# Concurrency Theory

## Lecture 11: Variations of $\pi$-Calculus & Communicating Sequential Processes

Joost-Pieter Katoen    Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH AACHEN UNIVERSITY

{katoen,noll}@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/ct13/

Winter Semester 2013/14

# Outline


1. Recap: Syntax and Semantics of the Monadic $\pi$-Calculus
2. Mobile Clients Revisited
3. The Polyadic $\pi$-Calculus
4. Adding Recursive Process Calls
5. Communicating Sequential Processes (CSP)

# Syntax of the Monadic $\pi$-Calculus

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

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

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

- The set $Prc^\pi$ of $\pi$-Calculus process expressions is defined by the following syntax:

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

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

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

# A Standard Form

**Theorem (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 $m = n = 0$: $\mathsf{nil}$; 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 10.5, 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:

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

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

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

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

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

($P[z/y]$ 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.)

# Outline


1. Recap: Syntax and Semantics of the Monadic $\pi$-Calculus
2. Mobile Clients Revisited
3. The Polyadic $\pi$-Calculus
4. Adding Recursive Process Calls
5. Communicating Sequential Processes (CSP)

# Example: Mobile Clients

## Example 11.1

- System specification (cf. Example 9.10):

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

## Example 11.1

- System specification (cf. Example 9.10):

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

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

# Example: Mobile Clients

## Example 11.1

- System specification (cf. Example 9.10):

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

- 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


1 Recap: Syntax and Semantics of the Monadic $\pi$-Calculus

2 Mobile Clients Revisited

3 The Polyadic $\pi$-Calculus

4 Adding Recursive Process Calls

5 Communicating Sequential Processes (CSP)

# 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:

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

(replacement of free names)

- **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:

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

  (replacement of free names)

- Obvious attempt for encoding:

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

# 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:

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

$$\swarrow \qquad \searrow$$

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

  Monadic encoding:

$$\begin{array}{cc} P[z_1/y_1, z_2/y_2] \parallel \ldots \quad \checkmark & P[z'_1/y_1, z'_2/y_2] \parallel \ldots \quad \checkmark \\ \uparrow 2 & \uparrow 2 \\ \multicolumn{2}{c}{x(y_1).x(y_2).P \parallel \overline{x}\langle z_1\rangle.\overline{x}\langle z_2\rangle.Q \parallel \overline{x}\langle z'_1\rangle.\overline{x}\langle z'_2\rangle.Q'} \\ \downarrow 2 & \downarrow 2 \\ P[z_1/y_1, z'_1/y_2] \parallel \ldots \quad \lightning & P[z'_1/y_1, z_1/y_2] \parallel \ldots \quad \lightning \end{array}$$

# 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:

$$\begin{array}{ccc} P[z_1/y_1, z_2/y_2] \parallel \ldots \quad \checkmark & & P[z'_1/y_1, z'_2/y_2] \parallel \ldots \quad \checkmark \\ \uparrow 2 & & \uparrow 2 \\ & x(y_1).x(y_2).P \parallel \overline{x}\langle z_1\rangle.\overline{x}\langle z_2\rangle.Q \parallel \overline{x}\langle z'_1\rangle.\overline{x}\langle z'_2\rangle.Q' & \\ \downarrow 2 & & \downarrow 2 \\ P[z_1/y_1, z'_1/y_2] \parallel \ldots \quad \lightning & & P[z'_1/y_1, z_1/y_2] \parallel \ldots \quad \lightning \end{array}$$

- **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:

$$\begin{array}{cc} P[z_1/y_1, z_2/y_2] \parallel \ldots \quad \checkmark & P[z_1'/y_1, z_2'/y_2] \parallel \ldots \quad \checkmark \\ \uparrow 2 & \uparrow 2 \\ \multicolumn{2}{c}{x(y_1).x(y_2).P \parallel \overline{x}\langle z_1\rangle.\overline{x}\langle z_2\rangle.Q \parallel \overline{x}\langle z_1'\rangle.\overline{x}\langle z_2'\rangle.Q'} \\ \downarrow 2 & \downarrow 2 \\ P[z_1/y_1, z_1'/y_2] \parallel \ldots \quad \lightning & P[z_1'/y_1, z_1/y_2] \parallel \ldots \quad \lightning \end{array}$$

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

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

  where $w \notin fn(Q)$

- **Correctness:** see exercises

# Outline


1. Recap: Syntax and Semantics of the Monadic $\pi$-Calculus
2. Mobile Clients Revisited
3. The Polyadic $\pi$-Calculus
4. Adding Recursive Process Calls
5. Communicating Sequential Processes (CSP)

# 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 $A$ is a process identifier and $P_A$ a process expression containing calls of $A$ (and 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 $A$ is a process identifier and $P_A$ a process expression containing calls of $A$ (and other parametric processes)
- Semantic interpretation by new congruence rule:

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

# 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 $A$ is a process identifier and $P_A$ a process expression containing calls of $A$ (and other parametric processes)
- Semantic interpretation by new congruence rule:

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

- 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 $P_A = \ldots A(\vec{u}) \ldots A(\vec{v}) \ldots$
- for main process $Q = \ldots A(\vec{y}) \ldots A(\vec{z}) \ldots$

is defined as follows:

# Recursive Process Calls II

The encoding

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

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

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

# Recursive Process Calls II

The encoding

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

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

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

## Example 11.2

One-place buffer:

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

(on the board)

# Outline


1. Recap: Syntax and Semantics of the Monadic $\pi$-Calculus
2. Mobile Clients Revisited
3. The Polyadic $\pi$-Calculus
4. Adding Recursive Process Calls
5. Communicating Sequential Processes (CSP)

# Communicating Sequential Processes

- Approach: Communicating Sequential Processes (CSP) by T. Hoare and R. Milner

# Communicating Sequential Processes

- Approach: Communicating Sequential Processes (CSP) by T. Hoare and R. Milner
- Models system of processors that
  - have (only) local store and
  - run a sequential program ("process")

# Communicating Sequential Processes

- Approach: Communicating Sequential Processes (CSP) by T. Hoare and R. Milner
- Models system of processors that
  - have (only) local store and
  - run a sequential program ("process")
- Communication proceeds in the following way:
  - processes communicate along channels
  - process can send/receive on a channel if another process simultaneously performs the complementary I/O operation

$\Longrightarrow$ no buffering (synchronous communication; just as in CCS)

# Communicating Sequential Processes

- Approach: Communicating Sequential Processes (CSP) by T. Hoare and R. Milner
- Models system of processors that
  - have (only) local store and
  - run a sequential program ("process")
- Communication proceeds in the following way:
  - processes communicate along channels
  - process can send/receive on a channel if another process simultaneously performs the complementary I/O operation

  $\Longrightarrow$ no buffering (synchronous communication; just as in CCS)
- Syntactic categories:

| Category | Domain | Meta variable |
|---|---|---|
| Numbers | $\mathbb{Z} = \{0, 1, -1, \ldots\}$ | $z$ |
| Truth values | $\mathbb{B} = \{\mathrm{tt}, \mathrm{ff}\}$ | $t$ |
| Variables | $Var = \{\mathrm{x}, \mathrm{y}, \ldots\}$ | $x$ |
| Arithmetic expressions | $AExp$ (next slide) | $a$ |
| Boolean expressions | $BExp$ (next slide) | $b$ |
| Commands | $Cmd$ (next slide) | $c$ |
| Guarded commands | $GCmd$ (next slide) | $gc$ |

## Definition 11.3 (Syntax of CSP)

The syntax of CSP is given by

$$
\begin{aligned}
a &::= z \mid x \mid a_1{+}a_2 \mid a_1{-}a_2 \mid a_1{*}a_2 \in AExp \\
b &::= t \mid a_1{=}a_2 \mid a_1{>}a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \in BExp \\
c &::= \mathsf{skip} \mid x := a \mid \alpha?x \mid \alpha!a \mid \\
&\qquad c_1; c_2 \mid \mathsf{if}\ gc\ \mathsf{fi} \mid \mathsf{do}\ gc\ \mathsf{od} \mid c_1 \parallel c_2 \in Cmd \\
gc &::= b \rightarrow c \mid b \wedge \alpha?x \rightarrow c \mid b \wedge \alpha!a \rightarrow c \mid gc_1 \,\Box\, gc_2 \in GCmd
\end{aligned}
$$

# Syntax of CSP

**Definition 11.3 (Syntax of CSP)**

The syntax of CSP is given by

$$\begin{aligned}
a &::= z \mid x \mid a_1{+}a_2 \mid a_1{-}a_2 \mid a_1{*}a_2 \in AExp \\
b &::= t \mid a_1{=}a_2 \mid a_1{>}a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \in BExp \\
c &::= \mathsf{skip} \mid x := a \mid \alpha?x \mid \alpha!a \mid \\
&\qquad c_1; c_2 \mid \mathsf{if}\ gc\ \mathsf{fi} \mid \mathsf{do}\ gc\ \mathsf{od} \mid c_1 \parallel c_2 \in Cmd \\
gc &::= b \to c \mid b \wedge \alpha?x \to c \mid b \wedge \alpha!a \to c \mid gc_1 \,\square\, gc_2 \in GCmd
\end{aligned}$$

- $\alpha?x/\alpha!a$ represents an in/put/output operation along channel $\alpha$
- In $c_1 \parallel c_2$, commands $c_1$ and $c_2$ must not share variables (only local store)
- Guarded command $gc_1 \,\square\, gc_2$ represents an alternative
- In $b \to c$, $b$ acts as a guard that enables the execution of $c$ only if evaluated to tt
- $b \wedge \alpha?x \to c$ and $b \wedge \alpha!a \to c$ additionally require the respective I/O operation to be enabled
- If none of its alternatives is enabled, a guarded command $gc$ fails (state fail)
- if nondeterministically picks an enabled alternative
- A do loop is iterated until its body fails

# Semantics of CSP I

- Defined as LTS over commands and memory states ("$\sigma$")
- Most important aspect: I/O operations
- E.g., $\langle \alpha?x; c, \sigma \rangle$ can only execute if a parallel command provides corresponding output

# Semantics of CSP I

- Defined as LTS over commands and memory states ("$\sigma$")
- Most important aspect: I/O operations
- E.g., $\langle \alpha?x; c, \sigma\rangle$ can only execute if a parallel command provides corresponding output

$\Longrightarrow$ Indicate communication potential by labels

$$L := \{\alpha?z \mid \alpha \in Chn, z \in \mathbb{Z}\} \cup \{\alpha!z \mid \alpha \in Chn, z \in \mathbb{Z}\}$$

- Yields following labeled transitions:

$$\langle \alpha?x; c, \sigma\rangle \xrightarrow{\alpha?z} \langle c, \sigma[x \mapsto z]\rangle \quad (\text{for all } z \in \mathbb{Z})$$
$$\langle \alpha!a; c', \sigma\rangle \xrightarrow{\alpha!z} \langle c', \sigma\rangle \quad (\text{if } \langle a, \sigma\rangle \to z)$$

# Semantics of CSP I

- Defined as LTS over commands and memory states ("$\sigma$")
- Most important aspect: I/O operations
- E.g., $\langle \alpha?x; c, \sigma\rangle$ can only execute if a parallel command provides corresponding output

$\Longrightarrow$ Indicate communication potential by labels

$$L := \{\alpha?z \mid \alpha \in Chn, z \in \mathbb{Z}\} \cup \{\alpha!z \mid \alpha \in Chn, z \in \mathbb{Z}\}$$

- Yields following labeled transitions:

$$\langle \alpha?x; c, \sigma\rangle \xrightarrow{\alpha?z} \langle c, \sigma[x \mapsto z]\rangle \quad (\text{for all } z \in \mathbb{Z})$$

$$\langle \alpha!a; c', \sigma\rangle \xrightarrow{\alpha!z} \langle c', \sigma\rangle \quad (\text{if } \langle a, \sigma\rangle \to z)$$

- Now both commands, if running in parallel, can communicate:

$$\langle (\alpha?x; c) \parallel (\alpha!a; c'), \sigma\rangle \to \langle c \parallel c', \sigma[x \mapsto z]\rangle.$$

# Semantics of CSP I

- Defined as LTS over commands and memory states ("$\sigma$")
- Most important aspect: I/O operations
- E.g., $\langle \alpha?x; c, \sigma \rangle$ can only execute if a parallel command provides corresponding output

$\Longrightarrow$ Indicate communication potential by labels

$$L := \{\alpha?z \mid \alpha \in Chn, z \in \mathbb{Z}\} \cup \{\alpha!z \mid \alpha \in Chn, z \in \mathbb{Z}\}$$

- Yields following labeled transitions:

$$\langle \alpha?x; c, \sigma \rangle \xrightarrow{\alpha?z} \langle c, \sigma[x \mapsto z] \rangle \quad (\text{for all } z \in \mathbb{Z})$$
$$\langle \alpha!a; c', \sigma \rangle \xrightarrow{\alpha!z} \langle c', \sigma \rangle \quad (\text{if } \langle a, \sigma \rangle \to z)$$

- Now both commands, if running in parallel, can communicate:

$$\langle (\alpha?x; c) \parallel (\alpha!a; c'), \sigma \rangle \to \langle c \parallel c', \sigma[x \mapsto z] \rangle.$$

- To allow communication with other processes, the following transitions should also be possible (for all $z' \in \mathbb{Z}$, $\langle a, \sigma \rangle \to z$):

$$\langle (\alpha?x; c) \parallel (\alpha!a; c'), \sigma \rangle \xrightarrow{\alpha?z'} \langle c \parallel (\alpha!a; c'), \sigma[x \mapsto z'] \rangle$$
$$\langle (\alpha?x; c) \parallel (\alpha!a; c'), \sigma \rangle \xrightarrow{\alpha!z} \langle (\alpha?x; c) \parallel c', \sigma \rangle$$

Definition of transition relation

$$\xrightarrow{\lambda} \subseteq (Cmd \times S) \times ((Cmd \cup \{\downarrow\}) \times S) \cup (GCmd \times S) \times ((Cmd \times S) \cup \{\text{fail}\})$$

(see following slides)

# Semantics of CSP II

Definition of transition relation

$$\xrightarrow{\lambda} \subseteq (Cmd \times S) \times ((Cmd \cup \{\downarrow\}) \times S) \cup (GCmd \times S) \times ((Cmd \times S) \cup \{\text{fail}\})$$

(see following slides)

- Memory states given by $S := \{\sigma \mid \sigma : \mathcal{X} \to \mathbb{Z}\}$

# Semantics of CSP II

Definition of transition relation

$$\xrightarrow{\lambda} \subseteq (Cmd \times S) \times ((Cmd \cup \{\downarrow\}) \times S) \cup (GCmd \times S) \times ((Cmd \times S) \cup \{\text{fail}\})$$

(see following slides)

- Memory states given by $S := \{\sigma \mid \sigma : \mathcal{X} \to \mathbb{Z}\}$
- Action $\lambda$ can be a label or empty: $\lambda \in L \cup \{\varepsilon\}$

# Semantics of CSP II

Definition of transition relation

$$\xrightarrow{\lambda} \subseteq (Cmd \times S) \times ((Cmd \cup \{\downarrow\}) \times S) \cup (GCmd \times S) \times ((Cmd \times S) \cup \{\text{fail}\})$$

(see following slides)

- Memory states given by $S := \{\sigma \mid \sigma : \mathcal{X} \to \mathbb{Z}\}$
- Action $\lambda$ can be a label or empty: $\lambda \in L \cup \{\varepsilon\}$
- $\downarrow$ stands for terminated command
  - avoids explicit distinction of final state $\sigma$ (represented by $\langle \downarrow, \sigma \rangle$)
  - satisfies $\downarrow; c = c; \downarrow = \downarrow \parallel c = c \parallel \downarrow = c$

# Semantics of CSP II

Definition of transition relation

$$\xrightarrow{\lambda} \subseteq (Cmd \times S) \times ((Cmd \cup \{\downarrow\}) \times S) \cup (GCmd \times S) \times ((Cmd \times S) \cup \{\text{fail}\})$$

(see following slides)

- Memory states given by $S := \{\sigma \mid \sigma : \mathcal{X} \to \mathbb{Z}\}$
- Action $\lambda$ can be a label or empty: $\lambda \in L \cup \{\varepsilon\}$
- $\downarrow$ stands for terminated command
  - avoids explicit distinction of final state $\sigma$ (represented by $\langle\downarrow, \sigma\rangle$)
  - satisfies $\downarrow; c = c; \downarrow = \downarrow \parallel c = c \parallel \downarrow = c$
- fail stands for failing guarded command (due to invalidity of guard)

# Semantics of CSP III

## Definition 11.4 (Semantics of CSP)

Rules for commands:

$$\frac{}{\langle \mathsf{skip}, \sigma \rangle \to \langle \downarrow, \sigma \rangle}$$

$$\frac{\langle a, \sigma \rangle \to z}{\langle x := a, \sigma \rangle \to \langle \downarrow, \sigma[x \mapsto z] \rangle}$$

$$\frac{}{\langle \alpha?x, \sigma \rangle \xrightarrow{\alpha?z} \langle \downarrow, \sigma[x \mapsto z] \rangle}$$

$$\frac{\langle a, \sigma \rangle \to z}{\langle \alpha!a, \sigma \rangle \xrightarrow{\alpha!z} \langle \downarrow, \sigma \rangle}$$

$$\frac{\langle c_1, \sigma \rangle \xrightarrow{\lambda} \langle c_1', \sigma' \rangle}{\langle c_1; c_2, \sigma \rangle \xrightarrow{\lambda} \langle c_1'; c_2, \sigma' \rangle}$$

$$\frac{\langle gc, \sigma \rangle \xrightarrow{\lambda} \langle c, \sigma' \rangle}{\langle \mathsf{if}\ gc\ \mathsf{fi}, \sigma \rangle \xrightarrow{\lambda} \langle c, \sigma' \rangle}$$

$$\frac{\langle gc, \sigma \rangle \xrightarrow{\lambda} \langle c, \sigma' \rangle}{\langle \mathsf{do}\ gc\ \mathsf{od}, \sigma \rangle \xrightarrow{\lambda} \langle c; \mathsf{do}\ gc\ \mathsf{od}, \sigma' \rangle}$$

$$\frac{\langle gc, \sigma \rangle \to \mathsf{fail}}{\langle \mathsf{do}\ gc\ \mathsf{od}, \sigma \rangle \to \langle \downarrow, \sigma \rangle}$$

$$\frac{\langle c_1, \sigma \rangle \xrightarrow{\lambda} \langle c_1', \sigma' \rangle}{\langle c_1 \parallel c_2, \sigma \rangle \xrightarrow{\lambda} \langle c_1' \parallel c_2, \sigma' \rangle}$$

$$\frac{\langle c_2, \sigma \rangle \xrightarrow{\lambda} \langle c_2', \sigma' \rangle}{\langle c_1 \parallel c_2, \sigma \rangle \xrightarrow{\lambda} \langle c_1 \parallel c_2', \sigma' \rangle}$$

$$\frac{\langle c_1, \sigma \rangle \xrightarrow{\alpha?z} \langle c_1', \sigma' \rangle \quad \langle c_2, \sigma \rangle \xrightarrow{\alpha!z} \langle c_2', \sigma \rangle}{\langle c_1 \parallel c_2, \sigma \rangle \to \langle c_1' \parallel c_2', \sigma' \rangle}$$

$$\frac{\langle c_1, \sigma \rangle \xrightarrow{\alpha!z} \langle c_1', \sigma \rangle \quad \langle c_2, \sigma \rangle \xrightarrow{\alpha?z} \langle c_2', \sigma' \rangle}{\langle c_1 \parallel c_2, \sigma \rangle \to \langle c_1' \parallel c_2', \sigma' \rangle}$$

## Definition 11.4 (Semantics of CSP; continued)

Rules for guarded commands:

$$\frac{\langle b, \sigma \rangle \to \mathrm{tt}}{\langle b \to c, \sigma \rangle \to \langle c, \sigma \rangle} \qquad \frac{\langle b, \sigma \rangle \to \mathrm{ff}}{\langle b \to c, \sigma \rangle \to \mathrm{fail}}$$

$$\frac{\langle b, \sigma \rangle \to \mathrm{tt}}{\langle b \wedge \alpha?x \to c, \sigma \rangle \xrightarrow{\alpha?z} \langle c, \sigma[x \mapsto z] \rangle} \qquad \frac{\langle b, \sigma \rangle \to \mathrm{ff}}{\langle b \wedge \alpha?x \to c, \sigma \rangle \to \mathrm{fail}}$$

$$\frac{\langle b, \sigma \rangle \to \mathrm{tt} \quad \langle a, \sigma \rangle \to z}{\langle b \wedge \alpha!a \to c, \sigma \rangle \xrightarrow{\alpha!z} \langle c, \sigma \rangle} \qquad \frac{\langle b, \sigma \rangle \to \mathrm{ff}}{\langle b \wedge \alpha!a \to c, \sigma \rangle \to \mathrm{fail}}$$

$$\frac{\langle gc_1, \sigma \rangle \xrightarrow{\lambda} \langle c, \sigma' \rangle}{\langle gc_1 \,\square\, gc_2, \sigma \rangle \xrightarrow{\lambda} \langle c, \sigma' \rangle} \qquad \frac{\langle gc_2, \sigma \rangle \xrightarrow{\lambda} \langle c, \sigma' \rangle}{\langle gc_1 \,\square\, gc_2, \sigma \rangle \xrightarrow{\lambda} \langle c, \sigma' \rangle}$$

$$\frac{\langle gc_1, \sigma \rangle \to \mathrm{fail} \quad \langle gc_2, \sigma \rangle \to \mathrm{fail}}{\langle gc_1 \,\square\, gc_2, \sigma \rangle \to \mathrm{fail}}$$

# CSP Examples

## Example 11.5

(on the board)

1. $\text{do } (\text{tt} \wedge \alpha?\text{x} \rightarrow \beta!\text{x}) \text{ od}$
   describes a process that repeatedly receives a value along $\alpha$ and forwards it along $\beta$ (i.e., a one-place buffer)

# CSP Examples

## Example 11.5

(on the board)

1. $\text{do }(\text{tt} \wedge \alpha?x \to \beta!x)\text{ od}$
   describes a process that repeatedly receives a value along $\alpha$ and forwards it along $\beta$ (i.e., a one-place buffer)
2. $\text{do tt} \wedge \alpha?x \to \beta!x\text{ od} \parallel \text{do tt} \wedge \beta?y \to \gamma!y\text{ od}$
   specifies a two-place buffer that receives along $\alpha$ and sends along $\gamma$ (using $\beta$ for internal communication)

# CSP Examples

## Example 11.5

(on the board)

1. $\text{do } (\text{tt} \wedge \alpha?x \to \beta!x) \text{ od}$
   describes a process that repeatedly receives a value along $\alpha$ and forwards it along $\beta$ (i.e., a one-place buffer)
2. $\text{do tt} \wedge \alpha?x \to \beta!x \text{ od} \parallel \text{do tt} \wedge \beta?y \to \gamma!y \text{ od}$
   specifies a two-place buffer that receives along $\alpha$ and sends along $\gamma$ (using $\beta$ for internal communication)
3. Nondeterministic choice between input channels:
   1. $\text{if } (\text{tt} \wedge \alpha?x \to c_1 \,\square\, \text{tt} \wedge \beta?y \to c_2) \text{ fi}$
   2. $\text{if } (\text{tt} \to (\alpha?x; c_1) \,\square\, \text{tt} \to (\beta?y; c_2)) \text{ fi}$

   Expected: progress whenever environment provides data on $\alpha$ or $\beta$
   1. correct
   2. incorrect (can deadlock)