

Concurrency Theory

Lecture 7: Mutually Recursive Equational Systems

Joost-Pieter Katoen Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)



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

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

Winter Semester 2013/14

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

Lemma

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

- ① $\llbracket F \rrbracket : 2^S \rightarrow 2^S$ is monotonic w.r.t. $(2^S, \subseteq)$
- ② $\text{fix}(\llbracket F \rrbracket) = \bigcap \{T \subseteq S \mid \llbracket F \rrbracket(T) \subseteq T\}$
- ③ $\text{FIX}(\llbracket F \rrbracket) = \bigcup \{T \subseteq S \mid T \subseteq \llbracket F \rrbracket(T)\}$

If, in addition, S is finite, then

- ④ $\text{fix}(\llbracket F \rrbracket) = \llbracket F \rrbracket^m(\emptyset)$ for some $m \in \mathbb{N}$
- ⑤ $\text{FIX}(\llbracket F \rrbracket) = \llbracket F \rrbracket^M(S)$ 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



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

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

$$\text{Inv}(\langle a \rangle \text{Forever}(b))$$

where

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

Syntax of Mutually Recursive Equational Systems

Definition 7.2 (Syntax of mutually recursive equational systems)

Let $\mathcal{X} = \{X_1, \dots, X_n\}$ be a set of **variables**. The set $HMF_{\mathcal{X}}$ of **Hennessey-Milner formulae over $\mathcal{X}$** is defined by the following syntax:

$F ::= X_i$	(variable)
tt	(true)
ff	(false)
$F_1 \wedge F_2$	(conjunction)
$F_1 \vee F_2$	(disjunction)
$\langle \alpha \rangle F$	(diamond)
$[\alpha] F$	(box)

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 ,

$$\llbracket E \rrbracket : (2^S)^n \rightarrow (2^S)^n,$$

is defined by

$$\llbracket E \rrbracket (T_1, \dots, T_n) := (\llbracket F_{X_1} \rrbracket (T_1, \dots, T_n), \dots, \llbracket F_{X_n} \rrbracket (T_1, \dots, T_n))$$

where

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

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

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

$$\begin{aligned}\bigsqcup \{(T_1^i, \dots, T_n^i) \mid i \in I\} &= (\bigcup \{T_1^i \mid i \in I\}, \dots, \bigcup \{T_n^i \mid i \in I\}) \\ \bigsqcap \{(T_1^i, \dots, T_n^i) \mid i \in I\} &= (\bigcap \{T_1^i \mid i \in I\}, \dots, \bigcap \{T_n^i \mid i \in I\})\end{aligned}$$

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

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

$$\begin{aligned}\bigsqcup \{(T_1^i, \dots, T_n^i) \mid i \in I\} &= (\bigcup \{T_1^i \mid i \in I\}, \dots, \bigcup \{T_n^i \mid i \in I\}) \\ \bigsqcap \{(T_1^i, \dots, T_n^i) \mid i \in I\} &= (\bigcap \{T_1^i \mid i \in I\}, \dots, \bigcap \{T_n^i \mid i \in I\})\end{aligned}$$

② $\llbracket E \rrbracket$ is monotonic w.r.t. $(D, \sqsubseteq)$

Lemma 7.4

Let $(S, \text{Act}, \longrightarrow)$ 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 $(T_1, \dots, T_n) \sqsubseteq (T'_1, \dots, T'_n)$ if $T_i \subseteq T'_i$ for every $1 \leq i \leq n$.

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

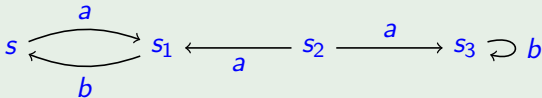
$$\begin{aligned}\bigsqcup \{(T_1^i, \dots, T_n^i) \mid i \in I\} &= (\bigcup \{T_1^i \mid i \in I\}, \dots, \bigcup \{T_n^i \mid i \in I\}) \\ \bigsqcap \{(T_1^i, \dots, T_n^i) \mid i \in I\} &= (\bigcap \{T_1^i \mid i \in I\}, \dots, \bigcap \{T_n^i \mid i \in I\})\end{aligned}$$

- ② $\llbracket E \rrbracket$ is monotonic w.r.t. $(D, \sqsubseteq)$
③ $\text{fix}(\llbracket E \rrbracket) = \llbracket E \rrbracket^m(\emptyset, \dots, \emptyset)$ for some $m \in \mathbb{N}$
④ $\text{FIX}(\llbracket E \rrbracket) = \llbracket E \rrbracket^M(S, \dots, S)$ for some $M \in \mathbb{N}$

Proof.

omitted □

Example 7.5



Let $S := \{s, s_1, s_2, s_3\}$ and E given by

$$\begin{aligned} X &\stackrel{\text{max}}{=} \langle a \rangle Y \wedge [a] Y \wedge [b] \text{ff} \\ Y &\stackrel{\text{max}}{=} \langle b \rangle X \wedge [b] X \wedge [a] \text{ff} \end{aligned}$$

Computation of $\text{FIX}(\llbracket E \rrbracket)$: on the board

- 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

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

(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

$$\begin{array}{l} E : X \stackrel{\min}{=} Y \\ \quad Y \stackrel{\max}{=} X \end{array}$$

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:

$$(\perp, \top) = (\emptyset, S) \xrightarrow{\llbracket E \rrbracket} (S, \emptyset) \xrightarrow{\llbracket E \rrbracket} (\emptyset, S) \xrightarrow{\llbracket E \rrbracket} \dots$$

Mixing Least and Greatest Fixed Points II

Caveat: arbitrary mixing can entail **non-monotonic behaviour**

Example 7.7

$$\begin{array}{l} E : X \stackrel{\min}{=} Y \\ \quad Y \stackrel{\max}{=} X \end{array}$$

Fixed-point iteration:

$$(\perp, \top) = (\emptyset, S) \xrightarrow{\llbracket E \rrbracket} (S, \emptyset) \xrightarrow{\llbracket E \rrbracket} (\emptyset, S) \xrightarrow{\llbracket E \rrbracket} \dots$$

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

$$\begin{array}{l} E : X \stackrel{\min}{=} Y \\ \quad Y \stackrel{\max}{=} X \end{array}$$

Fixed-point iteration:

$$(\perp, \top) = (\emptyset, S) \xrightarrow{\llbracket E \rrbracket} (S, \emptyset) \xrightarrow{\llbracket E \rrbracket} (\emptyset, S) \xrightarrow{\llbracket E \rrbracket} \dots$$

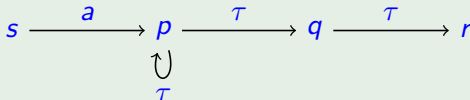
Solution: **nesting** of specifications by partitioning equations into a sequence of blocks such that all equations in one block

- are of **same type** (either *min* or *max*) and
- use only variables defined in **the same or subsequent blocks**

$\Rightarrow$ **bottom-up, block-wise evaluation** by fixed-point iteration

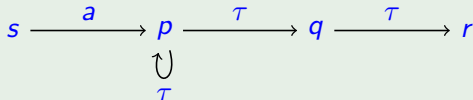
Example 7.8 (cf. Example 7.6)

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



Example 7.8 (cf. Example 7.6)

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

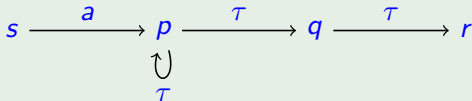


- ① Fixed-point iteration for $Livelock : T \mapsto \langle \cdot \tau \cdot \rangle (T)$:

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

Example 7.8 (cf. Example 7.6)

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

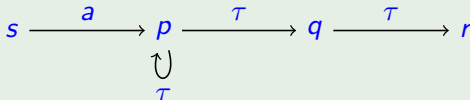


- ① Fixed-point iteration for $Livelock : T \mapsto \langle \cdot \tau \cdot \rangle (T)$:

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

Example 7.8 (cf. Example 7.6)

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

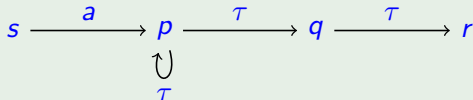


- ① Fixed-point iteration for $Livelock : T \mapsto \langle \cdot \tau \cdot \rangle (T)$:

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

Example 7.8 (cf. Example 7.6)

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

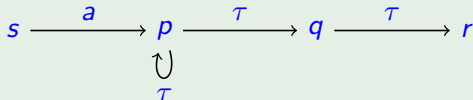


- ① Fixed-point iteration for $Livelock : T \mapsto \langle \cdot \tau \cdot \rangle (T)$:

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

Example 7.8 (cf. Example 7.6)

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



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

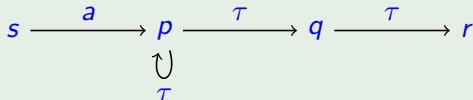
$$S = \{s, p, q, r\} \mapsto \{p, q\} \mapsto \{p\} \mapsto \{p\}$$

- ② Fixed-point iteration for $\text{PosLL} : T \mapsto \{p\} \cup \langle \cdot \text{Act} \cdot \rangle (T)$:

$$\emptyset$$

Example 7.8 (cf. Example 7.6)

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



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

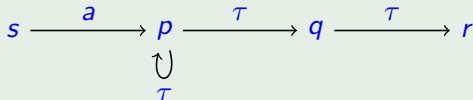
$$S = \{s, p, q, r\} \mapsto \{p, q\} \mapsto \{p\} \mapsto \{p\}$$

- ② Fixed-point iteration for $\text{PosLL} : T \mapsto \{p\} \cup \langle \cdot \text{Act} \cdot \rangle (T)$:

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

Example 7.8 (cf. Example 7.6)

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



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

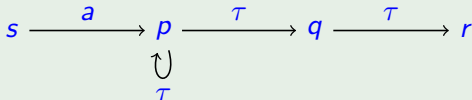
$$S = \{s, p, q, r\} \mapsto \{p, q\} \mapsto \{p\} \mapsto \{p\}$$

- 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)

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



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

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

- 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

- 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

- 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 $\text{NP} \cap \text{co-NP}$; linear for HML with one variable)
- Generally **undecidable** for **infinite** LTSs and HML with one variable
(CCS, Petri nets, ...)

- 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 $\text{NP} \cap \text{co-NP}$; 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

- 1 Recap: Fixed-Point Theory for HML
- 2 Mutually Recursive Equational Systems
- 3 Mixing Least and Greatest Fixed Points
- 4 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**)
 - 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";
   $b_i := \text{true}$ ;
   $k := j$ ;
  while  $b_j \wedge k = j$  do skip;
  "critical section";
   $b_i := \text{false}$ ;
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 :

$$\begin{aligned} B_{2f} &= \overline{b2rf}.B_{2f} + b2wf.B_{2f} + b2wt.B_{2t} \\ B_{2t} &= \overline{b2rt}.B_{2t} + b2wf.B_{2f} + b2wt.B_{2t} \\ K_1 &= \overline{kr1}.K_1 + kw1.K_1 + kw2.K_2 \\ K_2 &= \overline{kr2}.K_2 + kw1.K_1 + kw2.K_2 \end{aligned}$$

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 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

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

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

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

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

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

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

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

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

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

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

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

Modelling the Processes in CCS

Assumption: P_i cannot fail or terminate within critical section

Peterson's algorithm

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

CCS representation

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