Concurrency Theory

Winter Semester 2019/20

Lecture 8: The π-Calculus

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

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

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

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

Recap: Modelling Mutual Exclusion Algorithms

Obtaining the LTS I

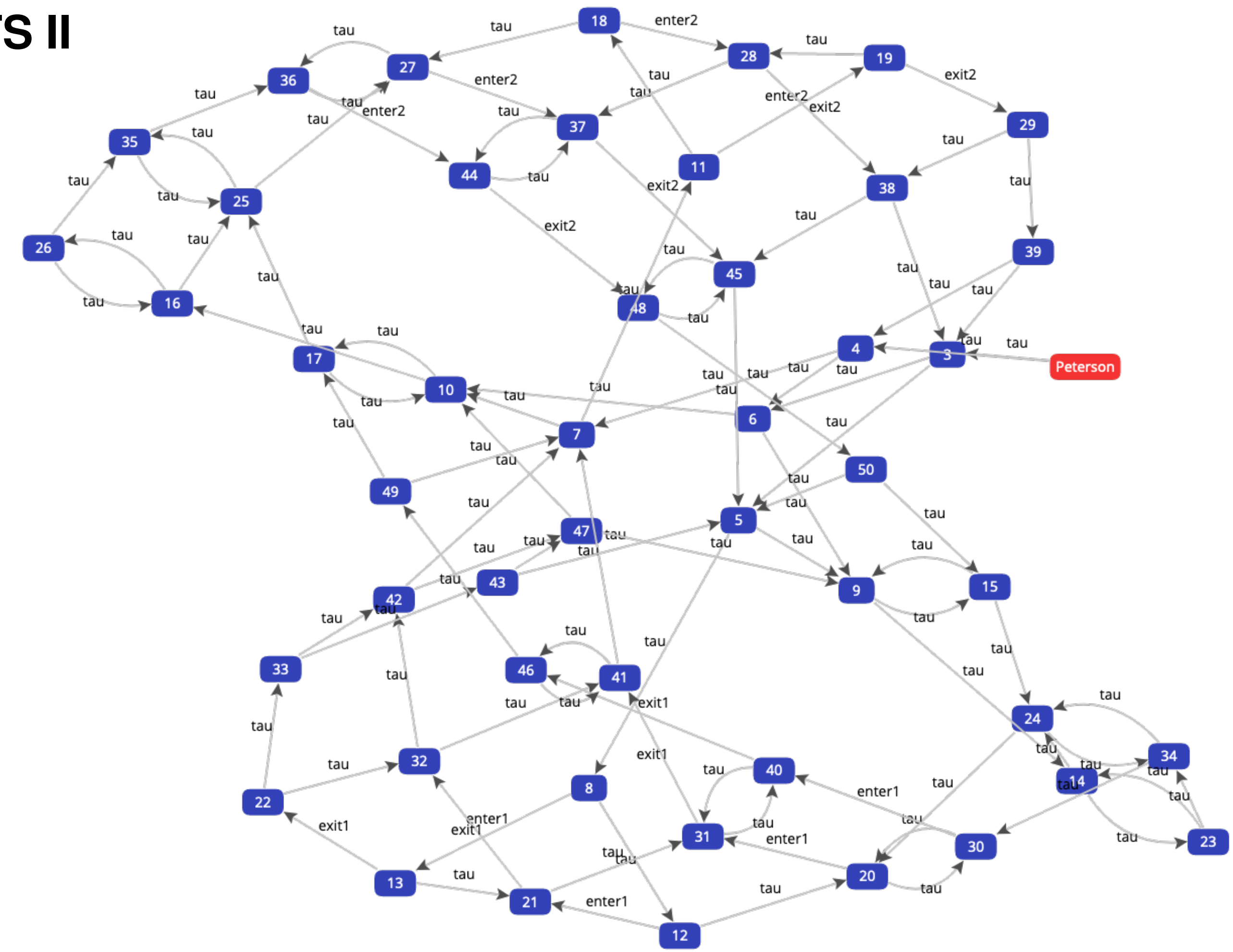
CAAL   Project   Edit   Explore   Verify   Games   About   Syntax

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;
```

Recap: Modelling Mutual Exclusion Algorithms

Obtaining the LTS II



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 Value-Passing CCS I

Definition (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:

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

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

Semantics of Value-Passing CCS I

Definition (Semantics of value-passing CCS)

A value-passing process definition $\big(C_i(x_1, \ldots, x_{n_i}) = P_i \mid 1 \leq i \leq k\big)$ 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 (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:

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

Translation of Value-Passing into Pure CCS II

Definition (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

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

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

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

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

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

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

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

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 $P, Q \in Prc$ 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 $P, Q \in Prc$ want to communicate, then both must syntactically refer to the same action name

$\Rightarrow$ every potential communication partner known beforehand,
no dynamic passing of communication links

Mobility in Concurrent Systems I

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

$\Rightarrow$ every potential communication partner known beforehand,
no dynamic passing of communication links

$\Rightarrow$ lack of modelling capabilities for mobility

Mobility in Concurrent Systems I

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

$\Rightarrow$ every potential communication partner known beforehand,
no dynamic passing of communication links

$\Rightarrow$ lack of modelling capabilities for mobility

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

$\Rightarrow$ 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
  $\Rightarrow$ 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:

$$\underbrace{\overline{b}\langle a\rangle.S'}_{S} \parallel \underbrace{b(c).\overline{c}\langle d\rangle.C'}_{C} \parallel \underbrace{a(e).P'}_{P}$$

- 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:

$$\begin{array}{rc} & \underbrace{\overline{b}\langle a\rangle.S'}_{S} \parallel \underbrace{b(c).\overline{c}\langle d\rangle.C'}_{C} \parallel \underbrace{a(e).P'}_{P} \\ \xrightarrow{\tau} & S' \parallel \overline{a}\langle d\rangle.C'[a/c] \parallel a(e).P' \end{array}$$

  - 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:

$$\begin{array}{ll} & \underbrace{\overline{b}\langle a\rangle.S'}_{S} \parallel \underbrace{b(c).\overline{c}\langle d\rangle.C'}_{C} \parallel \underbrace{a(e).P'}_{P} \\ \xrightarrow{\tau} & S' \parallel \overline{a}\langle d\rangle.C'[a/c] \parallel a(e).P' \\ \xrightarrow{\tau} & S' \parallel C'[a/c] \parallel P'[d/e] \end{array}$$

  - 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 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

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

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")

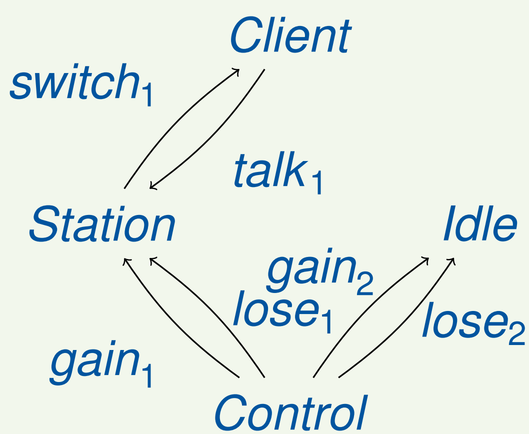
Mobile Clients I

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



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)

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*:

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

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*:

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

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

$$\begin{aligned} Control_1 &= \overline{lose_1}\langle talk_2, switch_2\rangle.\overline{gain_2}\langle talk_2, switch_2\rangle.Control_2 \\ Control_2 &= \overline{lose_2}\langle talk_1, switch_1\rangle.\overline{gain_1}\langle talk_1, switch_1\rangle.Control_1 \end{aligned}$$

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*:

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

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

$$\begin{aligned} Control_1 &= \overline{lose_1}\langle talk_2, switch_2\rangle.\overline{gain_2}\langle talk_2, switch_2\rangle.Control_2 \\ Control_2 &= \overline{lose_2}\langle talk_1, switch_1\rangle.\overline{gain_1}\langle talk_1, switch_1\rangle.Control_1 \end{aligned}$$

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

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

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

$$\begin{aligned} Client_i &:= Client(talk_i, switch_i) \\ Station_i &:= Station(talk_i, switch_i, gain_i, lose_i) \\ Idle_i &:= Idle(gain_i, lose_i) \\ L &:= (talk_i, switch_i, gain_i, lose_i \mid i \in \{1,2\}) \end{aligned}$$

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

$$\begin{aligned} Client_i &:= Client(talk_i, switch_i) \\ Station_i &:= Station(talk_i, switch_i, gain_i, lose_i) \\ Idle_i &:= Idle(gain_i, lose_i) \\ L &:= (talk_i, switch_i, gain_i, lose_i \mid i \in \{1, 2\}) \end{aligned}$$

- 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

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

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

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

Definition 8.3 (Syntax of monadic π-Calculus)

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

Syntax of the Monadic π-Calculus

Definition 8.3 (Syntax of monadic π-Calculus)

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

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

Syntax of the Monadic π-Calculus

Definition 8.3 (Syntax of monadic π-Calculus)

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

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

- The set Prc^{π} of π-Calculus process expressions is defined by the following syntax:

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

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

Software Modeling and Verification Chair

Syntax of the Monadic π-Calculus

Definition 8.3 (Syntax of monadic π-Calculus)

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

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

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

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

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

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

Free and Bound Names

Definition 8.4 (Free and bound names)

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

Free and Bound Names

Definition 8.4 (Free and bound names)

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

Free and Bound Names

Definition 8.4 (Free and bound names)

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

Free and Bound Names

Definition 8.4 (Free and bound names)

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

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

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 $\overline{x}\langle y\rangle$) is free.
- The set of bound/free names of a process expressions $P \in Prc^\pi$ is respectively denoted by $bn(P)/fn(P)$.

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

Example 8.5

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

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

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)

$P, Q \in Prc^{\pi}$ 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)

Structural Congruence

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

Definition 8.6 (Structural congruence)

$P, Q \in Prc^\pi$ 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 $+$)

Structural Congruence

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

Definition 8.6 (Structural congruence)

$P, Q \in Prc^\pi$ 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. $P \parallel Q \equiv Q \parallel P$, $P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$, $P \parallel \mathrm{nil} \equiv P$ (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)

$P, Q \in Prc^\pi$ 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. $P \parallel Q \equiv Q \parallel P$, $P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$, $P \parallel \mathsf{nil} \equiv P$ (Abelian monoid laws for $\parallel$)
4. $\mathsf{new}\,x\,\mathsf{nil} \equiv \mathsf{nil}$, $\mathsf{new}\,x, y\,P \equiv \mathsf{new}\,y, x\,P$,
   $P \parallel \mathsf{new}\,x\,Q \equiv \mathsf{new}\,x\,(P \parallel Q)$ if $x \notin fn(P)$ (scope extension)

Structural Congruence

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

Definition 8.6 (Structural congruence)

$P, Q \in Prc^{\pi}$ 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. $P \parallel Q \equiv Q \parallel P$, $P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$, $P \parallel \mathsf{nil} \equiv P$ (Abelian monoid laws for $\parallel$)
4. $\mathsf{new}\, x\, \mathsf{nil} \equiv \mathsf{nil}$, $\mathsf{new}\, x, y\, P \equiv \mathsf{new}\, y, x\, P$,
   $P \parallel \mathsf{new}\, x\, Q \equiv \mathsf{new}\, x\, (P \parallel Q)$ if $x \notin fn(P)$ (scope extension)
5. $!P \equiv P \parallel !P$ (unfolding)

A Standard Form

Theorem 8.7 (Standard form)

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

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

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)

A Standard Form

Theorem 8.7 (Standard form)

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

$$\mathsf{new}\ x_1, \ldots, x_k \left(P_1 \parallel \ldots \parallel P_m \parallel !Q_1 \parallel \ldots \parallel !Q_n\right)$$

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 $R \in Prc^\pi$ (on the board) □

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 $\longrightarrow \subseteq Prc^\pi \times Prc^\pi$ is generated by the rules:

$$\text{(Tau)}\frac{}{\tau.P + Q \longrightarrow P}$$

$$\text{(React)}\frac{}{(x(y).P + R) \parallel (\overline{x}\langle z\rangle.Q + S) \longrightarrow P[z/y] \parallel Q}$$

$$\text{(Par)}\frac{P \longrightarrow P'}{P \parallel Q \longrightarrow P' \parallel Q} \qquad \text{(Res)}\frac{P \rightarrow P'}{\text{new } x\, P \longrightarrow \text{new } x\, P'}$$

$$\text{(Struct)}\frac{P \longrightarrow P'}{Q \longrightarrow Q'} \quad \text{if } P \equiv Q \text{ and } P' \equiv Q'$$

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