Concurrency Theory

Lecture 10: The π-Calculus

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

Outline

Mobile Clients I

Example (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 case: two stations, one client

$$\begin{tikzcd}
 & \mathit{Client} \arrow[dl, bend left, "\mathit{talk}_1"] & \\
\mathit{Station} \arrow[ur, bend left, "\mathit{switch}_1"] & & \mathit{Idle} \\
 & \mathit{Control} \arrow[ul, bend left, "\mathit{gain}_1"] \arrow[ul, "\mathit{lose}_1"'] \arrow[ur, "\mathit{gain}_2"] \arrow[ur, bend right, "\mathit{lose}_2"'] &
\end{tikzcd}$$

Mobile Clients II

Example (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*:

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

- *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 (Hand-over protocol; continued)

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

$$\mathit{System}_1 = \mathrm{new}\, L\, (\mathit{Client}_1 \parallel \mathit{Station}_1 \parallel \mathit{Idle}_2 \parallel \mathit{Control}_1)$$

  where

$$\begin{aligned} \mathit{Client}_i &:= \mathit{Client}(\mathit{talk}_i, \mathit{switch}_i) \\ \mathit{Station}_i &:= \mathit{Station}(\mathit{talk}_i, \mathit{switch}_i, \mathit{gain}_i, \mathit{lose}_i) \\ \mathit{Idle}_i &:= \mathit{Idle}(\mathit{gain}_i, \mathit{lose}_i) \\ L &:= (\mathit{talk}_i, \mathit{switch}_i, \mathit{gain}_i, \mathit{lose}_i \mid i \in \{1, 2\}) \end{aligned}$$

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

$$\mathit{System}_1 \longrightarrow^* \mathit{System}_2$$

  where

$$\mathit{System}_2 = \mathrm{new}\, L\, (\mathit{Client}_2 \parallel \mathit{Idle}_1 \parallel \mathit{Station}_2 \parallel \mathit{Control}_2)$$

Outline

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

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 10.1 (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 10.1 (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

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

Syntax of the Monadic π-Calculus

Definition 10.1 (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, $x \in A$)

Syntax of the Monadic π-Calculus

Definition 10.1 (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

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

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

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

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

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

Free and Bound Names

Definition 10.2 (Free and bound names)

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

Free and Bound Names

Definition 10.2 (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 10.2 (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 denoted by $bn(P)/fn(P)$, resp.

Free and Bound Names

Definition 10.2 (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 denoted by $bn(P)/fn(P)$, resp.

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

Free and Bound Names

Definition 10.2 (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 denoted by $bn(P)/fn(P)$, resp.

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

Example 10.3

$$P = \mathsf{new}\, x\, (x(y).\mathsf{nil} \parallel \overline{z}\langle y\rangle.\mathsf{nil}) \\ \Longrightarrow \; bn(P) = \{x, y\}, fn(P) = \{y, z\}$$

Outline

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 10.4 (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 10.4 (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 10.4 (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 $\parallel$)

Structural Congruence

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

Definition 10.4 (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 10.4 (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 10.5 (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 10.5 (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)

Proof.

by induction on the structure of $R \in Prc^\pi$ (on the board) □

The Reaction Relation

Thanks to Theorem 10.5, only processes in standard form need to be considered for defining the operational semantics:

Definition 10.6

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

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

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

$$\text{(Par)}\,\frac{P \longrightarrow P'}{P \parallel Q \longrightarrow P' \parallel Q}$$

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

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

($P[z/y]$ 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.)

Example: Printer Server

Example 10.7

1. Printer server (cf. Example 9.9):

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

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

(on the board)

Example: Printer Server

Example 10.7

1. Printer server (cf. Example 9.9):

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

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

(on the board)

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

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

(on the board)