arrow[r, "a"] & p \arrow[r, "\tau"] \arrow[loop below, "\tau"] & q \arrow[r, "\tau"] & r \end{tikzcd}$$

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 $PosLL : T \mapsto \{p\} \cup \langle \cdot Act \cdot \rangle (T)$:

$$\emptyset \mapsto \{p\}$$

Mixing Least and Greatest Fixed Points III

Example 7.8 (cf. Example 7.6)

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

$$\begin{tikzcd} s \arrow[r, "a"] & p \arrow[r, "\tau"] \arrow[loop below, "\tau"] & q \arrow[r, "\tau"] & r \end{tikzcd}$$

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 $PosLL : T \mapsto \{p\} \cup \langle \cdot Act \cdot \rangle (T)$:

$$\emptyset \mapsto \{p\} \mapsto \{s, p\}$$

Example 7.8 (cf. Example 7.6)

$$PosLL \overset{min}{=} Livelock \vee \langle Act \rangle PosLL$$
$$Livelock \overset{max}{=} \langle \tau \rangle Livelock$$

$$s \xrightarrow{\ a\ } p \xrightarrow{\ \tau\ } q \xrightarrow{\ \tau\ } r \qquad p \overset{\tau}{\circlearrowleft}$$

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 $PosLL : T \mapsto \{p\} \cup \langle \cdot Act \cdot \rangle (T)$:

$$\emptyset \mapsto \{p\} \mapsto \{s, p\} \mapsto \{s, p\}$$

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

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

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, ...)

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

Outline

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)

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 7.9 (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

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 7.10 (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 7.10 (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:

$$\begin{aligned} B_{1f} &= \overline{b1rf}.B_{1f} + b1wf.B_{1f} + b1wt.B_{1t} \\ B_{1t} &= \overline{b1rt}.B_{1t} + b1wf.B_{1f} + b1wt.B_{1t} \end{aligned}$$

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 7.10 (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:

$$\begin{aligned} B_{1f} &= \overline{b1rf}.B_{1f} + b1wf.B_{1f} + b1wt.B_{1t} \\ B_{1t} &= \overline{b1rt}.B_{1t} + b1wf.B_{1f} + b1wt.B_{1t} \end{aligned}$$

- Similarly for b_2 and k:

$$\begin{aligned} B_{2f} &= \overline{b2rf}.B_{2f} + b2wf.B_{2f} + b2wt.B_{2t} \\ B_{2t} &= \overline{b2rt}.B_{2t} + b2wf.B_{2f} + b2wt.B_{2t} \end{aligned}$$

$$\begin{aligned} K_1 &= \overline{kr1}.K_1 + kw1.K_1 + kw2.K_2 \\ K_2 &= \overline{kr2}.K_2 + kw1.K_1 + kw2.K_2 \end{aligned}$$

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

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

$$
\begin{array}{l}
\texttt{while true do} \\
\quad \textit{“non-critical section”}; \\
\quad b_i := \texttt{true}; \\
\quad k := j; \\
\quad \texttt{while } b_j \wedge k = j \texttt{ do skip}; \\
\quad \textit{“critical section”}; \\
\quad b_i := \texttt{false}; \\
\texttt{end}
\end{array}
$$

CCS representation

$$
\begin{aligned}
P_1 &= \overline{b1wt}.\overline{kw2}.P_{11} \\
P_{11} &= b2rf.P_{12} + \\
&\quad\; b2rt.(kr1.P_{12} + kr2.P_{11}) \\
P_{12} &= enter_1.exit_1.\overline{b1wf}.P_1 \\
P_2 &= \overline{b2wt}.\overline{kw1}.P_{21} \\
P_{21} &= b1rf.P_{22} + \\
&\quad\; b1rt.(kr1.P_{21} + kr2.P_{22}) \\
P_{22} &= enter_2.exit_2.\overline{b2wf}.P_2
\end{aligned}
$$

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

where

$$
\begin{aligned}
L = \{&b1rf, b1rt, b1wf, b1wt, \\
&b2rf, b2rt, b2wf, b2wt, \\
&kr1, kr2, kw1, kw2\}
\end{aligned}
$$

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

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

CCS representation

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

where

$$
\begin{aligned}
L = \{&b1rf, b1rt, b1wf, b1wt,\\
&b2rf, b2rt, b2wf, b2wt,\\
&kr1, kr2, kw1, kw2\}
\end{aligned}
$$

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

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

CCS representation

$$
\begin{aligned}
P_1 &= \overline{b1wt}.\overline{kw2}.P_{11}\\
P_{11} &= b2rf.P_{12} + \\
&\quad\ b2rt.(kr1.P_{12} + kr2.P_{11})\\
P_{12} &= enter_1.exit_1.\overline{b1wf}.P_1\\
P_2 &= \overline{b2wt}.\overline{kw1}.P_{21}\\
P_{21} &= b1rf.P_{22} + \\
&\quad\ b1rt.(kr1.P_{21} + kr2.P_{22})\\
P_{22} &= enter_2.exit_2.\overline{b2wf}.P_2
\end{aligned}
$$

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

where

$$
\begin{aligned}
L = \{&b1rf, b1rt, b1wf, b1wt,\\
&b2rf, b2rt, b2wf, b2wt,\\
&kr1, kr2, kw1, kw2\}
\end{aligned}
$$

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

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

CCS representation

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

where

$$\begin{aligned}
L = \{&b1rf, b1rt, b1wf, b1wt,\\
&b2rf, b2rt, b2wf, b2wt,\\
&kr1, kr2, kw1, kw2\}
\end{aligned}$$

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

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

$$
\begin{aligned}
P_1 &= \overline{b1wt}.\overline{kw2}.P_{11} \\
P_{11} &= b2rf.P_{12} + \\
&\quad\; b2rt.(kr1.P_{12} + kr2.P_{11}) \\
P_{12} &= enter_1.exit_1.\overline{b1wf}.P_1 \\
P_2 &= \overline{b2wt}.\overline{kw1}.P_{21} \\
P_{21} &= b1rf.P_{22} + \\
&\quad\; b1rt.(kr1.P_{21} + kr2.P_{22}) \\
P_{22} &= enter_2.exit_2.\overline{b2wf}.P_2
\end{aligned}
$$

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

where

$$
\begin{aligned}
L = \{&b1rf, b1rt, b1wf, b1wt, \\
&b2rf, b2rt, b2wf, b2wt, \\
&kr1, kr2, kw1, kw2\}
\end{aligned}
$$

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

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

CCS representation

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

where

$$\begin{aligned}
L = \{&b1rf, b1rt, b1wf, b1wt,\\
&b2rf, b2rt, b2wf, b2wt,\\
&kr1, kr2, kw1, kw2\}
\end{aligned}$$

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

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

CCS representation

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