Concurrency Theory

Winter Semester 2015/16

Lecture 9: The π-Calculus

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

http://moves.rwth-aachen.de/teaching/ws-1516/ct/

Outline of Lecture 9

Recap: Modelling Mobile Concurrent Systems

Mobile Clients I

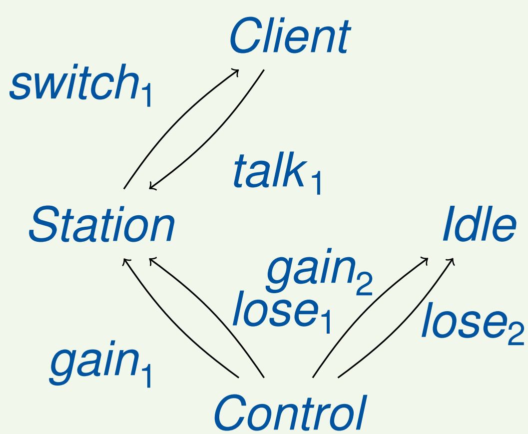
Example (Hand-over protocol)

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



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*:

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

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

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

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

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

Mobile Clients III

Example (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

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

Outline of Lecture 9

Recap: Modelling Mobile Concurrent Systems

Syntax of the Monadic π-Calculus

Semantics of the Monadic π-Calculus

Mobile Clients Revisited

The Polyadic 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 9.1 (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 9.1 (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 9.1 (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

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

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

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

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

Syntax of the Monadic π-Calculus

Definition 9.1 (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:

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

(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 9.2 (Free and bound names)

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

Free and Bound Names

Definition 9.2 (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 9.2 (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 respectively denoted by $bn(P)/fn(P)$.

Free and Bound Names

Definition 9.2 (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 respectively denoted by $bn(P)/fn(P)$.

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

Free and Bound Names

Definition 9.2 (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 respectively denoted by $bn(P)$/$fn(P)$.

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

Example 9.3

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

Outline of Lecture 9

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 9.4 (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 9.4 (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 9.4 (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 9.4 (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 9.4 (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 9.5 (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 9.5 (Standard form)

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

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

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

Definition 9.6

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

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

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

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

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

- $P[z/y]$ 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.

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Example: Printer Server

Example 9.7

1. Printer server (cf. Example 8.9):

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

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

(on the board)

Example: Printer Server

Example 9.7

1. Printer server (cf. Example 8.9):

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

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

   (on the board)

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

Outline of Lecture 9

Example: Mobile Clients

Example 9.8

- System specification (cf. Example 8.10):

$$\begin{aligned}
System_1 &= \text{new } L\,(Client_1 \parallel Station_1 \parallel Idle_2 \parallel Control_1) \\
System_2 &= \text{new } L\,(Client_2 \parallel Idle_1 \parallel Station_2 \parallel Control_2) \\
Station(talk, switch, gain, lose) &= talk.Station(talk, switch, gain, lose) + \\
&\quad lose(t, s).\overline{switch}\langle t, s\rangle.Idle(gain, lose) \\
Idle(gain, lose) &= gain(t, s).Station(t, s, gain, lose) \\
Control_1 &= \overline{lose_1}\langle talk_2, switch_2\rangle.\overline{gain_2}\langle talk_2, switch_2\rangle.Control_2 \\
Control_2 &= \overline{lose_2}\langle talk_1, switch_1\rangle.\overline{gain_1}\langle talk_1, switch_1\rangle.Control_1 \\
Client(talk, switch) &= \overline{talk}.Client(talk, switch) + switch(t, s).Client(t, s) \\
L &= (talk_i, switch_i, gain_i, lose_i \mid i \in \{1, 2\})
\end{aligned}$$

Example: Mobile Clients

Example 9.8

- System specification (cf. Example 8.10):

$$\begin{aligned}
System_1 &= \text{new } L\,(Client_1 \parallel Station_1 \parallel Idle_2 \parallel Control_1)\\
System_2 &= \text{new } L\,(Client_2 \parallel Idle_1 \parallel Station_2 \parallel Control_2)\\
Station(talk, switch, gain, lose) &= talk.Station(talk, switch, gain, lose) +\\
&\quad lose(t, s).\overline{switch}\langle t, s\rangle.Idle(gain, lose)\\
Idle(gain, lose) &= gain(t, s).Station(t, s, gain, lose)\\
Control_1 &= \overline{lose_1}\langle talk_2, switch_2\rangle.\overline{gain_2}\langle talk_2, switch_2\rangle.Control_2\\
Control_2 &= \overline{lose_2}\langle talk_1, switch_1\rangle.\overline{gain_1}\langle talk_1, switch_1\rangle.Control_1\\
Client(talk, switch) &= \overline{talk}.Client(talk, switch) + switch(t, s).Client(t, s)\\
L &= (talk_i, switch_i, gain_i, lose_i \mid i \in \{1, 2\})
\end{aligned}$$

- Use additional reaction rule for polyadic communication:

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

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

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Example: Mobile Clients

Example 9.8

- System specification (cf. Example 8.10):

$$
\begin{aligned}
System_1 &= \text{new}\, L\, (Client_1 \parallel Station_1 \parallel Idle_2 \parallel Control_1) \\
System_2 &= \text{new}\, L\, (Client_2 \parallel Idle_1 \parallel Station_2 \parallel Control_2) \\
Station(talk, switch, gain, lose) &= talk.Station(talk, switch, gain, lose) + \\
&\quad lose(t, s).\overline{switch}\langle t, s\rangle.Idle(gain, lose) \\
Idle(gain, lose) &= gain(t, s).Station(t, s, gain, lose) \\
Control_1 &= \overline{lose_1}\langle talk_2, switch_2\rangle.\overline{gain_2}\langle talk_2, switch_2\rangle.Control_2 \\
Control_2 &= \overline{lose_2}\langle talk_1, switch_1\rangle.\overline{gain_1}\langle talk_1, switch_1\rangle.Control_1 \\
Client(talk, switch) &= \overline{talk}.Client(talk, switch) + switch(t, s).Client(t, s) \\
L &= (talk_i, switch_i, gain_i, lose_i \mid i \in \{1, 2\})
\end{aligned}
$$

- Use additional reaction rule for polyadic communication:

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

- Use additional congruence rule for process calls: if $A(\vec{x}) = P_A$, then $A(\vec{y}) \equiv P_A[\vec{y}/\vec{x}]$
- Show $System_1 \longrightarrow^* System_2$ (on the board)

Outline of Lecture 9

Recap: Modelling Mobile Concurrent Systems

Syntax of the Monadic π-Calculus

Semantics of the Monadic π-Calculus

Mobile Clients Revisited

The Polyadic π-Calculus

Polyadic Communication I

- **So far:** messages with exactly one name
- **Now:** arbitrary number

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

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:

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

  (replacement of free names)

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:

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

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

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

Polyadic Communication II

- But consider the following counterexample.

  Polyadic representation:

$$x(y_1, y_2).P \parallel \overline{x}\langle z_1, z_2\rangle.Q \parallel \overline{x}\langle z_1', z_2'\rangle.Q'$$

$$\swarrow \qquad \searrow$$

$$P[z_1/y_1, z_2/y_2] \parallel Q \parallel \overline{x}\langle z_1', z_2'\rangle.Q' \qquad P[z_1'/y_1, z_2'/y_2] \parallel \overline{x}\langle z_1, z_2\rangle.Q \parallel Q'$$

Polyadic Communication II

- But consider the following counterexample.

  Polyadic representation:

$$x(y_1, y_2).P \parallel \overline{x}\langle z_1, z_2\rangle.Q \parallel \overline{x}\langle z'_1, z'_2\rangle.Q'$$

$$\swarrow \quad \searrow$$

$$P[z_1/y_1, z_2/y_2] \parallel Q \parallel \overline{x}\langle z'_1, z'_2\rangle.Q' \qquad P[z'_1/y_1, z'_2/y_2] \parallel \overline{x}\langle z_1, z_2\rangle.Q \parallel Q'$$

Monadic encoding: $P[z_1/y_1, z_2/y_2] \parallel \ldots$ ✓ $P[z'_1/y_1, z'_2/y_2] \parallel \ldots$ ✓

$$\uparrow^2 \qquad\qquad \uparrow^2$$

$$x(y_1).x(y_2).P \parallel \overline{x}\langle z_1\rangle.\overline{x}\langle z_2\rangle.Q \parallel \overline{x}\langle z'_1\rangle.\overline{x}\langle z'_2\rangle.Q'$$

$$\downarrow_2 \qquad\qquad \downarrow_2$$

$$P[z_1/y_1, z'_1/y_2] \parallel \ldots \;\lightning \qquad P[z'_1/y_1, z_1/y_2] \parallel \ldots \;\lightning$$

Polyadic Communication II

- But consider the following counterexample.

  Polyadic representation:

$$x(y_1, y_2).P \parallel \overline{x}\langle z_1, z_2\rangle.Q \parallel \overline{x}\langle z_1', z_2'\rangle.Q'$$

$$\swarrow \quad \searrow$$

$$P[z_1/y_1, z_2/y_2] \parallel Q \parallel \overline{x}\langle z_1', z_2'\rangle.Q' \qquad P[z_1'/y_1, z_2'/y_2] \parallel \overline{x}\langle z_1, z_2\rangle.Q \parallel Q'$$

  Monadic encoding: $P[z_1/y_1, z_2/y_2] \parallel \ldots \;\checkmark \quad P[z_1'/y_1, z_2'/y_2] \parallel \ldots \;\checkmark$

$$\uparrow^2 \qquad\qquad \uparrow^2$$

$$x(y_1).x(y_2).P \parallel \overline{x}\langle z_1\rangle.\overline{x}\langle z_2\rangle.Q \parallel \overline{x}\langle z_1'\rangle.\overline{x}\langle z_2'\rangle.Q'$$

$$\downarrow_2 \qquad\qquad \downarrow_2$$

$$P[z_1/y_1, z_1'/y_2] \parallel \ldots \;\lightning \qquad P[z_1'/y_1, z_1/y_2] \parallel \ldots \;\lightning$$

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

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

  where $w \notin fn(Q)$

Polyadic Communication II

- But consider the following counterexample.

  Polyadic representation:

$$x(y_1, y_2).P \parallel \overline{x}\langle z_1, z_2\rangle.Q \parallel \overline{x}\langle z_1', z_2'\rangle.Q'$$

$$\swarrow \quad \searrow$$

$$P[z_1/y_1, z_2/y_2] \parallel Q \parallel \overline{x}\langle z_1', z_2'\rangle.Q' \qquad P[z_1'/y_1, z_2'/y_2] \parallel \overline{x}\langle z_1, z_2\rangle.Q \parallel Q'$$

  Monadic encoding: $P[z_1/y_1, z_2/y_2] \parallel \ldots$ ✓ $P[z_1'/y_1, z_2'/y_2] \parallel \ldots$ ✓

$$\uparrow^2 \qquad \uparrow^2$$

$$x(y_1).x(y_2).P \parallel \overline{x}\langle z_1\rangle.\overline{x}\langle z_2\rangle.Q \parallel \overline{x}\langle z_1'\rangle.\overline{x}\langle z_2'\rangle.Q'$$

$$\downarrow_2 \qquad \downarrow_2$$

$$P[z_1/y_1, z_1'/y_2] \parallel \ldots \ \lightning \qquad P[z_1'/y_1, z_1/y_2] \parallel \ldots \ \lightning$$

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

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

  where $w \notin fn(Q)$

- **Correctness:** see exercises