



# Concurrency Theory

Winter Semester 2017/18

Lecture 9: Variations of  $\pi$ -Calculus

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

<http://moves.rwth-aachen.de/teaching/ws-1718/ct/>

# Recap: The Monadic $\pi$ -Calculus

## Syntax of the Monadic $\pi$ -Calculus

### Definition (Syntax of monadic $\pi$ -Calculus)

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

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

- The set  $\text{Proc}^\pi$  of  **$\pi$ -Calculus process expressions** is defined by the following syntax:

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

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

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

## Recap: The Monadic $\pi$ -Calculus

---

### Structural Congruence

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

#### Definition (Structural congruence)

$P, Q \in \text{Prc}^\pi$  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 ( $\alpha$ -conversion)
2. reordering of terms in a summation (commutativity of  $+$ )
3.  $P \parallel Q \equiv Q \parallel P$ ,  $P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$ ,  $P \parallel \text{nil} \equiv P$  (Abelian monoid laws for  $\parallel$ )
4.  $\text{new } x \text{ nil} \equiv \text{nil}$ ,  $\text{new } x, y P \equiv \text{new } y, x P$ ,  
 $P \parallel \text{new } x Q \equiv \text{new } x (P \parallel Q)$  if  $x \notin \text{fn}(P)$  (scope extension)
5.  $!P \equiv P \parallel !P$  (unfolding)

## Recap: The Monadic $\pi$ -Calculus

---

### A Standard Form

#### Theorem (Standard form)

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

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

where each  $P_i$  is a non-empty guarded 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  $R \in \text{Proc}^\pi$  (on the board) □

## Recap: The Monadic $\pi$ -Calculus

### The Reaction Relation

Thanks to Theorem 9.3, only processes in standard form need to be considered for defining the operational semantics:

#### Definition

The **reaction relation**  $\longrightarrow \subseteq \text{Proc}^\pi \times \text{Proc}^\pi$  is generated by the rules:

$$\begin{array}{c} \text{(Tau)} \frac{}{\tau.P + Q \longrightarrow P} \\ \\ \text{(React)} \frac{}{(x(y).P + R) \parallel (\bar{x}\langle z \rangle.Q + S) \longrightarrow P[z/y] \parallel Q} \\ \\ \text{(Par)} \frac{P \longrightarrow P'}{P \parallel Q \longrightarrow P' \parallel Q} \qquad \text{(Res)} \frac{P \longrightarrow P'}{\text{new } x P \longrightarrow \text{new } x P'} \\ \\ \text{(Struct)} \frac{P \longrightarrow P'}{Q \longrightarrow Q'} \quad \text{if } P \equiv Q \text{ and } P' \equiv Q' \end{array}$$

- $P[z/y]$  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**.

# Example Reactions

## The Printer Server Revisited

### Example 9.1

1. **Printer server** (cf. Example 8.1):

$$\underbrace{\bar{b}\langle a \rangle . S'}_S \parallel \underbrace{a(e) . P'}_P \parallel \underbrace{b(c) . \bar{c}\langle d \rangle . C'}_C \longrightarrow S' \parallel P \parallel \bar{a}\langle d \rangle . C'[a/c]$$

$$S' \parallel P \parallel \bar{a}\langle d \rangle . C'[a/c] \longrightarrow S' \parallel P'[d/e] \parallel C'[a/c]$$

(on the board)

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

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

(on the board)

# Example Reactions

## Mobile Clients Revisited

### Example 9.2

- System specification (cf. Example 8.2):

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

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

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

$$\mathit{Idle}(\mathit{gain}, \mathit{lose}) = \overline{\mathit{gain}}(t, s).\mathit{Station}(t, s, \mathit{gain}, \mathit{lose})$$

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

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

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

$$L = (\mathit{talk}_i, \mathit{switch}_i, \mathit{gain}_i, \mathit{lose}_i \mid i \in \{1, 2\})$$

- Use additional reaction rule for **polyadic communication**:

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

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

## Polyadic Communication I

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

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

where  $n \in \mathbb{N}$  and all  $y_i$  distinct

- Expected **behavior**:

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

(replacement of **free** names)

- Obvious attempt for **encoding**:

$$\begin{aligned} x(y_1, \dots, y_n).P &\mapsto x(y_1) \dots x(y_n).P \\ \bar{x}\langle z_1, \dots, z_n \rangle.Q &\mapsto \bar{x}\langle z_1 \rangle \dots \bar{x}\langle z_n \rangle.Q \end{aligned}$$



## Polyadic Communication II

- But consider the following **counterexample**.

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

$$\begin{array}{c}
 \swarrow \quad \searrow \\
 P[z_1/y_1, z_2/y_2] \parallel Q \parallel \bar{x}\langle z'_1, z'_2 \rangle.Q' \quad P[z'_1/y_1, z'_2/y_2] \parallel \bar{x}\langle z_1, z_2 \rangle.Q \parallel Q'
 \end{array}$$

Monadic encoding:  $P[z_1/y_1, z_2/y_2] \parallel \dots \checkmark \quad P[z'_1/y_1, z'_2/y_2] \parallel \dots \checkmark$

$$\begin{array}{ccc}
 \uparrow^2 & & \uparrow^2 \\
 x(y_1).x(y_2).P \parallel \bar{x}\langle z_1 \rangle.\bar{x}\langle z_2 \rangle.Q \parallel \bar{x}\langle z'_1 \rangle.\bar{x}\langle z'_2 \rangle.Q' & & \\
 \downarrow_2 & & \downarrow_2 \\
 P[z_1/y_1, z'_1/y_2] \parallel \dots \not\checkmark & & P[z'_1/y_1, z_1/y_2] \parallel \dots \not\checkmark
 \end{array}$$

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

$$\begin{array}{l}
 x(y_1, \dots, y_n).P \mapsto x(w).w(y_1) \dots w(y_n).P \\
 \bar{x}\langle z_1, \dots, z_n \rangle.Q \mapsto \text{new } w (\bar{x}\langle w \rangle.\bar{w}\langle z_1 \rangle \dots \bar{w}\langle z_n \rangle.Q)
 \end{array}$$

where  $w \notin \text{fn}(Q) \cup \{y_1, \dots, y_n, z_1, \dots, z_n\}$

- Correctness:** see exercises

# Adding Recursive Process Calls

---

## Recursive Process Calls I

- **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 possibly other parametric processes)

- Semantic interpretation by new **congruence rule**:

$$A(y_1, \dots, y_n) \equiv P_A[y_1/x_1, \dots, y_n/x_n]$$

- 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  $A$

# Adding Recursive Process Calls

## Recursive Process Calls II

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  $a \in A$  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  $\bar{a}\langle\vec{w}\rangle.\text{nil}$ .
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 9.3

One-place buffer:

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

(on the board)