Modeling Concurrent and Probabilistic Systems

Lecture 13: The Polyadic π-Calculus with Process Calls

Joost-Pieter Katoen   Thomas Noll

Software Modeling and Verification Group
RWTH Aachen University
`noll@cs.rwth-aachen.de`

`http://www-i2.informatik.rwth-aachen.de/i2/mcps07/`

Winter Semester 2007/08

Outline

Repetition: Syntax of the Monadic π-Calculus

Definition (Syntax of monadic π-calculus)

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

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

- The set P^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, $x \in N$)

Conventions:

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

Repetition: Structural Congruence

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

Definition (Structural congruence)

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

Repetition: A Standard Form

Theorem (Standard form)

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

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

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

(If $m = n = 0$: nil; if $k = 0$: no restriction)

Proof.

by induction on the structure of $R \in P^\pi$ (on the board) $\square$

Repetition: The Reaction Relation

Thanks to Theorem 13.3, only processes in standard form need to be considered for defining the operational semantics:

Definition

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

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

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

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

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

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

($P[y \mapsto z]$ 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.)

Outline

Example: Printer Server

Example 13.1

1. Printer server (cf. Lecture 11):

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

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

(on the board)

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

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

(on the board)

Example: Printer Server

Example 13.1

1. Printer server (cf. Lecture 11):

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

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

   (on the board)

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

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

   (on the board)

Example: Mobile Clients

Example 13.2

- System specification (cf. Lecture 12):

$$\begin{aligned}
System_1 &= \mathsf{new}\,L\,(Client_1 \parallel Station_1 \parallel Idle_2 \parallel Control_1)\\
System_2 &= \mathsf{new}\,L\,(Client_2 \parallel Idle_1 \parallel Station_2 \parallel Control_2)\\
\lefteqn{Station(talk, switch, gain, lose)}\\
&= talk.Station(talk, switch, gain, lose) +\\
&\quad lose(t,s).\overline{switch}\langle t,s\rangle.Idle(gain, lose)\\
Idle(gain, lose) &= gain(t,s).Station(t,s,gain,lose)\\
Control_1 &= \overline{lose_1}\langle talk_2, switch_2\rangle.\overline{gain_2}\langle talk_2, switch_2\rangle.Control_2\\
Control_2 &= \overline{lose_2}\langle talk_1, switch_1\rangle.\overline{gain_1}\langle talk_1, switch_1\rangle.Control_1\\
Client(talk, switch) &= \overline{talk}.Client(talk, switch) + switch(t,s).Client(t,s)
\end{aligned}$$

- Use additional congruence rule for process calls:
  if $A(\vec{x}) = P_A$, then $A(\vec{y}) \equiv P_A[\vec{x} \mapsto \vec{y}]$
- Use additional reaction rule for polyadic communication:

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

- Show $System_1 \longrightarrow^* System_2$ (on the board)

Example: Mobile Clients

Example 13.2

- System specification (cf. Lecture 12):

$$\begin{aligned}
System_1 &= \mathsf{new}\, L\, (Client_1 \parallel Station_1 \parallel Idle_2 \parallel Control_1)\\
System_2 &= \mathsf{new}\, L\, (Client_2 \parallel Idle_1 \parallel Station_2 \parallel Control_2)\\
\end{aligned}$$

$$\begin{aligned}
&Station(talk, switch, gain, lose)\\
&\quad = talk.Station(talk, switch, gain, lose) +\\
&\quad\quad lose(t, s).\overline{switch}\langle t, s\rangle.Idle(gain, lose)\\
Idle(gain, lose) &= gain(t, s).Station(t, s, gain, lose)\\
Control_1 &= \overline{lose_1}\langle talk_2, switch_2\rangle.\overline{gain_2}\langle talk_2, switch_2\rangle.Control_2\\
Control_2 &= \overline{lose_2}\langle talk_1, switch_1\rangle.\overline{gain_1}\langle talk_1, switch_1\rangle.Control_1\\
Client(talk, switch) &= \overline{talk}.Client(talk, switch) + switch(t, s).Client(t, s)
\end{aligned}$$

- Use additional congruence rule for process calls:
  if $A(\vec{x}) = P_A$, then $A(\vec{y}) \equiv P_A[\vec{x} \mapsto \vec{y}]$

- Use additional reaction rule for polyadic communication:

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

- Show $System_1 \longrightarrow^* System_2$ (on the board)

Example: Mobile Clients

Example 13.2

- System specification (cf. Lecture 12):

$$\begin{aligned}
System_1 &= \mathsf{new}\, L\, (Client_1 \parallel Station_1 \parallel Idle_2 \parallel Control_1) \\
System_2 &= \mathsf{new}\, L\, (Client_2 \parallel Idle_1 \parallel Station_2 \parallel Control_2) \\
Station(talk, switch, gain, lose) & \\
&= talk.Station(talk, switch, gain, lose) + \\
&\quad lose(t, s).\overline{switch}\langle t, s\rangle.Idle(gain, lose) \\
Idle(gain, lose) &= gain(t, s).Station(t, s, gain, lose) \\
Control_1 &= \overline{lose_1}\langle talk_2, switch_2\rangle.\overline{gain_2}\langle talk_2, switch_2\rangle.Control_2 \\
Control_2 &= \overline{lose_2}\langle talk_1, switch_1\rangle.\overline{gain_1}\langle talk_1, switch_1\rangle.Control_1 \\
Client(talk, switch) &= \overline{talk}.Client(talk, switch) + switch(t, s).Client(t, s)
\end{aligned}$$

- Use additional congruence rule for process calls:
  if $A(\vec{x}) = P_A$, then $A(\vec{y}) \equiv P_A[\vec{x} \mapsto \vec{y}]$

- Use additional reaction rule for polyadic communication:

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

- Show $System_1 \longrightarrow^* System_2$ (on the board)

Example: Mobile Clients

Example 13.2

- System specification (cf. Lecture 12):

$$\begin{aligned}
\mathit{System}_1 &= \mathsf{new}\, L\, (\mathit{Client}_1 \parallel \mathit{Station}_1 \parallel \mathit{Idle}_2 \parallel \mathit{Control}_1) \\
\mathit{System}_2 &= \mathsf{new}\, L\, (\mathit{Client}_2 \parallel \mathit{Idle}_1 \parallel \mathit{Station}_2 \parallel \mathit{Control}_2) \\
\mathit{Station}(\mathit{talk}, \mathit{switch}, \mathit{gain}, \mathit{lose}) & \\
&= \mathit{talk}.\mathit{Station}(\mathit{talk}, \mathit{switch}, \mathit{gain}, \mathit{lose}) + \\
&\quad \mathit{lose}(t,s).\overline{\mathit{switch}}\langle t,s\rangle.\mathit{Idle}(\mathit{gain}, \mathit{lose}) \\
\mathit{Idle}(\mathit{gain}, \mathit{lose}) &= \mathit{gain}(t,s).\mathit{Station}(t,s,\mathit{gain},\mathit{lose}) \\
\mathit{Control}_1 &= \overline{\mathit{lose}_1}\langle \mathit{talk}_2, \mathit{switch}_2\rangle.\overline{\mathit{gain}_2}\langle \mathit{talk}_2, \mathit{switch}_2\rangle.\mathit{Control}_2 \\
\mathit{Control}_2 &= \overline{\mathit{lose}_2}\langle \mathit{talk}_1, \mathit{switch}_1\rangle.\overline{\mathit{gain}_1}\langle \mathit{talk}_1, \mathit{switch}_1\rangle.\mathit{Control}_1 \\
\mathit{Client}(\mathit{talk}, \mathit{switch}) &= \overline{\mathit{talk}}.\mathit{Client}(\mathit{talk}, \mathit{switch}) + \mathit{switch}(t,s).\mathit{Client}(t,s)
\end{aligned}$$

- Use additional congruence rule for process calls:
  if $A(\vec{x}) = P_A$, then $A(\vec{y}) \equiv P_A[\vec{x} \mapsto \vec{y}]$

- Use additional reaction rule for polyadic communication:

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

- Show $\mathit{System}_1 \longrightarrow^* \mathit{System}_2$ (on the board)

Outline

Polyadic Communication I

- **So far:** messages with exactly one name
- **Now:** arbitrary number
- New types of action prefixes:

$$x(y_1, \ldots, y_n) \qquad \text{and} \qquad \overline{x}\langle z_1, \ldots, z_n \rangle$$

  where $n \in \mathbb{N}$ and all y_i distinct
- Expected behavior:

$$x(y_1, \ldots, y_n).P \parallel \overline{x}\langle z_1, \ldots, z_n \rangle.Q \longrightarrow P[y_1 \mapsto z_1, \ldots, y_n \mapsto z_n] \parallel Q$$

  (replacement of free names)
- Obvious attempt for encoding:

$$x(y_1) \ldots x(y_n).P \qquad \text{and} \qquad \overline{x}\langle z_1 \rangle \ldots \overline{x}\langle z_n \rangle.Q$$

Polyadic Communication I

- **So far:** messages with exactly one name
- **Now:** arbitrary number
- New types of action prefixes:

$$x(y_1, \dots, y_n) \qquad \text{and} \qquad \overline{x}\langle z_1, \dots, z_n \rangle$$

  where $n \in \mathbb{N}$ and all y_i distinct
- Expected behavior:

$$x(y_1, \dots, y_n).P \parallel \overline{x}\langle z_1, \dots, z_n \rangle.Q \longrightarrow P[y_1 \mapsto z_1, \dots, y_n \mapsto z_n] \parallel Q$$

  (replacement of free names)
- Obvious attempt for encoding:

$$x(y_1) \dots x(y_n).P \qquad \text{and} \qquad \overline{x}\langle z_1 \rangle \dots \overline{x}\langle z_n \rangle.Q$$

Polyadic Communication I

- **So far:** messages with exactly one name
- **Now:** arbitrary number
- New types of action prefixes:

$$x(y_1, \ldots, y_n) \qquad \text{and} \qquad \overline{x}\langle z_1, \ldots, z_n \rangle$$

  where $n \in \mathbb{N}$ and all y_i distinct
- Expected behavior:

$$x(y_1, \ldots, y_n).P \parallel \overline{x}\langle z_1, \ldots, z_n \rangle.Q \longrightarrow P[y_1 \mapsto z_1, \ldots, y_n \mapsto z_n] \parallel Q$$

  (replacement of free names)
- Obvious attempt for encoding:

$$x(y_1) \ldots x(y_n).P \qquad \text{and} \qquad \overline{x}\langle z_1 \rangle \ldots \overline{x}\langle z_n \rangle.Q$$

Polyadic Communication I

- **So far:** messages with exactly one name
- **Now:** arbitrary number
- New types of action prefixes:

$$x(y_1, \ldots, y_n) \qquad \text{and} \qquad \overline{x}\langle z_1, \ldots, z_n\rangle$$

  where $n \in \mathbb{N}$ and all y_i distinct
- Expected behavior:

$$x(y_1, \ldots, y_n).P \parallel \overline{x}\langle z_1, \ldots, z_n\rangle.Q \longrightarrow P[y_1 \mapsto z_1, \ldots, y_n \mapsto z_n] \parallel Q$$

  (replacement of free names)
- Obvious attempt for encoding:

$$x(y_1) \ldots x(y_n).P \qquad \text{and} \qquad \overline{x}\langle z_1\rangle \ldots \overline{x}\langle z_n\rangle.Q$$

Polyadic Communication II

- But consider the following counterexample.

  Polyadic representation:

$$\begin{tikzcd}[column sep=small] & x(y_1, y_2).P \parallel \overline{x}\langle z_1, z_2\rangle.Q \parallel \overline{x}\langle z_1', z_2'\rangle.Q' \arrow[dl] \arrow[dr] & \\ P[y_1 \mapsto z_1, y_2 \mapsto z_2] \parallel Q \parallel \overline{x}\langle z_1', z_2'\rangle.Q' & & P[y_1 \mapsto z_1', y_2 \mapsto z_2'] \parallel \overline{x}\langle z_1, z_2\rangle.Q \parallel Q' \end{tikzcd}$$

  Monadic encoding:

$$\begin{tikzcd}[column sep=small] P[y_1 \mapsto z_1, y_2 \mapsto z_2] \parallel \ldots \surd & P[y_1 \mapsto z_1', y_2 \mapsto z_2'] \parallel \ldots \surd \\ x(y_1).x(y_2).P \parallel \overline{x}\langle z_1\rangle.\overline{x}\langle z_2\rangle.Q \parallel \overline{x}\langle z_1'\rangle.\overline{x}\langle z_2'\rangle.Q' \arrow[u, "2"] \arrow[ur, "2"] \arrow[d, "2"] \arrow[dr, "2"] & \\ P[y_1 \mapsto z_1, y_2 \mapsto z_1'] \parallel \ldots \divideontimes & P[y_1 \mapsto z_1', y_2 \mapsto z_1] \parallel \ldots \divideontimes \end{tikzcd}$$

- **Solution:** avoid interferences by first introducing a fresh channel:

$$\begin{aligned} x(y_1, \ldots, y_n).P &\mapsto x(w).w(y_1) \ldots w(y_n).P \\ \overline{x}\langle z_1, \ldots, z_n\rangle.Q &\mapsto \mathsf{new}\, w\, (\overline{x}\langle w\rangle.\overline{w}\langle z_1\rangle \ldots \overline{w}\langle z_n\rangle.Q) \end{aligned}$$

  where $w \notin fn(Q)$

- **Correctness:** see Ex. 8.4

Polyadic Communication II

- But consider the following counterexample.

  Polyadic representation:

$$\begin{tikzcd}[column sep=tiny]
 & x(y_1,y_2).P \parallel \overline{x}\langle z_1,z_2\rangle.Q \parallel \overline{x}\langle z'_1,z'_2\rangle.Q' \arrow[dl] \arrow[dr] & \\
P[y_1 \mapsto z_1, y_2 \mapsto z_2] \parallel Q \parallel \overline{x}\langle z'_1,z'_2\rangle.Q' & & P[y_1 \mapsto z'_1, y_2 \mapsto z'_2] \parallel \overline{x}\langle z_1,z_2\rangle.Q \parallel Q'
\end{tikzcd}$$

  Monadic encoding:

$$\begin{tikzcd}[column sep=tiny]
P[y_1 \mapsto z_1, y_2 \mapsto z_2] \parallel \ldots \surd & & P[y_1 \mapsto z'_1, y_2 \mapsto z'_2] \parallel \ldots \surd \\
 & x(y_1).x(y_2).P \parallel \overline{x}\langle z_1\rangle.\overline{x}\langle z_2\rangle.Q \parallel \overline{x}\langle z'_1\rangle.\overline{x}\langle z'_2\rangle.Q' \arrow[ul, "2"] \arrow[ur, "2"'] \arrow[dl, "2"'] \arrow[dr, "2"] & \\
P[y_1 \mapsto z_1, y_2 \mapsto z'_1] \parallel \ldots ⋇ & & P[y_1 \mapsto z'_1, y_2 \mapsto z_1] \parallel \ldots ⋇
\end{tikzcd}$$

- **Solution:** avoid interferences by first introducing a fresh channel:

$$x(y_1,\ldots,y_n).P \mapsto x(w).w(y_1)\ldots w(y_n).P$$
$$\overline{x}\langle z_1,\ldots,z_n\rangle.Q \mapsto \mathsf{new}\, w\,(\overline{x}\langle w\rangle.\overline{w}\langle z_1\rangle\ldots\overline{w}\langle z_n\rangle.Q)$$

  where $w \notin fn(Q)$

- **Correctness:** see Ex. 8.4

Polyadic Communication II

- But consider the following counterexample.

  Polyadic representation:

$$\begin{tikzcd}[column sep=tiny]
 & x(y_1, y_2).P \parallel \overline{x}\langle z_1, z_2\rangle.Q \parallel \overline{x}\langle z_1', z_2'\rangle.Q' \arrow[dl] \arrow[dr] & \\
P[y_1 \mapsto z_1, y_2 \mapsto z_2] \parallel Q \parallel \overline{x}\langle z_1', z_2'\rangle.Q' & & P[y_1 \mapsto z_1', y_2 \mapsto z_2'] \parallel \overline{x}\langle z_1, z_2\rangle.Q \parallel Q'
\end{tikzcd}$$

  Monadic encoding:

$$\begin{tikzcd}[column sep=tiny]
P[y_1 \mapsto z_1, y_2 \mapsto z_2] \parallel \ldots \surd & & P[y_1 \mapsto z_1', y_2 \mapsto z_2'] \parallel \ldots \surd \\
 & x(y_1).x(y_2).P \parallel \overline{x}\langle z_1\rangle.\overline{x}\langle z_2\rangle.Q \parallel \overline{x}\langle z_1'\rangle.\overline{x}\langle z_2'\rangle.Q' \arrow[ul, "2"] \arrow[ur, "2"'] \arrow[dl, "2"'] \arrow[dr, "2"] & \\
P[y_1 \mapsto z_1, y_2 \mapsto z_1'] \parallel \ldots ⁂ & & P[y_1 \mapsto z_1', y_2 \mapsto z_1] \parallel \ldots ⁂
\end{tikzcd}$$

- **Solution:** avoid interferences by first introducing a fresh channel:

$$\begin{aligned} x(y_1, \ldots, y_n).P &\mapsto x(w).w(y_1) \ldots w(y_n).P \\ \overline{x}\langle z_1, \ldots, z_n\rangle.Q &\mapsto \mathsf{new}\, w\, (\overline{x}\langle w\rangle.\overline{w}\langle z_1\rangle \ldots \overline{w}\langle z_n\rangle.Q) \end{aligned}$$

  where $w \notin fn(Q)$

- **Correctness:** see Ex. 8.4

Polyadic Communication II

- But consider the following counterexample.

  Polyadic representation:

$$\begin{tikzcd}[column sep=tiny] & x(y_1,y_2).P \parallel \overline{x}\langle z_1,z_2\rangle.Q \parallel \overline{x}\langle z_1',z_2'\rangle.Q' \arrow[dl] \arrow[dr] & \\ P[y_1 \mapsto z_1, y_2 \mapsto z_2] \parallel Q \parallel \overline{x}\langle z_1',z_2'\rangle.Q' & & P[y_1 \mapsto z_1', y_2 \mapsto z_2'] \parallel \overline{x}\langle z_1,z_2\rangle.Q \parallel Q' \end{tikzcd}$$

  Monadic encoding:

$$\begin{array}{ccc} P[y_1 \mapsto z_1, y_2 \mapsto z_2] \parallel \ldots \checkmark & & P[y_1 \mapsto z_1', y_2 \mapsto z_2'] \parallel \ldots \checkmark \\ \uparrow 2 & & \uparrow 2 \\ & x(y_1).x(y_2).P \parallel \overline{x}\langle z_1\rangle.\overline{x}\langle z_2\rangle.Q \parallel \overline{x}\langle z_1'\rangle.\overline{x}\langle z_2'\rangle.Q' & \\ \downarrow 2 & & \downarrow 2 \\ P[y_1 \mapsto z_1, y_2 \mapsto z_1'] \parallel \ldots \divideontimes & & P[y_1 \mapsto z_1', y_2 \mapsto z_1] \parallel \ldots \divideontimes \end{array}$$

- **Solution:** avoid interferences by first introducing a fresh channel:

$$\begin{aligned} x(y_1,\ldots,y_n).P &\mapsto x(w).w(y_1)\ldots w(y_n).P \\ \overline{x}\langle z_1,\ldots,z_n\rangle.Q &\mapsto \mathsf{new}\, w\, (\overline{x}\langle w\rangle.\overline{w}\langle z_1\rangle \ldots \overline{w}\langle z_n\rangle.Q) \end{aligned}$$

  where $w \notin fn(Q)$

- **Correctness:** see Ex. 8.4

Outline

Recursive Process Calls I

- **So far:** process replication $!P$
- **Now:** parametric process definitions of the form

$$A(x_1, \ldots, x_n) = P_A$$

  where A is a process identifier and P_A a process expression containing calls of A (and other parametric processes)
- Again: possible to simulate in basic calculus by using
  - message passing to model parameter passing to A
  - replication to model the multiple activations of A
  - restriction to model the scope of the definition of A

Recursive Process Calls I

- **So far:** process replication $!P$
- **Now:** parametric process definitions of the form

$$A(x_1, \ldots, x_n) = P_A$$

  where A is a process identifier and P_A a process expression containing calls of A (and other parametric processes)
- Again: possible to simulate in basic calculus by using
  - message passing to model parameter passing to A
  - replication to model the multiple activations of A
  - restriction to model the scope of the definition of A

Recursive Process Calls II

The encoding

- of a process definition $A(\vec{x}) = P_A$
- with right-hand side $P_A = \ldots A(\vec{u}) \ldots A(\vec{v}) \ldots$
- for main process $Q = \ldots A(\vec{y}) \ldots A(\vec{z}) \ldots$

is defined as follows:

1. Let $a \in N$ be a new name (standing for A).
2. For any process R, let $\hat{R}$ be the result of replacing every call $A(\vec{w})$ by $\overline{a}\langle\vec{w}\rangle$.
3. Replace Q by $Q' := \mathsf{new}\, a\, (\hat{Q} \parallel !a(\vec{x}).\hat{P}_A)$.

(In the presence of more than one process identifier, Q' will contain a replicated component for each definition.)

Example 13.3

One-place buffer:

$$B(in, out) = in(x).\overline{out}\langle x\rangle.B(in, out)$$

(on the board)

Recursive Process Calls II

The encoding

- of a process definition $A(\vec{x}) = P_A$
- with right-hand side $P_A = \ldots A(\vec{u}) \ldots A(\vec{v}) \ldots$
- for main process $Q = \ldots A(\vec{y}) \ldots A(\vec{z}) \ldots$

is defined as follows:

1. Let $a \in N$ be a new name (standing for A).
2. For any process R, let $\hat{R}$ be the result of replacing every call $A(\vec{w})$ by $\overline{a}\langle\vec{w}\rangle$.
3. Replace Q by $Q' := \mathsf{new}\, a\, (\hat{Q} \parallel !a(\vec{x}).\hat{P_A})$.

(In the presence of more than one process identifier, Q' will contain a replicated component for each definition.)

Example 13.3

One-place buffer:

$$B(in, out) = in(x).\overline{out}\langle x\rangle.B(in, out)$$

(on the board)

Recursive Process Calls II

The encoding

- of a process definition $A(\vec{x}) = P_A$
- with right-hand side $P_A = \ldots A(\vec{u}) \ldots A(\vec{v}) \ldots$
- for main process $Q = \ldots A(\vec{y}) \ldots A(\vec{z}) \ldots$

is defined as follows:

1. Let $a \in N$ be a new name (standing for A).
2. For any process R, let $\hat{R}$ be the result of replacing every call $A(\vec{w})$ by $\overline{a}\langle\vec{w}\rangle$.
3. Replace Q by $Q' := \mathsf{new}\, a\, (\hat{Q} \parallel !a(\vec{x}).\hat{P}_A)$.

(In the presence of more than one process identifier, Q' will contain a replicated component for each definition.)

Example 13.3

One-place buffer:

$$B(in, out) = in(x).\overline{out}\langle x\rangle.B(in, out)$$

(on the board)