Concurrency Theory

Winter Semester 2019/20

Lecture 9: Variations of π-Calculus

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

https://moves.rwth-aachen.de/teaching/ws-19-20/ct/

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Outline of Lecture 9

Syntax of the Monadic π-Calculus

Definition (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:

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

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

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

Recap: The Monadic π-Calculus

Structural Congruence

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

Definition (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 (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 8.7, only processes in standard form need to be considered for defining the operational semantics:

Definition

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

Outline of Lecture 9

Recap: The Monadic π-Calculus

Example Reactions

The Polyadic π-Calculus

Adding Recursive Process Calls

Example Reactions

The Printer Server Revisited

Example 9.1

1. Printer server (cf. Example 8.1):

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

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

(on the board)

Example Reactions

The Printer Server Revisited

Example 9.1

1. Printer server (cf. Example 8.1):

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

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

   (on the board)

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

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

   (on the board)

Mobile Clients Revisited

Example 9.2

- System specification (cf. Example 8.2):

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

Example Reactions

Mobile Clients Revisited

Example 9.2

- System specification (cf. Example 8.2):

$$
\begin{aligned}
\mathit{System}_1 &= \text{new } L\,(\mathit{Client}_1 \parallel \mathit{Station}_1 \parallel \mathit{Idle}_2 \parallel \mathit{Control}_1)\\
\mathit{System}_2 &= \text{new } L\,(\mathit{Client}_2 \parallel \mathit{Idle}_1 \parallel \mathit{Station}_2 \parallel \mathit{Control}_2)\\
\mathit{Station}(\mathit{talk}, \mathit{switch}, \mathit{gain}, \mathit{lose}) &= \mathit{talk}.\mathit{Station}(\mathit{talk}, \mathit{switch}, \mathit{gain}, \mathit{lose}) +\\
&\quad\ \mathit{lose}(t, s).\overline{\mathit{switch}}\langle t, s\rangle.\mathit{Idle}(\mathit{gain}, \mathit{lose})\\
\mathit{Idle}(\mathit{gain}, \mathit{lose}) &= \mathit{gain}(t, s).\mathit{Station}(t, s, \mathit{gain}, \mathit{lose})\\
\mathit{Control}_1 &= \overline{\mathit{lose}_1}\langle \mathit{talk}_2, \mathit{switch}_2\rangle.\overline{\mathit{gain}_2}\langle \mathit{talk}_2, \mathit{switch}_2\rangle.\mathit{Control}_2\\
\mathit{Control}_2 &= \overline{\mathit{lose}_2}\langle \mathit{talk}_1, \mathit{switch}_1\rangle.\overline{\mathit{gain}_1}\langle \mathit{talk}_1, \mathit{switch}_1\rangle.\mathit{Control}_1\\
\mathit{Client}(\mathit{talk}, \mathit{switch}) &= \overline{\mathit{talk}}.\mathit{Client}(\mathit{talk}, \mathit{switch}) + \mathit{switch}(t, s).\mathit{Client}(t, s)\\
L &= (\mathit{talk}_i, \mathit{switch}_i, \mathit{gain}_i, \mathit{lose}_i \mid i \in \{1, 2\})
\end{aligned}
$$

- 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 $A(\vec{y}) \equiv P_A[\vec{y}/\vec{x}]$

Mobile Clients Revisited

Example 9.2

- System specification (cf. Example 8.2):

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

- 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: The Monadic π-Calculus

Example Reactions

The Polyadic π-Calculus

Adding Recursive Process Calls

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 behaviour (cf. Example 9.2):

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

  (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 behaviour (cf. Example 9.2):

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

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

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

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

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

  $\nwarrow^{2} \qquad \nearrow^{2}$

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

  $\swarrow^{2} \qquad \searrow^{2}$

  $P[z_1/y_1, z'_1/y_2] \parallel \ldots$ ↯ $\quad P[z'_1/y_1, z_1/y_2] \parallel \ldots$ ↯

Polyadic Communication II

- But consider the following counterexample.

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

  $$\swarrow \qquad \searrow$$

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

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

  $$\nwarrow^{2} \qquad \nearrow^{2}$$

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

  $$\swarrow^{2} \qquad \searrow^{2}$$

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

- **Solution:** avoid interferences by first introducing a fresh communication 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 $w \notin fn(Q) \cup \{y_1, \ldots, y_n, z_1, \ldots, z_n\}$

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

  $$\nwarrow^2 \qquad \nearrow^2$$

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

  $$\swarrow^2 \qquad \searrow^2$$

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

- **Solution:** avoid interferences by first introducing a fresh communication 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 $w \notin fn(Q) \cup \{y_1, \ldots, y_n, z_1, \ldots, z_n\}$
- **Correctness:** see exercises

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Outline of Lecture 9

Recursive Process Calls I

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

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

  where $A \in Pid$ is a process identifier and $P_A \in Prc^\pi$ a process expression containing calls of A (and possibly other parametric processes)

Recursive Process Calls I

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

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

  where $A \in Pid$ is a process identifier and $P_A \in Prc^\pi$ a process expression containing calls of A (and possibly other parametric processes)
- Semantic interpretation by new congruence rule (cf. Example 9.2):

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

Recursive Process Calls I

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

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

  where $A \in Pid$ is a process identifier and $P_A \in Prc^\pi$ a process expression containing calls of A (and possibly other parametric processes)
- Semantic interpretation by new congruence rule (cf. Example 9.2):

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

- 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

Recursive Process Calls II

The encoding

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

is defined as follows:

Software Modeling and Verification Chair
RWTH AACHEN UNIVERSITY

Recursive Process Calls II

The encoding

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

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 $\overline{a}\langle\vec{w}\rangle.\mathrm{nil}$.
3. Replace Q by $Q' := \mathrm{new}\, a\, (\hat{Q} \parallel !a(\vec{x}).\hat{P}_A)$.

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

Adding Recursive Process Calls

Recursive Process Calls II

The encoding

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

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 $\overline{a}\langle\vec{w}\rangle.\mathrm{nil}$.
3. Replace Q by $Q' := \mathrm{new}\, a\, (\hat{Q} \parallel !a(\vec{x}).\hat{P}_A)$.

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

Example 9.3

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

(encoding on the board)