

Modeling Concurrent and Probabilistic Systems

Lecture 14: Bisimulation in the 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

- 1 Repetition: Encoding Recursive Process Calls
- 2 The Commitment Relation
- 3 Strong Bisimulation

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

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

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

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

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

Conventions:

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

Repetition: The Reaction Relation

Definition

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

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

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

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

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

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

$(P[y \mapsto z])$ replaces every free occurrence of y in P by z .

In (React), the pair $(x(y), \bar{x}\langle z \rangle)$ is called a **redex**.)

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

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

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

The **encoding**

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

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 $\bar{a}\langle\vec{w}\rangle$.
- ③ Replace Q by $Q' := \text{new } a (\hat{Q} \parallel !a(\vec{x}).\hat{P}_A)$.

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

Example

- One-place buffer: $B(in, out) = in(x).\overline{out}\langle x \rangle.B(in, out)$
- Main process: $Q := \overline{in}\langle y \rangle \parallel B(in, out) \parallel out(z)$
- Encoding:

$$\begin{aligned} Q' &:= \text{new } b \left(\overline{in}\langle y \rangle \parallel \bar{b}(in, out) \parallel out(z) \parallel \right. \\ &\quad \left. \underbrace{!b(in, out).in(x).\overline{out}\langle x \rangle.\bar{b}(in, out)}_{=:P} \right) \\ &\equiv \\ &\text{new } b \left(\overline{in}\langle y \rangle \parallel \bar{b}(in, out) \parallel out(z) \parallel b(in, out).in(x).\overline{out}\langle x \rangle.\bar{b}(in, out) \parallel P \right) \\ &\quad \downarrow \\ &\text{new } b \left(\overline{in}\langle y \rangle \parallel out(z) \parallel \bar{b}(in, out).in(x).\overline{out}\langle x \rangle.\bar{b}(in, out) \parallel P \right) \\ &\quad \downarrow \\ &\text{new } b \left(out(z) \parallel \overline{out}\langle y \rangle.\bar{b}(in, out) \parallel P \right) \\ &\quad \downarrow \\ &\text{new } b \left(\bar{b}(in, out) \parallel P \right) \\ &\quad \downarrow \\ &\text{new } b \left(out(z) \parallel \overline{out}\langle y \rangle.\bar{b}(in, out) \parallel P \right) \end{aligned}$$

Encoding Recursive Process Calls II

Example

- One-place buffer: $B(in, out) = in(x).\overline{out}\langle x \rangle.B(in, out)$
- Main process: $Q := \overline{in}\langle y \rangle \parallel B(in, out) \parallel out(z)$
- Encoding:

$$\begin{aligned} Q' &:= \text{new } b \left(\overline{in}\langle y \rangle \parallel \bar{b}(in, out) \parallel out(z) \parallel \right. \\ &\quad \left. \underbrace{!b(in, out).in(x).\overline{out}\langle x \rangle.\bar{b}(in, out)}_{=:P} \right) \\ &\equiv \\ &\text{new } b \left(\overline{in}\langle y \rangle \parallel \bar{b}(in, out) \parallel out(z) \parallel b(in, out).in(x).\overline{out}\langle x \rangle.\bar{b}(in, out) \parallel P \right) \\ &\quad \downarrow \\ &\text{new } b \left(\overline{in}\langle y \rangle \parallel out(z) \parallel \bar{b}(in, out).in(x).\overline{out}\langle x \rangle.\bar{b}(in, out) \parallel P \right) \\ &\quad \downarrow \\ &\text{new } b \left(out(z) \parallel \overline{out}\langle y \rangle.\bar{b}(in, out) \parallel P \right) \\ &\quad \downarrow \\ &\text{new } b \left(\bar{b}(in, out) \parallel P \right) \\ &\quad \downarrow \\ &\text{new } b \left(\overline{in}\langle y \rangle \parallel \overline{out}\langle z \rangle.\bar{b}(in, out) \parallel P \right) \end{aligned}$$

Example

- One-place buffer: $B(in, out) = in(x).\overline{out}\langle x \rangle.B(in, out)$
- Main process: $Q := \overline{in}\langle y \rangle \parallel B(in, out) \parallel out(z)$
- Encoding:

$$\begin{aligned} Q' := \mathbf{new} \, b \, (\overline{in}\langle y \rangle \parallel \overline{b}\langle in, out \rangle \parallel out(z) \parallel \\ \underbrace{\mathbf{!}b(in, out).in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle}_{=:P}) \\ \equiv \\ \mathbf{new} \, b \, (\overline{in}\langle y \rangle \parallel out(z) \parallel \overline{b}\langle in, out \rangle.in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle \parallel P) \\ \downarrow \\ \mathbf{new} \, b \, (\overline{in}\langle y \rangle \parallel out(z) \parallel \overline{in}(x).\overline{out}(x).\overline{b}\langle in, out \rangle \parallel P) \\ \downarrow \\ \mathbf{new} \, b \, (out(z) \parallel \overline{out}(x).\overline{b}\langle in, out \rangle \parallel P) \\ \downarrow \\ \mathbf{new} \, b \, (\overline{b}\langle in, out \rangle \parallel P) \\ \downarrow \\ \mathbf{new} \, b \, (out(z) \parallel \overline{b}\langle in, out \rangle \parallel P) \end{aligned}$$

Encoding Recursive Process Calls II

Example

- One-place buffer: $B(in, out) = in(x).\overline{out}\langle x \rangle.B(in, out)$
- Main process: $Q := \overline{in}\langle y \rangle \parallel B(in, out) \parallel out(z)$
- Encoding:

$$\begin{aligned} Q' := \mathbf{new} \ b \ (\overline{in}\langle y \rangle \parallel \overline{b}\langle in, out \rangle \parallel out(z) \parallel \\ \underbrace{!b(in, out).in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle}_{=:P}) \\ \equiv \\ \mathbf{new} \ b \ (\overline{in}\langle y \rangle \parallel \overline{b}\langle in, out \rangle \parallel out(z) \parallel \mathbf{red} \ b(in, out).in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle \parallel P) \end{aligned}$$



Example

- One-place buffer: $B(in, out) = in(x).\overline{out}\langle x \rangle.B(in, out)$
- Main process: $Q := \overline{in}\langle y \rangle \parallel B(in, out) \parallel out(z)$
- Encoding:

$$\begin{aligned} Q' &:= \mathbf{new} \, b \, (\overline{in}\langle y \rangle \parallel \overline{b}\langle in, out \rangle \parallel out(z) \parallel \\ &\quad \underbrace{!b(in, out).in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle}_{=:P}) \\ &\equiv \\ \mathbf{new} \, b \, (\overline{in}\langle y \rangle \parallel \overline{b}\langle in, out \rangle \parallel out(z) \parallel b(in, out).in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle \parallel P) \\ &\quad \downarrow \\ \mathbf{new} \, b \, (\overline{in}\langle y \rangle \parallel out(z) \parallel \color{red}{in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle \parallel P}) \\ &\quad \downarrow \\ &\quad \color{green}{\mathbf{new} \, b \, (\overline{in}\langle y \rangle \parallel \overline{b}\langle in, out \rangle \parallel P)} \\ &\quad \downarrow \\ &\quad \color{green}{\mathbf{new} \, b \, (\overline{in}\langle y \rangle \parallel \overline{b}\langle in, out \rangle \parallel P)} \end{aligned}$$

Example

- One-place buffer: $B(in, out) = in(x).\overline{out}\langle x \rangle.B(in, out)$
- Main process: $Q := \overline{in}\langle y \rangle \parallel B(in, out) \parallel out(z)$
- Encoding:

$$Q' := \mathbf{new} \ b \ (\overline{in}\langle y \rangle \parallel \overline{b}\langle in, out \rangle \parallel out(z) \parallel \underbrace{!b(in, out).in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle}_{=:P})$$
$$\equiv$$
$$\mathbf{new} \ b \ (\overline{in}\langle y \rangle \parallel \overline{b}\langle in, out \rangle \parallel out(z) \parallel b(in, out).in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle \parallel P)$$

$$\downarrow$$
$$\mathbf{new} \ b \ (\overline{in}\langle y \rangle \parallel out(z) \parallel in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle \parallel P)$$

$$\downarrow$$
$$\mathbf{new} \ b \ (\textcolor{red}{out}(z) \parallel \textcolor{red}{\overline{out}\langle y \rangle}.\overline{b}\langle in, out \rangle \parallel P)$$



Example

- One-place buffer: $B(in, out) = in(x).\overline{out}\langle x \rangle.B(in, out)$
- Main process: $Q := \overline{in}\langle y \rangle \parallel B(in, out) \parallel out(z)$
- Encoding:

$$\begin{aligned} Q' &:= \mathbf{new} \, b \, (\overline{in}\langle y \rangle \parallel \overline{b}\langle in, out \rangle \parallel out(z) \parallel \\ &\quad \underbrace{!b(in, out).in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle}_{=:P}) \\ &\equiv \\ \mathbf{new} \, b \, (\overline{in}\langle y \rangle \parallel \overline{b}\langle in, out \rangle \parallel out(z) \parallel b(in, out).in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle \parallel P) \\ &\quad \downarrow \\ \mathbf{new} \, b \, (\overline{in}\langle y \rangle \parallel out(z) \parallel in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle \parallel P) \\ &\quad \downarrow \\ \mathbf{new} \, b \, (out(z) \parallel \overline{out}\langle y \rangle.\overline{b}\langle in, out \rangle \parallel P) \\ &\quad \downarrow \\ \mathbf{new} \, b \, (\overline{b}\langle in, out \rangle \parallel P) \\ &\quad \downarrow \end{aligned}$$

Example

- One-place buffer: $B(in, out) = in(x).\overline{out}\langle x \rangle.B(in, out)$
- Main process: $Q := \overline{in}\langle y \rangle \parallel B(in, out) \parallel out(z)$
- Encoding:

$$\begin{aligned}
 Q' &:= \mathbf{new} \, b \, (\overline{in}\langle y \rangle \parallel \overline{b}\langle in, out \rangle \parallel out(z) \parallel \\
 &\quad \underbrace{!b(in, out).in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle}_{=:P}) \\
 &\equiv \\
 \mathbf{new} \, b \, (\overline{in}\langle y \rangle \parallel \overline{b}\langle in, out \rangle \parallel out(z) \parallel b(in, out).in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle \parallel P) \\
 &\quad \downarrow \\
 \mathbf{new} \, b \, (\overline{in}\langle y \rangle \parallel out(z) \parallel in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle \parallel P) \\
 &\quad \downarrow \\
 \mathbf{new} \, b \, (out(z) \parallel \overline{out}\langle y \rangle.\overline{b}\langle in, out \rangle \parallel P) \\
 &\quad \downarrow \\
 \mathbf{new} \, b \, (\overline{b}\langle in, out \rangle \parallel P) \\
 &\quad \downarrow \\
 \mathbf{new} \, b \, (in(x).\overline{out}\langle x \rangle.\overline{b}\langle in, out \rangle \parallel P)
 \end{aligned}$$

- ① Repetition: Encoding Recursive Process Calls
- ② The Commitment Relation
- ③ Strong Bisimulation

Commitments

- **Goal:** establish **equivalence relations** between π -calculus processes (e.g., for establishing the correctness of encodings)
- **But:** reaction relation $\longrightarrow \subseteq P^\pi \times P^\pi$ is **too coarse**
- For example, $x(y).\text{nil}$ and $\bar{x}\langle z \rangle.\text{nil}$ both have no reactions
 \implies bisimilar w.r.t. \longrightarrow , but different **reaction capabilities**
- **Solution:** introduce **commitments** in π -calculus, corresponding to transitions in CCS
 - CCS: $P \parallel Q = (a.P' + \dots) \parallel (\bar{a}.Q' + \dots) \xrightarrow{\tau} P' \parallel Q'$

$$\begin{array}{ccc} \downarrow a & & \downarrow \bar{a} \\ P' & & Q' \end{array}$$
commitments
 - π -calculus:

$$P \parallel Q = (x(\vec{y}).P' + \dots) \parallel (\bar{x}\langle \vec{z} \rangle.Q' + \dots) \longrightarrow P'[\vec{y} \mapsto \vec{z}] \parallel Q'$$

$$\begin{array}{ccc} \downarrow x & & \downarrow \bar{x} \\ (\vec{y}).P' & & \langle \vec{z} \rangle.Q' \end{array}$$
commitments

abstraction
concretion

Commitments

- **Goal:** establish **equivalence relations** between π -calculus processes (e.g., for establishing the correctness of encodings)
- **But:** reaction relation $\longrightarrow \subseteq P^\pi \times P^\pi$ is **too coarse**
- For example, $x(y).\text{nil}$ and $\bar{x}\langle z \rangle.\text{nil}$ both have no reactions
 \implies bisimilar w.r.t. \longrightarrow , but different **reaction capabilities**
- **Solution:** introduce **commitments** in π -calculus, corresponding to transitions in CCS
 - CCS: $P \parallel Q = (a.P' + \dots) \parallel (\bar{a}.Q' + \dots) \xrightarrow{\tau} P' \parallel Q'$
 $\downarrow a$ $\downarrow \bar{a}$ commitments
 P' Q'
 - π -calculus:
 $P \parallel Q = (x(\vec{y}).P' + \dots) \parallel (\bar{x}\langle \vec{z} \rangle.Q' + \dots) \longrightarrow P'[\vec{y} \mapsto \vec{z}] \parallel Q'$
 $\downarrow x$ $\downarrow \bar{x}$ commitments
 $(\vec{y}).P'$ $\langle \vec{z} \rangle.Q'$
abstraction concretion

Commitments

- **Goal:** establish **equivalence relations** between π -calculus processes (e.g., for establishing the correctness of encodings)
- **But:** reaction relation $\longrightarrow \subseteq P^\pi \times P^\pi$ is **too coarse**
- For example, $x(y).\text{nil}$ and $\bar{x}\langle z \rangle.\text{nil}$ both have no reactions
 \implies bisimilar w.r.t. \longrightarrow , but different **reaction capabilities**
- **Solution:** introduce **commitments** in π -calculus, corresponding to transitions in CCS

- CCS: $P \parallel Q = (a.P' + \dots) \parallel (\bar{a}.Q' + \dots) \xrightarrow{\tau} P' \parallel Q'$
 $\quad \downarrow a \qquad \qquad \downarrow \bar{a} \qquad \text{commitments}$
 $\quad P' \qquad \qquad \qquad Q'$

- π -calculus:

$$P \parallel Q = (x(\vec{y}).P' + \dots) \parallel (\bar{x}\langle \vec{z} \rangle.Q' + \dots) \longrightarrow P'[\vec{y} \mapsto \vec{z}] \parallel Q'$$

$\downarrow x \qquad \qquad \downarrow \bar{x} \qquad \text{commitments}$

$(\vec{y}).P' \qquad \qquad \langle \vec{z} \rangle.Q'$

abstraction **concretion**

Commitments

- **Goal:** establish equivalence relations between π -calculus processes (e.g., for establishing the correctness of encodings)
- **But:** reaction relation $\longrightarrow \subseteq P^\pi \times P^\pi$ is **too coarse**
- For example, $x(y).\text{nil}$ and $\bar{x}\langle z \rangle.\text{nil}$ both have no reactions
 \implies bisimilar w.r.t. \longrightarrow , but different **reaction capabilities**
- **Solution:** introduce **commitments** in π -calculus, corresponding to transitions in CCS
 - CCS: $P \parallel Q = (a.P' + \dots) \parallel (\bar{a}.Q' + \dots) \xrightarrow{\tau} P' \parallel Q'$

$$\begin{array}{ccc} \downarrow a & & \downarrow \bar{a} \\ P' & & Q' \end{array}$$
commitments
 - π -calculus:

$$P \parallel Q = (x(\vec{y}).P' + \dots) \parallel (\bar{x}\langle \vec{z} \rangle.Q' + \dots) \longrightarrow P'[\vec{y} \mapsto \vec{z}] \parallel Q'$$

$$\begin{array}{ccc} \downarrow x & & \downarrow \bar{x} \\ (\vec{y}).P' & & \langle \vec{z} \rangle.Q' \end{array}$$
commitments

abstraction
concretion

Definition 14.1

- An **abstraction** of arity $n \in \mathbb{N}$ is of the form $(\vec{x}).P$, where $\vec{x} = (x_1, \dots, x_n)$ and $P \in P^\pi$.
- A **concretion** of arity $n \in \mathbb{N}$ is of the form $\text{new } \vec{x} \langle \vec{y} \rangle.P$, where $\vec{y} = (y_1, \dots, y_n)$, $\vec{x} \subseteq \vec{y}$, and $P \in P^\pi$.
- An **agent** is an abstraction or a concretion (notation: A^π).

Remarks:

- We use
 - F, G to denote abstractions,
 - C, D to denote concretions,
 - $A, B \in A^\pi$ to denote agents
- Note: a process $P \in P^\pi$ is both an abstraction and a concretion of arity 0
- $\equiv/fn/bn$ also extends to agents
- Guarded sum now considered as $\sum \alpha_i A_i$ where $\alpha_i \in N \cup \overline{N} \cup \{\tau\}$

Definition 14.1

- An **abstraction** of arity $n \in \mathbb{N}$ is of the form $(\vec{x}).P$, where $\vec{x} = (x_1, \dots, x_n)$ and $P \in P^\pi$.
- A **concretion** of arity $n \in \mathbb{N}$ is of the form $\text{new } \vec{x} \langle \vec{y} \rangle.P$, where $\vec{y} = (y_1, \dots, y_n)$, $\vec{x} \subseteq \vec{y}$, and $P \in P^\pi$.
- An **agent** is an abstraction or a concretion (notation: A^π).

Remarks:

- We use
 - F, G to denote abstractions,
 - C, D to denote concretions,
 - $A, B \in A^\pi$ to denote agents
- Note: a process $P \in P^\pi$ is both an abstraction and a concretion of arity 0
- $\equiv/fn/bn$ also extends to agents
- Guarded sum now considered as $\sum \alpha_i A_i$ where $\alpha_i \in N \cup \bar{N} \cup \{\tau\}$

Definition 14.1

- An **abstraction** of arity $n \in \mathbb{N}$ is of the form $(\vec{x}).P$, where $\vec{x} = (x_1, \dots, x_n)$ and $P \in P^\pi$.
- A **concretion** of arity $n \in \mathbb{N}$ is of the form $\text{new } \vec{x} \langle \vec{y} \rangle.P$, where $\vec{y} = (y_1, \dots, y_n)$, $\vec{x} \subseteq \vec{y}$, and $P \in P^\pi$.
- An **agent** is an abstraction or a concretion (notation: A^π).

Remarks:

- We use
 - F, G to denote abstractions,
 - C, D to denote concretions,
 - $A, B \in A^\pi$ to denote agents
- Note: a process $P \in P^\pi$ is both an abstraction and a concretion of arity 0
- $\equiv/fn/bn$ also extends to agents
- Guarded sum now considered as $\sum \alpha_i A_i$ where $\alpha_i \in N \cup \overline{N} \cup \{\tau\}$

Definition 14.1

- An **abstraction** of arity $n \in \mathbb{N}$ is of the form $(\vec{x}).P$, where $\vec{x} = (x_1, \dots, x_n)$ and $P \in P^\pi$.
- A **concretion** of arity $n \in \mathbb{N}$ is of the form $\text{new } \vec{x} \langle \vec{y} \rangle.P$, where $\vec{y} = (y_1, \dots, y_n)$, $\vec{x} \subseteq \vec{y}$, and $P \in P^\pi$.
- An **agent** is an abstraction or a concretion (notation: A^π).

Remarks:

- We use
 - F, G to denote abstractions,
 - C, D to denote concretions,
 - $A, B \in A^\pi$ to denote agents
- Note: a process $P \in P^\pi$ is both an abstraction and a concretion of arity 0
- $\equiv/fn/bn$ also extends to agents
- Guarded sum now considered as $\sum \alpha_i A_i$ where $\alpha_i \in N \cup \overline{N} \cup \{\tau\}$

Definition 14.2

The **application** $F @ C$ (where F and C are of equal arity) is defined as follows, assuming $\vec{z} \cap fn((\vec{x}).P) = \emptyset$:

$$(\vec{x}).P @ \text{new } \vec{z} \langle \vec{y} \rangle . Q := \text{new } \vec{z} (P[\vec{x} \mapsto \vec{y}] \parallel Q)$$

Remarks:

- The (React) rule can now be represented as

$$\text{(React)} \frac{}{(xF + P) \parallel (\bar{x}C + Q) \longrightarrow F @ C}$$

- We add the following equations for structural congruence (assuming $z \notin \vec{x}$ and $\vec{x} \cap fn(Q) = \emptyset$):

$$\begin{aligned} \text{new } z ((\vec{x}).P) &\equiv (\vec{x}).\text{new } z P \\ ((\vec{x}).P) \parallel Q &\equiv (\vec{x}).(P \parallel Q) \end{aligned}$$

Definition 14.2

The **application** $F @ C$ (where F and C are of equal arity) is defined as follows, assuming $\vec{z} \cap fn((\vec{x}).P) = \emptyset$:

$$(\vec{x}).P @ \text{new } \vec{z} \langle \vec{y} \rangle . Q := \text{new } \vec{z} (P[\vec{x} \mapsto \vec{y}] \parallel Q)$$

Remarks:

- The (React) rule can now be represented as

$$\text{(React)} \frac{}{(xF + P) \parallel (\bar{x}C + Q) \longrightarrow F @ C}$$

- We add the following equations for structural congruence (assuming $z \notin \vec{x}$ and $\vec{x} \cap fn(Q) = \emptyset$):

$$\begin{aligned} \text{new } z ((\vec{x}).P) &\equiv (\vec{x}).\text{new } z P \\ ((\vec{x}).P) \parallel Q &\equiv (\vec{x}).(P \parallel Q) \end{aligned}$$

Commitment Relation

Definition 14.3

The **commitment relation** $\longrightarrow \subseteq A^\pi \times (N \cup \bar{N} \cup \{\tau\}) \times A^\pi$ is generated by the following rules:

$$\begin{array}{c} (\text{SUM}) \frac{}{\alpha.A + M \xrightarrow{\alpha} A} \quad (\text{REACT}) \frac{P \xrightarrow{x} F \quad Q \xrightarrow{\bar{x}} C}{P \parallel Q \xrightarrow{\tau} F @ C} \\ (\text{PAR}) \frac{P \xrightarrow{\alpha} A}{P \parallel Q \xrightarrow{\alpha} A \parallel Q} \quad (\text{RES}) \frac{P \xrightarrow{\alpha} A \quad \alpha \notin \{x, \bar{x}\}}{\text{new } x \, P \xrightarrow{\alpha} \text{new } x \, A} \\ (\text{STRUCT}) \frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q' \equiv Q}{P \xrightarrow{\alpha} Q} \end{array}$$

Example 14.4

$$y(z).P \parallel \text{new } x \, (\bar{y}\langle x \rangle.Q + \bar{w}\langle v \rangle.R) \xrightarrow{\tau} \text{new } x \, (P[z \mapsto x] \parallel Q)$$

(if $x \notin \text{fn}(P)$; on the board)

Commitment Relation

Definition 14.3

The **commitment relation** $\longrightarrow \subseteq A^\pi \times (N \cup \overline{N} \cup \{\tau\}) \times A^\pi$ is generated by the following rules:

$$\begin{array}{c} (\text{SUM}) \frac{}{\alpha.A + M \xrightarrow{\alpha} A} \quad (\text{REACT}) \frac{P \xrightarrow{x} F \quad Q \xrightarrow{\bar{x}} C}{P \parallel Q \xrightarrow{\tau} F @ C} \\ (\text{PAR}) \frac{P \xrightarrow{\alpha} A}{P \parallel Q \xrightarrow{\alpha} A \parallel Q} \quad (\text{RES}) \frac{P \xrightarrow{\alpha} A \quad \alpha \notin \{x, \bar{x}\}}{\text{new } x \, P \xrightarrow{\alpha} \text{new } x \, A} \\ (\text{STRUCT}) \frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q' \equiv Q}{P \xrightarrow{\alpha} Q} \end{array}$$

Example 14.4

$$y(z).P \parallel \text{new } x \, (\bar{y}\langle x \rangle.Q + \bar{w}\langle v \rangle.R) \xrightarrow{\tau} \text{new } x \, (P[z \mapsto x] \parallel Q)$$

(if $x \notin fn(P)$; on the board)

- 1 Repetition: Encoding Recursive Process Calls
- 2 The Commitment Relation
- 3 Strong Bisimulation

Problem: the target $A \in A^\pi$ of a commitment $P \xrightarrow{\alpha} A$ (where $\alpha \neq \tau$) is not necessarily a process, but an abstraction or a concretion
⇒ extend process equivalences to **agent equivalences**

Definition 14.5

Let $\rho \subseteq P^\pi \times P^\pi$ be a binary relation on processes. The **extension** of ρ to agents, $\rho \subseteq A^\pi \times A^\pi$, is defined by

$F\rho G : \iff$ for all $\vec{y} : (F @ \langle \vec{y} \rangle.\text{nil})\rho(G @ \langle \vec{y} \rangle.\text{nil})$

$C\rho D : \iff$ there exist $P\rho Q$ such that $C \equiv \text{new } \vec{z} \langle \vec{y} \rangle.P$ and
 $D \equiv \text{new } \vec{z} \langle \vec{y} \rangle.Q$

Problem: the target $A \in A^\pi$ of a commitment $P \xrightarrow{\alpha} A$ (where $\alpha \neq \tau$) is not necessarily a process, but an abstraction or a concretion
⇒ extend process equivalences to **agent equivalences**

Definition 14.5

Let $\rho \subseteq P^\pi \times P^\pi$ be a binary relation on processes. The **extension** of ρ to agents, $\rho \subseteq A^\pi \times A^\pi$, is defined by

$$F\rho G : \iff \text{for all } \vec{y} : (F @ \langle \vec{y} \rangle.\text{nil})\rho(G @ \langle \vec{y} \rangle.\text{nil})$$

$$C\rho D : \iff \text{there exist } P\rho Q \text{ such that } C \equiv \text{new } \vec{z} \langle \vec{y} \rangle.P \text{ and} \\ D \equiv \text{new } \vec{z} \langle \vec{y} \rangle.Q$$

Strong Bisimulation II

Definition 14.6 (Strong bisimulation)

A relation $\rho \subseteq P^\pi \times P^\pi$ is called a **strong bisimulation** if $P\rho Q$ implies

- ① $P \xrightarrow{\alpha} A \implies \text{ex. } B \in A^\pi \text{ such that } Q \xrightarrow{\alpha} B \text{ and } A\rho B$
- ② $Q \xrightarrow{\alpha} B \implies \text{ex. } A \in A^\pi \text{ such that } P \xrightarrow{\alpha} A \text{ and } A\rho B$

Two agents $A, B \in A^\pi$ are called **strongly bisimilar** (notation: $A \sim B$) if $A\rho B$ for some strong bisimulation ρ .

Lemma 14.7

- ① \sim is a strong bisimulation (the largest one).
- ② \sim is an equivalence relation.

Proof.

similar to CCS case (Theorem 4.2)



Example 14.8

on the board

Strong Bisimulation II

Definition 14.6 (Strong bisimulation)

A relation $\rho \subseteq P^\pi \times P^\pi$ is called a **strong bisimulation** if $P\rho Q$ implies

- ① $P \xrightarrow{\alpha} A \implies \text{ex. } B \in A^\pi \text{ such that } Q \xrightarrow{\alpha} B \text{ and } A\rho B$
- ② $Q \xrightarrow{\alpha} B \implies \text{ex. } A \in A^\pi \text{ such that } P \xrightarrow{\alpha} A \text{ and } A\rho B$

Two agents $A, B \in A^\pi$ are called **strongly bisimilar** (notation: $A \sim B$) if $A\rho B$ for some strong bisimulation ρ .

Lemma 14.7

- ① \sim is a strong bisimulation (the largest one).
- ② \sim is an equivalence relation.

Proof.

similar to CCS case (Theorem 4.2) □

Example 14.8

on the board

Strong Bisimulation II

Definition 14.6 (Strong bisimulation)

A relation $\rho \subseteq P^\pi \times P^\pi$ is called a **strong bisimulation** if $P\rho Q$ implies

- ① $P \xrightarrow{\alpha} A \implies \text{ex. } B \in A^\pi \text{ such that } Q \xrightarrow{\alpha} B \text{ and } A\rho B$
- ② $Q \xrightarrow{\alpha} B \implies \text{ex. } A \in A^\pi \text{ such that } P \xrightarrow{\alpha} A \text{ and } A\rho B$

Two agents $A, B \in A^\pi$ are called **strongly bisimilar** (notation: $A \sim B$) if $A\rho B$ for some strong bisimulation ρ .

Lemma 14.7

- ① \sim is a strong bisimulation (the largest one).
- ② \sim is an equivalence relation.

Proof.

similar to CCS case (Theorem 4.2)



Example 14.8

on the board

Strong Bisimulation II

Definition 14.6 (Strong bisimulation)

A relation $\rho \subseteq P^\pi \times P^\pi$ is called a **strong bisimulation** if $P\rho Q$ implies

- ① $P \xrightarrow{\alpha} A \implies \text{ex. } B \in A^\pi \text{ such that } Q \xrightarrow{\alpha} B \text{ and } A\rho B$
- ② $Q \xrightarrow{\alpha} B \implies \text{ex. } A \in A^\pi \text{ such that } P \xrightarrow{\alpha} A \text{ and } A\rho B$

Two agents $A, B \in A^\pi$ are called **strongly bisimilar** (notation: $A \sim B$) if $A\rho B$ for some strong bisimulation ρ .

Lemma 14.7

- ① \sim is a strong bisimulation (the largest one).
- ② \sim is an equivalence relation.

Proof.

similar to CCS case (Theorem 4.2)



Example 14.8

on the board

- **Problem:** strong bisimulation is **not** a π -calculus process congruence

- Not preserved by **input prefix**

- **Example:** $P := \bar{x} \parallel y$, $Q := \bar{x}.y + y.\bar{x}$

- $P \sim Q$ (obvious) but

- $P[y \mapsto x] \not\sim Q[y \mapsto x]$

- (since $P[y \mapsto x] = \bar{x} \parallel x \xrightarrow{\tau} \text{nil}$ and $Q[y \mapsto x] = \bar{x}.x + x.\bar{x} \not\xrightarrow{\tau}$)

$\implies (y).P @ \langle x \rangle.\text{nil} \not\sim (y).Q @ \langle x \rangle.\text{nil}$ (Def. 14.2 of $@$)

$\implies (y).P \not\sim (y).Q$ (Def. 14.5 of \sim for abstractions)

- **Problem:** strong bisimulation is **not** a π -calculus process congruence
- Not preserved by **input prefix**
- **Example:** $P := \bar{x} \parallel y$, $Q := \bar{x}.y + y.\bar{x}$
 - $P \sim Q$ (obvious) but
 - $P[y \mapsto x] \not\sim Q[y \mapsto x]$
(since $P[y \mapsto x] = \bar{x} \parallel x \xrightarrow{\tau} \text{nil}$ and $Q[y \mapsto x] = \bar{x}.x + x.\bar{x} \not\xrightarrow{\tau}$)

$$\implies (y).P @ \langle x \rangle.\text{nil} \not\sim (y).Q @ \langle x \rangle.\text{nil} \text{ (Def. 14.2 of @)}$$

$$\implies (y).P \not\sim (y).Q \text{ (Def. 14.5 of } \sim \text{ for abstractions)}$$

- **Problem:** strong bisimulation is **not** a π -calculus process congruence
- Not preserved by **input prefix**
- **Example:** $P := \bar{x} \parallel y$, $Q := \bar{x}.y + y.\bar{x}$
 - $P \sim Q$ (obvious) but
 - $P[y \mapsto x] \not\sim Q[y \mapsto x]$
(since $P[y \mapsto x] = \bar{x} \parallel x \xrightarrow{\tau} \text{nil}$ and $Q[y \mapsto x] = \bar{x}.x + x.\bar{x} \not\xrightarrow{\tau}$)

$\implies (y).P @ \langle x \rangle.\text{nil} \not\sim (y).Q @ \langle x \rangle.\text{nil}$ (Def. 14.2 of $@$)

$\implies (y).P \not\sim (y).Q$ (Def. 14.5 of \sim for abstractions)

- **Problem:** strong bisimulation is **not** a π -calculus process congruence

- Not preserved by **input prefix**

- **Example:** $P := \bar{x} \parallel y$, $Q := \bar{x}.y + y.\bar{x}$

- $P \sim Q$ (obvious) but

- $P[y \mapsto x] \not\sim Q[y \mapsto x]$

(since $P[y \mapsto x] = \bar{x} \parallel x \xrightarrow{\tau} \text{nil}$ and $Q[y \mapsto x] = \bar{x}.x + x.\bar{x} \not\xrightarrow{\tau}$)

$\implies (y).P @ \langle x \rangle.\text{nil} \not\sim (y).Q @ \langle x \rangle.\text{nil}$ (Def. 14.2 of $@$)

$\implies (y).P \not\sim (y).Q$ (Def. 14.5 of \sim for abstractions)

Definition 14.9 (Strong congruence)

Two processes $P, Q \in P^\pi$ are called **strongly congruent** ($P \sim Q$) if $P\sigma \dot{\sim} Q\sigma$ for every substitution $\sigma : N \rightarrow N$.

Lemma 14.10

\sim is the largest congruence in $\dot{\sim}$.

Proof.

omitted



Definition 14.9 (Strong congruence)

Two processes $P, Q \in P^\pi$ are called **strongly congruent** ($P \sim Q$) if $P\sigma \dot{\sim} Q\sigma$ for every substitution $\sigma : N \rightarrow N$.

Lemma 14.10

\sim is the largest congruence in $\dot{\sim}$.

Proof.

omitted



Definition 14.9 (Strong congruence)

Two processes $P, Q \in P^\pi$ are called **strongly congruent** ($P \sim Q$) if $P\sigma \dot{\sim} Q\sigma$ for every substitution $\sigma : N \rightarrow N$.

Lemma 14.10

\sim is the largest congruence in $\dot{\sim}$.

Proof.

omitted

