Modeling Concurrent and Probabilistic Systems

Lecture 12: The Monadic π-Calculus

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

1 Repetition: Modeling Mobile Concurrent Systems

2 Another Example: Mobile Clients

3 The Monadic π-Calculus

Repetition: Mobility in Concurrent Systems I

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

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

$\Longrightarrow$ no mobility

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

$\Longrightarrow$ π-calculus

Example (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
  $\Longrightarrow$ 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:

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

$$\xrightarrow{\tau} S' \parallel \overline{a}\langle d\rangle.C' \parallel a(e).P'$$

$$\xrightarrow{\tau} S' \parallel C' \parallel P'[e \mapsto d]$$

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

Example (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
  $\implies$ 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:

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

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

Example (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
  $\Longrightarrow$ 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:

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

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

Example (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
  $\implies$ 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:

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

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

Example (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
  $\Longrightarrow$ 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:

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

$$\xrightarrow{\tau} S' \parallel \overline{a}\langle d\rangle.C' \parallel a(e).P'$$

$$\xrightarrow{\tau} S' \parallel C' \parallel P'[e \mapsto d]$$

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

Example (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
  $\Longrightarrow$ 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:

$$\underbrace{\bar{b}\langle a\rangle.S'}_{S} \parallel \underbrace{b(c).\overline{c}\langle d\rangle.C'}_{C} \parallel \underbrace{a(e).P'}_{P}$$
$$\xrightarrow{\tau} S' \parallel \overline{a}\langle d\rangle.C' \parallel a(e).P'$$
$$\xrightarrow{\tau} S' \parallel C' \parallel P'[e \mapsto d]$$

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

Example (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: for P
  - b: for S
  - d: for data to be printed
- If a is only way to access P
  $\implies$ P "moves" from S to C

Outline

1 Repetition: Modeling Mobile Concurrent Systems

2 Another Example: Mobile Clients

3 The Monadic π-Calculus

Mobile Clients I

Scenario:

- client devices moving around (phones, PCs, sensors, ...)
- each radioconnected 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

$$Client$$

$$talk_1 \swarrow \; \nearrow switch_1$$

$$Station \qquad\qquad Idle$$

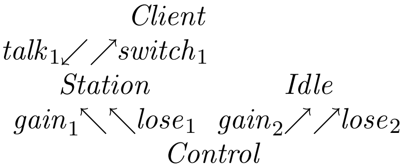
$$gain_1 \nwarrow \; \nwarrow lose_1 \quad gain_2 \nearrow \; \nearrow lose_2$$

$$Control$$

Scenario:

- client devices moving around (phones, PCs, sensors, ...)
- each radioconnected 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

- 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 II

- 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 II

- 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 II

- 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

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

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

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

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

Mobile Clients III

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

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

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

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

Outline

1 Repetition: Modeling Mobile Concurrent Systems

2 Another Example: Mobile Clients

3 The Monadic π-Calculus

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 pi-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

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 pi-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

Definition 12.1 (Syntax of monadic π-calculus)

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

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

- The set P^π of π-calculus process expressions is defined by the following syntax:

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

(where I finite, $x \in N$)

Conventions:

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

Syntax of the Monadic π-Calculus

Definition 12.1 (Syntax of monadic π-calculus)

- Let $N = \{a, b, c \ldots, x, y, z, \ldots\}$ 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 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:

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

Syntax of the Monadic π-Calculus

Definition 12.1 (Syntax of monadic π-calculus)

- Let $N = \{a, b, c \ldots, x, y, z, \ldots\}$ 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 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:

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

Syntax of the Monadic π-Calculus

Definition 12.1 (Syntax of monadic π-calculus)

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

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

- The set P^π of π-calculus process expressions is defined by the following syntax:

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

(where I finite, $x \in N$)

Conventions:

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

Free and Bound Names

Definition 12.2 (Free and bound names)

- The input prefix $x(y)$ and the restriction $\mathsf{new}\, y\, P$ both bind y.
- Every other occurrrence 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 $P \in P^\pi$ is denoted by $bn(P)/fn(P)$, resp.

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

Free and Bound Names

Definition 12.2 (Free and bound names)

- The input prefix $x(y)$ and the restriction $\mathsf{new}\, y\, P$ both bind y.
- Every other occurrrence 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 $P \in P^{\pi}$ is denoted by $bn(P)/fn(P)$, resp.

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

Free and Bound Names

Definition 12.2 (Free and bound names)

- The input prefix $x(y)$ and the restriction $\mathsf{new}\, y\, P$ both bind y.
- Every other occurrrence 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 $P \in P^{\pi}$ is denoted by $bn(P)/fn(P)$, resp.

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

Free and Bound Names

Definition 12.2 (Free and bound names)

- The input prefix $x(y)$ and the restriction $\mathsf{new}\, y\, P$ both bind y.
- Every other occurrrence 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 $P \in P^\pi$ is denoted by $bn(P)/fn(P)$, resp.

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

Structural Congruence I

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

Definition 12.3 (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)

Structural Congruence I

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

Definition 12.3 (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)

Structural Congruence I

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

Definition 12.3 (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)

Structural Congruence I

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

Definition 12.3 (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)

Structural Congruence I

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

Definition 12.3 (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)

Structural Congruence I

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

Definition 12.3 (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)

Structural Congruence II

Corollary 12.4 (Structural congruence)

$\equiv$ is a congruence relation on P^π, i.e., if $P \equiv Q$ then

1. $\pi.P + R \equiv \pi.Q + R$
2. $P \parallel R \equiv Q \parallel R$ *and* $R \parallel P \equiv R \parallel Q$
3. $\mathsf{new}\,x\,P \equiv \mathsf{new}\,x\,Q$
4. $!P \equiv !Q$

Proof.

apply operations and equations for $\equiv$ within respective contexts □

Structural Congruence II

Corollary 12.4 (Structural congruence)

$\equiv$ is a congruence relation on P^π, i.e., if $P \equiv Q$ then

1. $\pi.P + R \equiv \pi.Q + R$
2. $P \parallel R \equiv Q \parallel R$ *and* $R \parallel P \equiv R \parallel Q$
3. $\mathsf{new}\, x\, P \equiv \mathsf{new}\, x\, Q$
4. $!P \equiv !Q$

Proof.

apply operations and equations for $\equiv$ within respective contexts □

A Standard Form

Theorem 12.5 (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 $k = 0$: no restriction)

Proof.

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

A Standard Form

Theorem 12.5 (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 $k = 0$: no restriction)

Proof.

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

The Reaction Relation

Thanks to Theorem 12.5, only processes in standard form need to be considered for defining the operational semantics:

Definition 12.6

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.)