



Concurrency Theory

Winter Semester 2017/18

Lecture 8: The π -Calculus

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

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

Recap: Modelling Mutual Exclusion Algorithms

Outline of Lecture 8

Recap: Modelling Mutual Exclusion Algorithms

Recap: Value-Passing CCS

Modelling Mobile Concurrent Systems

Another Example: Mobile Clients

Syntax of the Monadic π -Calculus

Semantics of the Monadic π -Calculus

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

Recap: Modelling Mutual Exclusion Algorithms

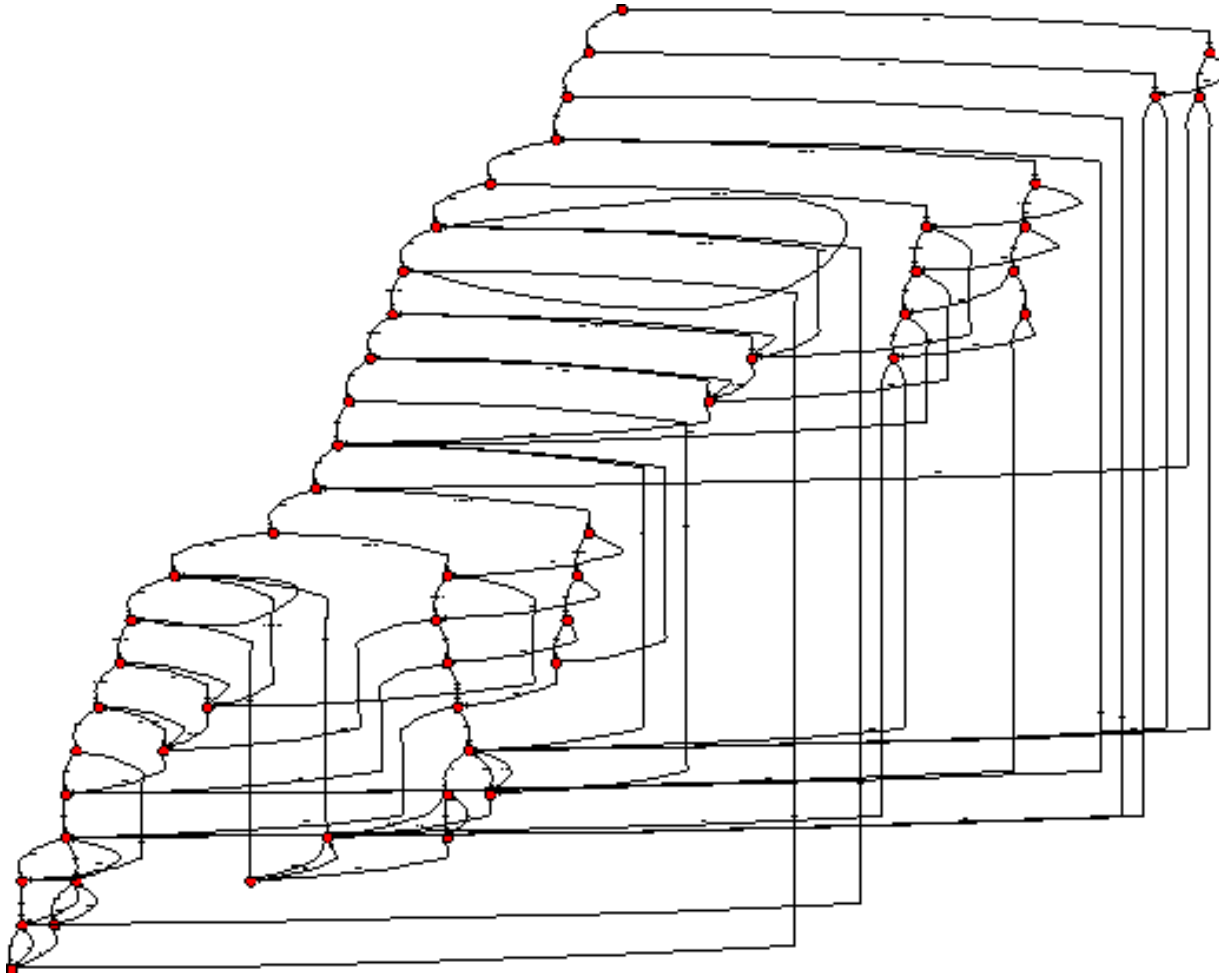
Obtaining the LTS I

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 **54** states (see next slide)
 - **CWB** (Edinburgh Concurrency Workbench):
<http://homepages.inf.ed.ac.uk/perdita/cwb/>
 - somewhat outdated

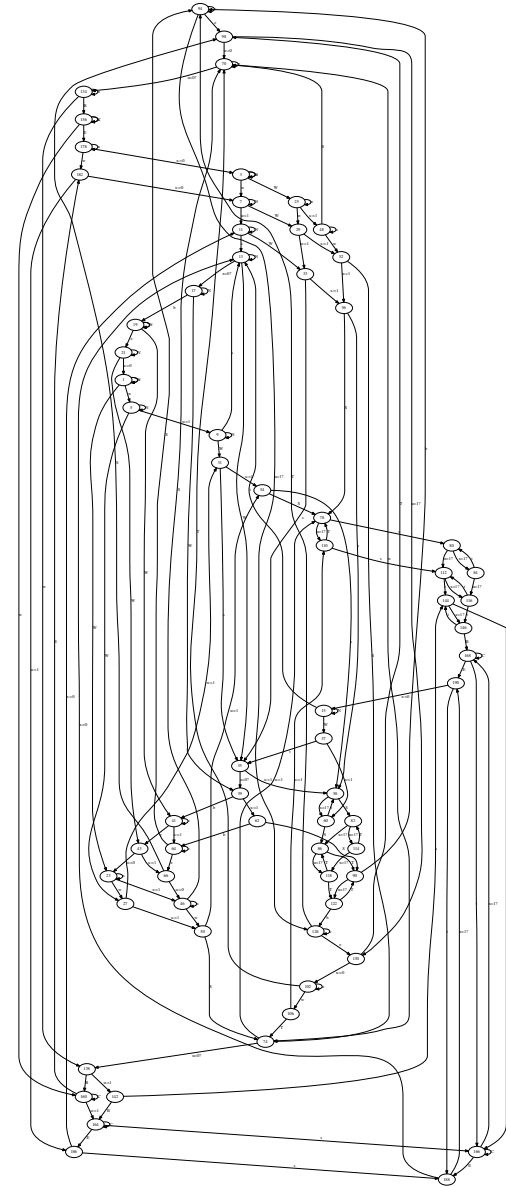
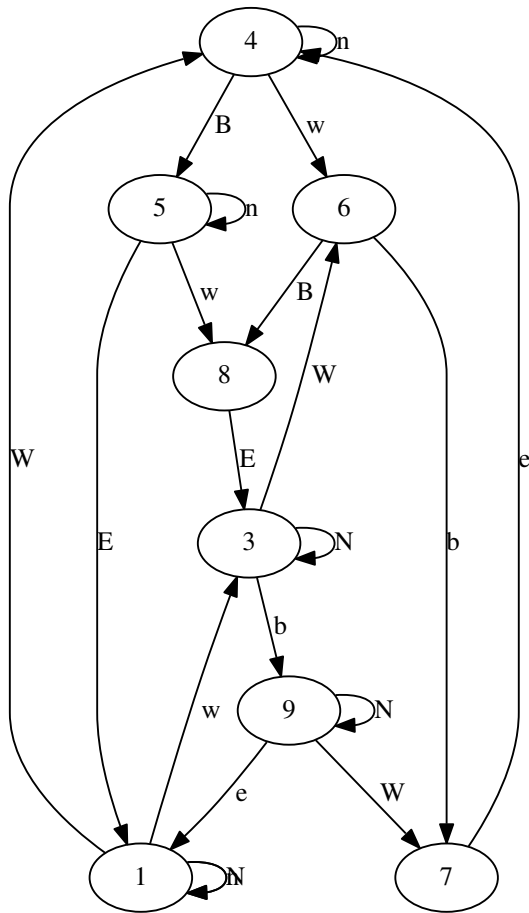
Recap: Modelling Mutual Exclusion Algorithms

Obtaining the LTS II



Was passiert, wenn P_0 und P_1 laufen?

Betrachte $(P_0 \sqcup P_1) \circ B_U \circ B_X \circ B_S$.



Recap: Value-Passing CCS

Outline of Lecture 8

Recap: Modelling Mutual Exclusion Algorithms

Recap: Value-Passing CCS

Modelling Mobile Concurrent Systems

Another Example: Mobile Clients

Syntax of the Monadic π -Calculus

Semantics of the Monadic π -Calculus

Recap: Value-Passing CCS

Syntax of Value-Passing CCS

Definition (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 $f : A \rightarrow A$.

Recap: Value-Passing CCS

Semantics of Value-Passing CCS I

Definition (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, $z \in \mathbb{Z}$, $\alpha \in Act$, $\lambda \in (A \cup \bar{A}) \times \mathbb{Z}$):

$$\text{(In)} \frac{}{a(x).P \xrightarrow{a(z)} P[z/x]}$$

$$\text{(Out)} \frac{(z \text{ value of } e)}{\bar{a}(e).P \xrightarrow{\bar{a}(z)} P}$$

$$\text{(Tau)} \frac{}{\tau.P \xrightarrow{\tau} P}$$

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

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

$$\text{(Par}_1\text{)} \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}$$

$$\text{(Par}_2\text{)} \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'}$$

$$\text{(Com)} \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\bar{\lambda}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

Recap: Value-Passing CCS

Semantics of Value-Passing CCS II

Definition (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

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

Recap: Value-Passing CCS

Translation of Value-Passing into Pure CCS

Definition (Translation of value-passing into pure CCS)

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

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

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

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

Modelling Mobile Concurrent Systems

Outline of Lecture 8

Recap: Modelling Mutual Exclusion Algorithms

Recap: Value-Passing CCS

Modelling Mobile Concurrent Systems

Another Example: Mobile Clients

Syntax of the Monadic π -Calculus

Semantics of the Monadic π -Calculus

Mobility in Concurrent Systems I

Observation: CCS imposes **static communication structures**: if $P, Q \in Proc$ want to communicate, then both must syntactically refer to the same action name

Mobility in Concurrent Systems I

Observation: CCS imposes **static communication structures**: if $P, Q \in Proc$ want to communicate, then both must syntactically refer to the same action name

⇒ every potential communication partner known beforehand,
no dynamic passing of communication links

Mobility in Concurrent Systems I

Observation: CCS imposes **static communication structures**: if $P, Q \in Proc$ want to communicate, then both must syntactically refer to the same action name

- ⇒ every potential communication partner known beforehand,
no dynamic passing of communication links
- ⇒ lack of modelling capabilities for **mobility**

Mobility in Concurrent Systems I

Observation: CCS imposes **static communication structures**: if $P, Q \in \text{Proc}$ want to communicate, then both must syntactically refer to the same action name

⇒ every potential communication partner known beforehand,
no dynamic passing of communication links

⇒ lack of modelling capabilities for **mobility**

Goal: develop calculus in the spirit of CCS which supports mobility

⇒ π -Calculus

Mobility in Concurrent Systems II

Example 8.1 (Dynamic access to resources)

- Server S controls access to printer P
- Client C wishes to use P

Mobility in Concurrent Systems II

Example 8.1 (Dynamic access to resources)

- Server S controls access to printer P
- Client C wishes to use P
- In CCS: P and C must share some action name a
⇒ C could access P without being granted it by S

Mobility in Concurrent Systems II

Example 8.1 (Dynamic access to resources)

- Server S controls access to printer P
- Client C wishes to use P
- In CCS: P and C must share some action name a
 $\Rightarrow C$ could access P without being granted it by S
- In π -Calculus:
 - initially only S has access to P (using link a)
 - using another link b , C can request access to P

Mobility in Concurrent Systems II

Example 8.1 (Dynamic access to resources)

- Server S controls access to printer P
- Client C wishes to use P
- In CCS: P and C must share some action name a
 $\Rightarrow C$ could access P without being granted it by S
- In π -Calculus:
 - initially only S has access to P (using link a)
 - using another link b , C can request access to P
- Formally:

$$\underbrace{\bar{b}\langle a \rangle . S'}_S \parallel \underbrace{b(c) . \bar{c}\langle d \rangle . C'}_C \parallel \underbrace{a(e) . P'}_P$$

- a : link to P
- b : link between S and C
- c : “placeholder” for a
- d : data to be printed
- e : “placeholder” for d

Mobility in Concurrent Systems II

Example 8.1 (Dynamic access to resources)

- Server S controls access to printer P
- Client C wishes to use P
- In CCS: P and C must share some action name a
 $\Rightarrow C$ could access P without being granted it by S
- In π -Calculus:
 - initially only S has access to P (using link a)
 - using another link b , C can request access to P
- Formally:

$$\begin{array}{c} \underbrace{\bar{b}\langle a \rangle . S'}_S \parallel \underbrace{b(c) . \bar{c}\langle d \rangle . C'}_C \parallel \underbrace{a(e) . P'}_P \\ \xrightarrow{\tau} S' \parallel \bar{a}\langle d \rangle . C'[a/c] \parallel a(e) . P' \end{array}$$

- a : link to P
- b : link between S and C
- c : “placeholder” for a
- d : data to be printed
- e : “placeholder” for d

Mobility in Concurrent Systems II

Example 8.1 (Dynamic access to resources)

- Server S controls access to printer P
- Client C wishes to use P
- In CCS: P and C must share some action name a
 $\Rightarrow C$ could access P without being granted it by S
- In π -Calculus:
 - initially only S has access to P (using link a)
 - using another link b , C can request access to P
- Formally:

$$\begin{aligned} & \underbrace{\bar{b}\langle a \rangle . S'}_S \parallel \underbrace{b(c) . \bar{c}\langle d \rangle . C'}_C \parallel \underbrace{a(e) . P'}_P \\ \xrightarrow{\tau} & S' \parallel \bar{a}\langle d \rangle . C'[a/c] \parallel a(e) . P' \\ \xrightarrow{\tau} & S' \parallel C'[a/c] \parallel P'[d/e] \end{aligned}$$

- a : link to P
- b : link between S and C
- c : “placeholder” for a
- d : data to be printed
- e : “placeholder” for d

Mobility in Concurrent Systems III

Example 8.1 (Dynamic access to resources; continued)

- Different rôles of action name a :
 - in interaction between S and C : **object transferred** from S to C
 - in interaction between C and P : name of **communication link**

Mobility in Concurrent Systems III

Example 8.1 (Dynamic access to resources; continued)

- Different rôles of action name a :
 - in interaction between S and C : object transferred from S to C
 - in interaction between C and P : name of communication link
- Intuitively, names represent access rights:
 - a : to P
 - b : to S
 - d : to data to be printed

Mobility in Concurrent Systems III

Example 8.1 (Dynamic access to resources; continued)

- Different rôles of action name a :
 - in interaction between S and C : object transferred from S to C
 - in interaction between C and P : name of communication link
- Intuitively, names represent access rights:
 - a : to P
 - b : to S
 - d : to data to be printed
- If a is only way to access P
 $\Rightarrow P$ “moves” from S to C

Another Example: Mobile Clients

Outline of Lecture 8

Recap: Modelling Mutual Exclusion Algorithms

Recap: Value-Passing CCS

Modelling Mobile Concurrent Systems

Another Example: Mobile Clients

Syntax of the Monadic π -Calculus

Semantics of the Monadic π -Calculus

Another Example: Mobile Clients

Mobile Clients I

Example 8.2 (Hand-over protocol)

Scenario:

- **client devices** moving around (phones, PCs, sensors, ...)
- each radio-connected to some **base station**
- stations wired to **central control**
- some event (e.g., signal fading) may cause a client to be **switched** to another station
- essential: specification of switching process (“**hand-over protocol**”)

Another Example: Mobile Clients

Mobile Clients I

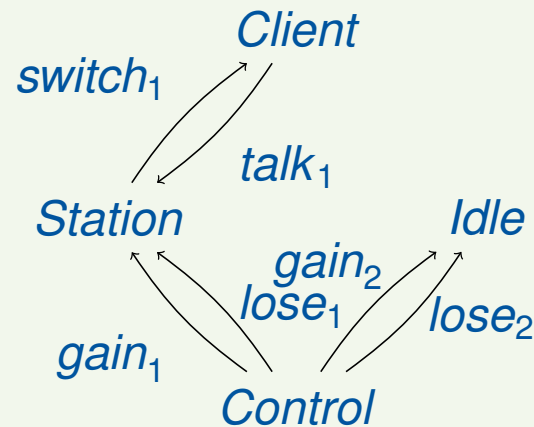
Example 8.2 (Hand-over protocol)

Scenario:

- **client devices** moving around (phones, PCs, sensors, ...)
- each radio-connected to some **base station**
- stations wired to **central control**
- some event (e.g., signal fading) may cause a client to be **switched** to another station
- essential: specification of switching process (“**hand-over protocol**”)

Simplest configuration:

two stations, one client



Another Example: Mobile Clients

Mobile Clients II

Example 8.2 (Hand-over protocol; continued)

- Every station is in one of two **modes**: *Station* (active; four links) or *Idle* (inactive; two links)

Another Example: Mobile Clients

Mobile Clients II

Example 8.2 (Hand-over protocol; continued)

- Every station is in one of two **modes**: *Station* (active; four links) or *Idle* (inactive; two links)
- *Client* can **talk** via *Station*, and at any time *Control* can request *Station/Idle* to **lose/gain** *Client*:

$$\begin{aligned} \text{Station}(\text{talk}, \text{switch}, \text{gain}, \text{lose}) &= \text{talk}.\text{Station}(\text{talk}, \text{switch}, \text{gain}, \text{lose}) + \\ &\quad \text{lose}(t, s).\overline{\text{switch}}\langle t, s \rangle.\text{Idle}(\text{gain}, \text{lose}) \\ \text{Idle}(\text{gain}, \text{lose}) &= \text{gain}(t, s).\text{Station}(t, s, \text{gain}, \text{lose}) \end{aligned}$$

Another Example: Mobile Clients

Mobile Clients II

Example 8.2 (Hand-over protocol; continued)

- Every station is in one of two **modes**: *Station* (active; four links) or *Idle* (inactive; two links)
- *Client* can **talk** via *Station*, and at any time *Control* can request *Station/Idle* to **lose/gain** *Client*:

$$\begin{aligned} \text{Station}(\text{talk}, \text{switch}, \text{gain}, \text{lose}) &= \text{talk}.\text{Station}(\text{talk}, \text{switch}, \text{gain}, \text{lose}) + \\ &\quad \text{lose}(t, s).\overline{\text{switch}}\langle t, s \rangle.\text{Idle}(\text{gain}, \text{lose}) \\ \text{Idle}(\text{gain}, \text{lose}) &= \text{gain}(t, s).\text{Station}(t, s, \text{gain}, \text{lose}) \end{aligned}$$

- If *Control* decides *Station* to lose *Client*, it issues a **new pair of channels** to be shared by *Client* and *Idle*:

$$\begin{aligned} \text{Control}_1 &= \overline{\text{lose}}_1\langle \text{talk}_2, \text{switch}_2 \rangle.\overline{\text{gain}}_2\langle \text{talk}_2, \text{switch}_2 \rangle.\text{Control}_2 \\ \text{Control}_2 &= \overline{\text{lose}}_2\langle \text{talk}_1, \text{switch}_1 \rangle.\overline{\text{gain}}_1\langle \text{talk}_1, \text{switch}_1 \rangle.\text{Control}_1 \end{aligned}$$

Another Example: Mobile Clients

Mobile Clients II

Example 8.2 (Hand-over protocol; continued)

- Every station is in one of two **modes**: *Station* (active; four links) or *Idle* (inactive; two links)
- *Client* can **talk** via *Station*, and at any time *Control* can request *Station/Idle* to **lose/gain** *Client*:

$$\begin{aligned} \text{Station}(\text{talk}, \text{switch}, \text{gain}, \text{lose}) &= \text{talk}.\text{Station}(\text{talk}, \text{switch}, \text{gain}, \text{lose}) + \\ &\quad \text{lose}(t, s).\overline{\text{switch}}\langle t, s \rangle.\text{Idle}(\text{gain}, \text{lose}) \\ \text{Idle}(\text{gain}, \text{lose}) &= \text{gain}(t, s).\text{Station}(t, s, \text{gain}, \text{lose}) \end{aligned}$$

- If *Control* decides *Station* to lose *Client*, it issues a **new pair of channels** to be shared by *Client* and *Idle*:

$$\begin{aligned} \text{Control}_1 &= \overline{\text{lose}}_1\langle \text{talk}_2, \text{switch}_2 \rangle.\overline{\text{gain}}_2\langle \text{talk}_2, \text{switch}_2 \rangle.\text{Control}_2 \\ \text{Control}_2 &= \overline{\text{lose}}_2\langle \text{talk}_1, \text{switch}_1 \rangle.\overline{\text{gain}}_1\langle \text{talk}_1, \text{switch}_1 \rangle.\text{Control}_1 \end{aligned}$$

- *Client* can either **talk** or, if requested, **switch** to a new pair of channels:

$$\text{Client}(\text{talk}, \text{switch}) = \overline{\text{talk}}.\text{Client}(\text{talk}, \text{switch}) + \text{switch}(t, s).\text{Client}(t, s)$$

Another Example: Mobile Clients

Mobile Clients III

Example 8.2 (Hand-over protocol; continued)

- As usual, the whole system is a **restricted composition** of processes:

$$System_1 = new L (Client_1 \parallel Station_1 \parallel Idle_2 \parallel Control_1)$$

where

$$Client_i := Client(talk_i, switch_i)$$

$$Station_i := Station(talk_i, switch_i, gain_i, lose_i)$$

$$Idle_i := Idle(gain_i, lose_i)$$

$$L := (talk_i, switch_i, gain_i, lose_i \mid i \in \{1, 2\})$$

Another Example: Mobile Clients

Mobile Clients III

Example 8.2 (Hand-over protocol; continued)

- As usual, the whole system is a **restricted composition** of processes:

$$System_1 = \text{new } L (Client_1 \parallel Station_1 \parallel Idle_2 \parallel Control_1)$$

where

$$Client_i := Client(talk_i, switch_i)$$

$$Station_i := Station(talk_i, switch_i, gain_i, lose_i)$$

$$Idle_i := Idle(gain_i, lose_i)$$

$$L := (talk_i, switch_i, gain_i, lose_i \mid i \in \{1, 2\})$$

- After having formally defined the π -Calculus we will see that this protocol is **correct**, i.e., that the hand-over does indeed occur:

$$System_1 \longrightarrow^* System_2$$

where

$$System_2 = \text{new } L (Idle_1 \parallel Client_2 \parallel Station_2 \parallel Control_2)$$

Syntax of the Monadic π -Calculus

Outline of Lecture 8

Recap: Modelling Mutual Exclusion Algorithms

Recap: Value-Passing CCS

Modelling Mobile Concurrent Systems

Another Example: Mobile Clients

Syntax of the Monadic π -Calculus

Semantics of the Monadic π -Calculus

Syntax of the Monadic π -Calculus

Introduction

Literature on π -Calculus:

- Initial research paper:
R. Milner, J. Parrow, D. Walker: *A calculus of mobile processes*, Part I/II. *Journal of Inf. & Comp.*, 100:1–77, 1992
- Overview article:
J. Parrow: *An introduction to the π -Calculus*. Chapter 8 of *Handbook of Process Algebra*, 479–543, Elsevier, 2001
- Textbook:
R. Milner: *Communicating and mobile systems: the π -Calculus*. Cambridge University Press, 1999

Syntax of the Monadic π -Calculus

Introduction

Literature on π -Calculus:

- Initial research paper:
R. Milner, J. Parrow, D. Walker: *A calculus of mobile processes*, Part I/II. Journal of Inf. & Comp., 100:1–77, 1992
- Overview article:
J. Parrow: *An introduction to the π -Calculus*. Chapter 8 of *Handbook of Process Algebra*, 479–543, Elsevier, 2001
- Textbook:
R. Milner: *Communicating and mobile systems: the π -Calculus*. Cambridge University Press, 1999

To simplify the presentation (as in Milner's book):

1. **Monadic π -Calculus with replication** (message = one name, no process identifiers)
2. Extension to **polyadic** calculus
3. Extension by **process equations**

Syntax of the Monadic π -Calculus

Syntax of the Monadic π -Calculus

Definition 8.3 (Syntax of monadic π -Calculus)

- Let $A = \{a, b, c, \dots, x, y, z, \dots\}$ be a set of **names**.

Syntax of the Monadic π -Calculus

Syntax of the Monadic π -Calculus

Definition 8.3 (Syntax of monadic π -Calculus)

- Let $A = \{a, b, c, \dots, x, y, z, \dots\}$ be a set of **names**.
- The set of **action prefixes** is given by

$$\begin{array}{l|l} \pi ::= x(y) & \text{(receive } y \text{ along } x) \\ & | \bar{x}\langle y \rangle & \text{(send } y \text{ along } x) \\ & | \tau & \text{(unobservable action)} \end{array}$$

Syntax of the Monadic π -Calculus

Syntax of the Monadic π -Calculus

Definition 8.3 (Syntax of monadic π -Calculus)

- Let $A = \{a, b, c, \dots, x, y, z, \dots\}$ be a set of **names**.
- The set of **action prefixes** is given by

$$\begin{array}{l|l} \pi ::= x(y) & \text{(receive } y \text{ along } x) \\ \quad | \bar{x}(y) & \text{(send } y \text{ along } x) \\ \quad | \tau & \text{(unobservable action)} \end{array}$$

- The set Proc^π of **π -Calculus process expressions** is defined by the following syntax:

$$\begin{array}{l|l} P ::= \sum_{i \in I} \pi_i.P_i & \text{(guarded sum)} \\ \quad | P_1 \parallel P_2 & \text{(parallel composition)} \\ \quad | \text{new } x P & \text{(restriction)} \\ \quad | !P & \text{(replication)} \end{array}$$

(where I finite index set, $x \in A$)

Syntax of the Monadic π -Calculus

Syntax of the Monadic π -Calculus

Definition 8.3 (Syntax of monadic π -Calculus)

- Let $A = \{a, b, c, \dots, x, y, z, \dots\}$ be a set of **names**.
- The set of **action prefixes** is given by

$$\begin{array}{l|l} \pi ::= x(y) & \text{(receive } y \text{ along } x) \\ \quad | \bar{x}(y) & \text{(send } y \text{ along } x) \\ \quad | \tau & \text{(unobservable action)} \end{array}$$

- The set Proc^π of **π -Calculus process expressions** is defined by the following syntax:

$$\begin{array}{l|l} P ::= \sum_{i \in I} \pi_i.P_i & \text{(guarded sum)} \\ \quad | P_1 \parallel P_2 & \text{(parallel composition)} \\ \quad | \text{new } x P & \text{(restriction)} \\ \quad | !P & \text{(replication)} \end{array}$$

(where I finite index set, $x \in A$)

Conventions: $\text{nil} := \sum_{i \in \emptyset} \pi_i.P_i$, $\text{new } x_1, \dots, x_n P := \text{new } x_1 (\dots \text{new } x_n P)$

Syntax of the Monadic π -Calculus

Free and Bound Names

Definition 8.4 (Free and bound names)

- The input prefix $x(y)$ and the restriction $\text{new } y P$ both **bind** y .

Syntax of the Monadic π -Calculus

Free and Bound Names

Definition 8.4 (Free and bound names)

- The input prefix $x(y)$ and the restriction $\text{new } y P$ both **bind** y .
- Every other occurrence of a name (i.e., x in $x(y)$ and x, y in $\bar{x}\langle y \rangle$) is **free**.

Free and Bound Names

Definition 8.4 (Free and bound names)

- The input prefix $x(y)$ and the restriction $\text{new } y P$ both **bind** y .
- Every other occurrence of a name (i.e., x in $x(y)$ and x, y in $\bar{x}\langle y \rangle$) is **free**.
- The set of bound/free names of a process expressions $P \in \text{Prc}^\pi$ is respectively denoted by $bn(P)/fn(P)$.

Syntax of the Monadic π -Calculus

Free and Bound Names

Definition 8.4 (Free and bound names)

- The input prefix $x(y)$ and the restriction $\text{new } y P$ both **bind** y .
- Every other occurrence of a name (i.e., x in $x(y)$ and x, y in $\bar{x}\langle y \rangle$) is **free**.
- The set of bound/free names of a process expressions $P \in \text{Prc}^\pi$ is respectively denoted by $bn(P)/fn(P)$.

Remark: $bn(P) \cap fn(P) \neq \emptyset$ is possible

Syntax of the Monadic π -Calculus

Free and Bound Names

Definition 8.4 (Free and bound names)

- The input prefix $x(y)$ and the restriction $\text{new } y P$ both **bind** y .
- Every other occurrence of a name (i.e., x in $x(y)$ and x, y in $\bar{x}\langle y \rangle$) is **free**.
- The set of bound/free names of a process expressions $P \in \text{Prc}^\pi$ is respectively denoted by $bn(P)/fn(P)$.

Remark: $bn(P) \cap fn(P) \neq \emptyset$ is possible

Example 8.5

For $P = \text{new } x (x(y).\text{nil} \parallel \bar{z}\langle y \rangle.\text{nil})$:

$$bn(P) = \{x, y\}, fn(P) = \{y, z\}$$

Semantics of the Monadic π -Calculus

Outline of Lecture 8

Recap: Modelling Mutual Exclusion Algorithms

Recap: Value-Passing CCS

Modelling Mobile Concurrent Systems

Another Example: Mobile Clients

Syntax of the Monadic π -Calculus

Semantics of the Monadic π -Calculus

Structural Congruence

Goal: simplify definition of operational semantics by ignoring “purely syntactic” differences between processes

Structural Congruence

Goal: simplify definition of operational semantics by ignoring “purely syntactic” differences between processes

Definition 8.6 (Structural congruence)

$P, Q \in \text{Proc}^\pi$ are **structurally congruent**, written $P \equiv Q$, if one can be transformed into the other by applying the following operations and equations:

1. renaming of bound names (α -conversion)

Structural Congruence

Goal: simplify definition of operational semantics by ignoring “purely syntactic” differences between processes

Definition 8.6 (Structural congruence)

$P, Q \in \text{Proc}^\pi$ are **structurally congruent**, written $P \equiv Q$, if one can be transformed into the other by applying the following operations and equations:

1. renaming of bound names (α -conversion)
2. reordering of terms in a summation (commutativity of $+$)

Structural Congruence

Goal: simplify definition of operational semantics by ignoring “purely syntactic” differences between processes

Definition 8.6 (Structural congruence)

$P, Q \in \text{Prc}^\pi$ are **structurally congruent**, written $P \equiv Q$, if one can be transformed into the other by applying the following operations and equations:

1. renaming of bound names (α -conversion)
2. reordering of terms in a summation (commutativity of $+$)
3. $P \parallel Q \equiv Q \parallel P$, $P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$, $P \parallel \text{nil} \equiv P$ (Abelian monoid laws for \parallel)

Structural Congruence

Goal: simplify definition of operational semantics by ignoring “purely syntactic” differences between processes

Definition 8.6 (Structural congruence)

$P, Q \in \text{Prc}^\pi$ are **structurally congruent**, written $P \equiv Q$, if one can be transformed into the other by applying the following operations and equations:

1. renaming of bound names (α -conversion)
2. reordering of terms in a summation (commutativity of $+$)
3. $P \parallel Q \equiv Q \parallel P$, $P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$, $P \parallel \text{nil} \equiv P$ (Abelian monoid laws for \parallel)
4. $\text{new } x \text{ nil} \equiv \text{nil}$, $\text{new } x, y P \equiv \text{new } y, x P$,
 $P \parallel \text{new } x Q \equiv \text{new } x (P \parallel Q)$ if $x \notin \text{fn}(P)$ (scope extension)

Structural Congruence

Goal: simplify definition of operational semantics by ignoring “purely syntactic” differences between processes

Definition 8.6 (Structural congruence)

$P, Q \in \text{Prc}^\pi$ are **structurally congruent**, written $P \equiv Q$, if one can be transformed into the other by applying the following operations and equations:

1. renaming of bound names (α -conversion)
2. reordering of terms in a summation (commutativity of $+$)
3. $P \parallel Q \equiv Q \parallel P$, $P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$, $P \parallel \text{nil} \equiv P$ (Abelian monoid laws for \parallel)
4. $\text{new } x \text{ nil} \equiv \text{nil}$, $\text{new } x, y P \equiv \text{new } y, x P$,
 $P \parallel \text{new } x Q \equiv \text{new } x (P \parallel Q)$ if $x \notin \text{fn}(P)$ (scope extension)
5. $!P \equiv P \parallel !P$ (unfolding)

Semantics of the Monadic π -Calculus

A Standard Form

Theorem 8.7 (Standard form)

Every process expression is structurally congruent to a process of the *standard form*

$$\text{new } x_1, \dots, x_k (P_1 \parallel \dots \parallel P_m \parallel !Q_1 \parallel \dots \parallel !Q_n)$$

where each P_i is a non-empty sum, and each Q_j is in standard form.

(If $m = n = 0$: nil; if $k = 0$: restriction absent)

Semantics of the Monadic π -Calculus

A Standard Form

Theorem 8.7 (Standard form)

Every process expression is structurally congruent to a process of the *standard form*

$$\text{new } x_1, \dots, x_k (P_1 \parallel \dots \parallel P_m \parallel !Q_1 \parallel \dots \parallel !Q_n)$$

where each P_i is a non-empty sum, and each Q_j is in standard form.

(If $m = n = 0$: nil; if $k = 0$: restriction absent)

Proof.

by induction on the structure of $R \in \text{Proc}^\pi$ (on the board) □

Semantics of the Monadic π -Calculus

The Reaction Relation

Thanks to Theorem 8.7, only processes in standard form need to be considered for defining the operational semantics:

Definition 8.8

The **reaction relation** $\longrightarrow \subseteq \text{Proc}^\pi \times \text{Proc}^\pi$ is generated by the rules:

$$\begin{array}{c} \text{(Tau)} \frac{}{\tau.P + Q \longrightarrow P} \\ \\ \text{(React)} \frac{}{(x(y).P + R) \parallel (\bar{x}\langle z \rangle.Q + S) \longrightarrow P[z/y] \parallel Q} \\ \\ \text{(Par)} \frac{P \longrightarrow P'}{P \parallel Q \longrightarrow P' \parallel Q} \qquad \text{(Res)} \frac{P \longrightarrow P'}{\text{new } x P \longrightarrow \text{new } x P'} \\ \\ \text{(Struct)} \frac{P \longrightarrow P'}{Q \longrightarrow Q'} \quad \text{if } P \equiv Q \text{ and } P' \equiv Q' \end{array}$$

- $P[z/y]$ replaces every free occurrence of y in P by z .
- In (React), the pair $(x(y), \bar{x}\langle z \rangle)$ is called a **redex**.

Example: Printer Server

Example 8.9

1. **Printer server** (cf. Example 8.1):

$$\underbrace{\bar{b}\langle a \rangle . S'}_S \parallel \underbrace{a(e) . P'}_P \parallel \underbrace{b(c) . \bar{c}\langle d \rangle . C'}_C \longrightarrow S' \parallel a(e) . P' \parallel \bar{a}\langle d \rangle . C'[a/c]$$

$$S' \parallel a(e) . P' \parallel \bar{a}\langle d \rangle . C'[a/c] \longrightarrow S' \parallel P'[d/e] \parallel C'[a/c]$$

(on the board)

Example: Printer Server

Example 8.9

1. **Printer server** (cf. Example 8.1):

$$\underbrace{\bar{b}\langle a \rangle . S'}_S \parallel \underbrace{a(e) . P'}_P \parallel \underbrace{b(c) . \bar{c}\langle d \rangle . C'}_C \longrightarrow S' \parallel a(e) . P' \parallel \bar{a}\langle d \rangle . C'[a/c]$$

$$S' \parallel a(e) . P' \parallel \bar{a}\langle d \rangle . C'[a/c] \longrightarrow S' \parallel P'[d/e] \parallel C'[a/c]$$

(on the board)

2. With **scope extension** ($P \parallel \text{new } x Q \equiv \text{new } x (P \parallel Q)$ if $x \notin \text{fn}(P)$):

$$\begin{aligned} & \text{new } b (\text{new } a (\bar{b}\langle a \rangle . S' \parallel a(e) . P') \parallel b(c) . \bar{c}\langle d \rangle . C') \\ \longrightarrow & \text{new } a, b (S' \parallel a(e) . P' \parallel \bar{a}\langle d \rangle . C'[a/c]) \end{aligned}$$

(on the board)