The π-calculus with Stop

David Sabel

Goethe-Universität, Frankfurt am Main

January 21, 2014

Overview

Introduction

- the π-**calculus** is a core language for **concurrent processes**
- is a **message passing** model
- the **control flow** of π-programs is expressed by **process communication**
- introduced by **R. Milner, J. Parrow & D. Walker, 1992**
- extends CCS (Calculus of Communicating Systems, R. Milner, 1980) by **mobility of processes**

Some Applications

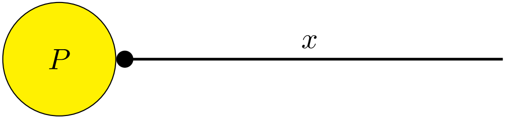
- The **Spi-calculus** and the **applied π-calculus** to **verify cryptographic protocols**
  (Abadi & Gordon 1997, Abadi & Fournet 2001)
- π-calculus as a theoretical basis of **business processes**
  (Smith & Fingar, 2002)
- representation of **biochemical processes** using the **stochastic π-calculus** (Priami, Regev, Silverman & Shapiro, 2001)
- the **join calculus** is a core model for the **distributed programming language** JoCaml (Laneve 1996, Fournet & Gonthier 2000)

Parallel Composition

$$P \mid Q$$

"processes P and Q run concurrently"

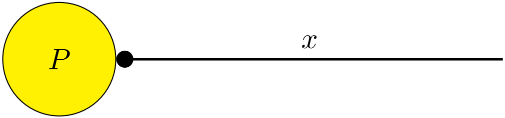
Links



$$x.P \quad \text{or} \quad \overline{x}.P$$

"P is linked to channel named x"

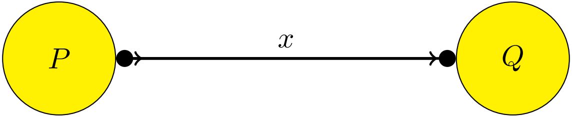
Links



$$\underbrace{x.P}_{\text{receive}} \quad \text{or} \quad \underbrace{\overline{x}.P}_{\text{send}}$$

"P is linked to channel named x"

Communication



$$\overline{x}.P \mid x.Q$$

"P (sender) and Q (receiver) can communicate"

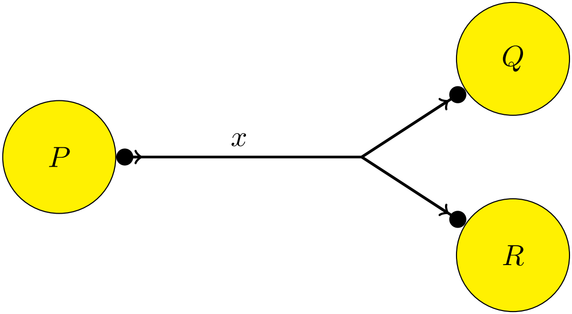
Communication

$$\overline{x}.P \mid x.Q \quad \rightarrow \quad P \mid Q$$

"P (sender) and Q (receiver) can communicate"

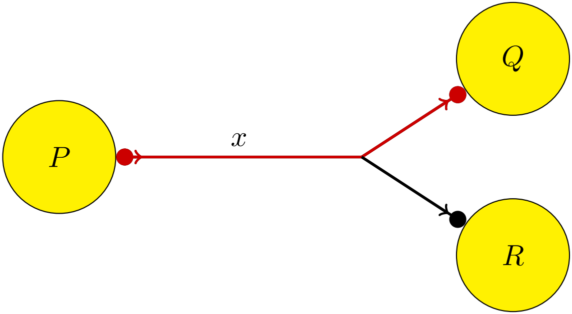
"P sent a message to Q"

Nondeterminism



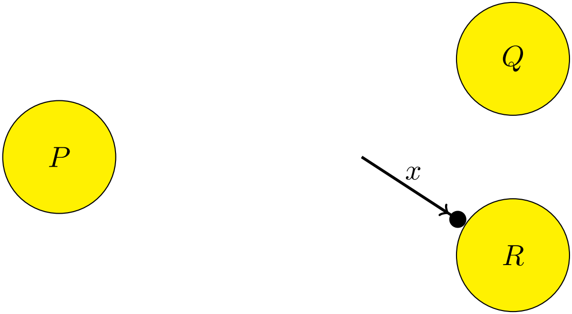
$$\overline{x}.P \mid x.Q \mid x.R$$

Nondeterminism



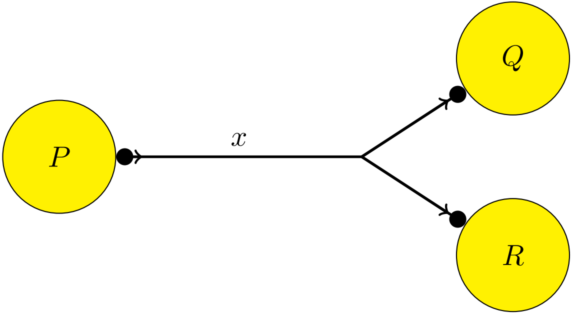
$$\overline{x}.P \mid x.Q \mid x.R$$

Nondeterminism



$$\overline{x}.P \mid x.Q \mid x.R \rightarrow P \mid Q \mid x.R$$

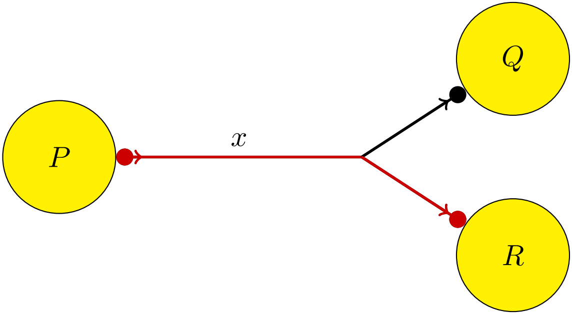
Nondeterminism



$$P \mid Q \mid x.R$$

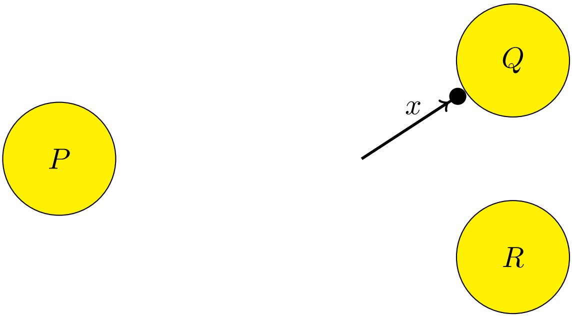
$$\overline{x}.P \mid x.Q \mid x.R \nearrow$$

Nondeterminism



$$\color{red}{\overline{x}.P} \mid x.Q \mid \color{red}{x.R} \longrightarrow P \mid Q \mid x.R$$

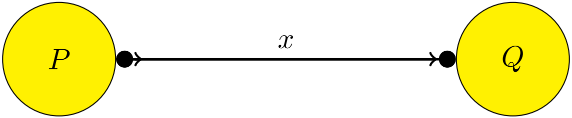
Nondeterminism



$$\overline{x}.P \mid x.Q \mid x.R \rightarrow P \mid Q \mid x.R$$

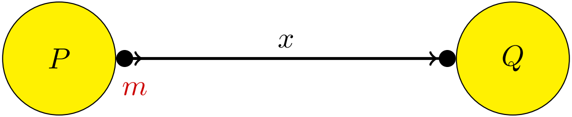
$$\overline{x}.P \mid x.Q \mid x.R \rightarrow P \mid x.Q \mid R$$

Messages



$$\overline{x}.P \mid x.Q \rightarrow P \mid Q$$

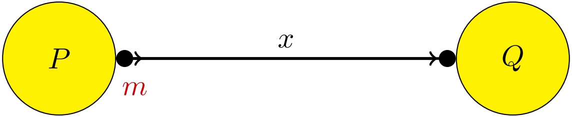
Messages



$$\overline{x}\langle m\rangle.P \mid x.Q \quad \rightarrow \quad P \mid Q$$

"P sends message m along x"

Messages



$$\overline{x}\langle m\rangle.P \mid x(y).Q \;\rightarrow\; P \mid Q$$

"P sends message m along x"

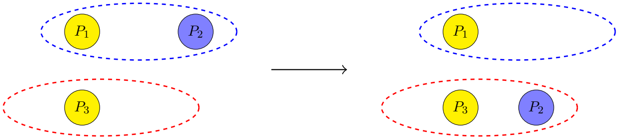
Messages

$$\overline{x}\langle m\rangle.P \mid x(y).Q \;\to\; P \mid Q[m/y]$$

"P sends message m along x"

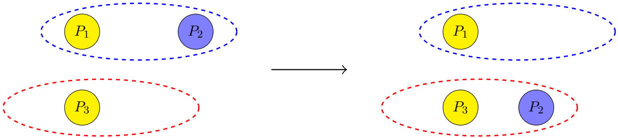
Approaches to Mobility

1. Processes move their location in the physical space of processes

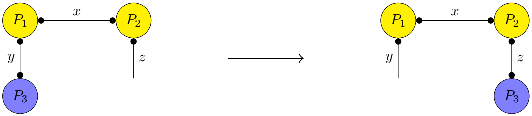


Approaches to Mobility

1. Processes move their location in the physical space of processes

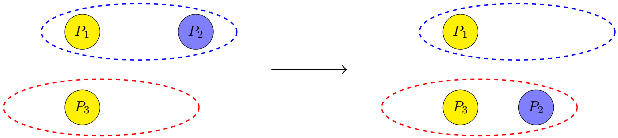


2. Processes move their location in the virtual space of linked processes

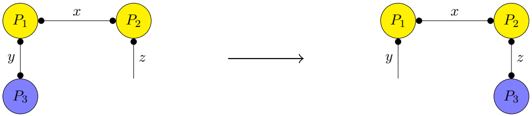


Approaches to Mobility

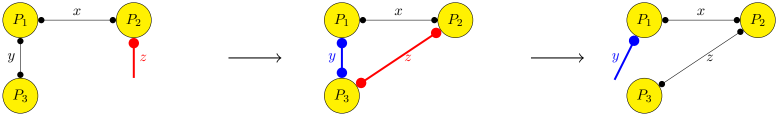
1. Processes move their location in the physical space of processes



2. Processes move their location in the virtual space of linked processes



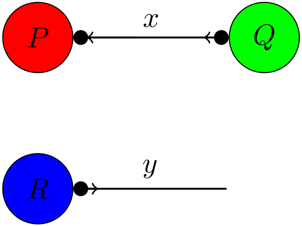
3. Links move in the virtual space of linked processes
(approach of the π-calculus, includes the second approach)



Mobility (2)

How to move links?

$\Rightarrow$ Send them as messages!

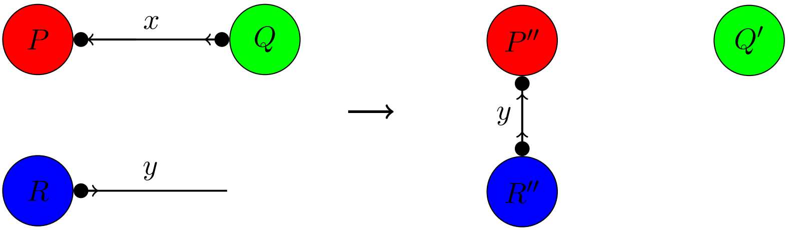


$$\underbrace{x(z).z(w).P'}_{P} \mid \underbrace{\overline{x}\langle y\rangle.Q'}_{Q} \mid \underbrace{\overline{y}\langle u'\rangle.R'}_{R}$$

Mobility (2)

How to move links?

$\Rightarrow$ Send them as messages!



$$\underbrace{x(z).z(w).P'}_{P} \mid \underbrace{\overline{x}\langle y\rangle.Q'}_{Q} \mid \underbrace{\overline{y}\langle u'\rangle.R'}_{R} \longrightarrow (z(w).P')[z/y] \mid Q' \mid \overline{y}\langle u'\rangle.R'$$

$$= \underbrace{y(w).P'}_{P''} \mid Q' \mid \underbrace{\overline{y}\langle u'\rangle.R'}_{R''}$$

Private Communication

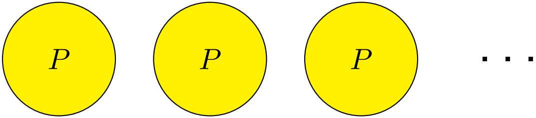
$$\nu x.P$$

"channel x is private for P"

Example: $\nu x.(x(y).P \mid \overline{x}\langle z\rangle.Q) \mid \overline{x}\langle z'\rangle.R$

- no communication between R and P allowed
- equivalent to $\nu x'.(x'(y).P \mid \overline{x'}\langle z\rangle.Q) \mid \overline{x}\langle z'\rangle.R$

Replication



$$!\,P$$

"$!\,P$ means $\underbrace{P \mid P \mid P \mid \ldots}_{\text{infinitely many parallel copies of } P}$"

Syntax of a minimalistic (synchronous) π-calculus

Syntax of **calculus** Π where $x, y \in \mathcal{N}$ is a countably-infinite set of names

P	$::=$	$\pi.P$	(action)
	$\mid$	$P_1 \mid P_2$	(parallel composition)
	$\mid$	$!\,P$	(replication)
	$\mid$	$\mathbf{0}$	(silent process)
	$\mid$	$\nu x.P$	(name restriction)
π	$::=$	$x(y)$	input
	$\mid$	$\overline{x}\langle y\rangle$	output

Binding scopes:

- in $\nu x.P$ name x is bound with scope P
- in $x(y).P$ name y is bound with scope P

Contexts $C \in \mathcal{C}$: Process with a hole $[\cdot]$ at process position

Structural Congruence

Structural Congruence $\equiv$

Smallest congruence on processes satisfying the following axioms

$$
\begin{array}{rcll}
P & \equiv & Q, & \text{if } P \text{ and } Q \text{ are } \alpha\text{-equivalent} \\
P_1 \mid (P_2 \mid P_3) & \equiv & (P_1 \mid P_2) \mid P_3 & \\
P_1 \mid P_2 & \equiv & P_2 \mid P_1 & \\
P \mid \mathbf{0} & \equiv & P & \\
\nu z.\nu w.P & \equiv & \nu w.\nu z.P & \\
\nu z.\mathbf{0} & \equiv & \mathbf{0} & \\
\nu z.(P_1 \mid P_2) & \equiv & P_1 \mid \nu z.P_2, & \text{if } z \notin \mathsf{fn}(P_1) \\
!\,P & \equiv & P \mid !\,P &
\end{array}
$$

Remark:

- Decidability of $P \equiv Q$ is unknown
- Schmidt-Schauß, Rau, S. 2013: EXPSPACE-hardness

Operational Semantics: Small-Step Reduction

Reduction rule for interaction:

$$x(y).P \mid \overline{x}\langle v\rangle.Q \xrightarrow{ia} P[v/y] \mid Q.$$

Reduction contexts $\mathcal{D}$:

$$D \in \mathcal{D} ::= [\cdot] \mid D \mid P \mid P \mid D \mid \nu x.D$$

Standard reduction $\xrightarrow{sr}$:

$$\frac{P \equiv D[P'], \quad P' \xrightarrow{ia} Q', D[Q'] \equiv Q, \text{ and } D \in \mathcal{D}}{P \xrightarrow{sr} Q}$$

Notation:

- $\xrightarrow{sr,*} := \bigcup_{i\geq 0} \xrightarrow{sr,i}$ and $\xrightarrow{sr,+} := \bigcup_{i>0} \xrightarrow{sr,i}$
- $P \xrightarrow{sr,0} P$ and $P \xrightarrow{sr,i} Q$ iff $\exists P'$: $P \xrightarrow{sr} P'$ and $P' \xrightarrow{sr,i-1} Q$.

Examples

Encoding of internal choice $\oplus$

$$P \oplus Q := \nu x, y.(\overline{x}\langle y\rangle.P \mid \overline{x}\langle y\rangle.Q \mid x(z).\mathbf{0})$$

$$(x, y \notin (\mathsf{fn}(P) \cup \mathsf{fn}(Q)))$$

$$\begin{array}{lcl} P \oplus Q & \equiv & \nu x, y.([x(z).\mathbf{0} \mid \overline{x}\langle y\rangle.P] \mid \overline{x}\langle y\rangle.Q) \\ & \xrightarrow{sr} & \nu x, y.([P \mid \mathbf{0}] \mid \overline{x}\langle y\rangle.Q) \\ & \equiv & P \mid \underbrace{\nu x, y.(\overline{x}\langle y\rangle.Q)}_{\text{“garbage”}} \end{array}$$

Other reduction possibility:

$$P \oplus Q \xrightarrow{sr} Q \mid \underbrace{\nu x, y.(\overline{x}\langle y\rangle.P)}_{\text{“garbage”}}$$

Process Equivalence

- equate processes if their "behavior" is indistinguishable
- should be a **congruence**
- a lot of approaches for process equivalence

Observed behavior: input and output capabilities

- $\nu\mathcal{X}.(x(y).P_1 \mid P_2)$ with $x \notin \mathcal{X}$ has an **input** capability.
- $\nu\mathcal{X}.(\overline{x}\langle y\rangle.P_1 \mid P_2)$ with $x \notin \mathcal{X}$ has an **output** capability.

  two cases:

  - $y \notin \mathcal{X}$: The emitted name is free
  - $y \in \mathcal{X}$: The emitted name is bound

A Hierarchy of Process Equivalences

full strong labelled bisimilarity

$\cap$

full (weak) labelled bisimilarity

$|\!\cap$

barbed congruence

$\cap$

barbed should-testing

$\cap$

barbed may-testing

fine

coarse

Barbs

Barb

- $P \upharpoonright^{x}$ iff $P \equiv \nu\mathcal{X}.(x(y).P' \mid P'')$ where $x \notin \mathcal{X}$,
- $P \upharpoonright^{\overline{x}}$ iff $P \equiv \nu\mathcal{X}.(\overline{x}\langle y\rangle.P' \mid P'')$ where $x \notin \mathcal{X}$.

May-barb and Should-barb

For $\mu \in \{x, \overline{x}\}$:

- $P \downarrow_{\mu}$ (P may have a barb on x) iff $\exists Q : P \xrightarrow{sr,*} Q \wedge Q \upharpoonright^{\mu}$, and
- $P \Downarrow_{\mu}$ (P should have a barb on x) iff $\forall Q : P \xrightarrow{sr,*} Q \implies Q \downarrow_{\mu}$.

Notations:

- $P \upharpoonright_{\mu}$ iff $\neg(P \Downarrow_{\mu})$
- $P \Uparrow_{\mu}$ iff $\neg(P \downarrow_{\mu})$

Barbed Testing

For $\mu \in \{x, \overline{x}\}$, and $\xi \in \{\downharpoonleft_\mu, \Downarrow_\mu, \upharpoonleft_\mu, \Uparrow_\mu\}$

- barbed may-testing preorder:

$$P \sqsubseteq_{c,\text{may}} Q \text{ iff } \forall x \in \mathcal{N}, \mu \in \{x, \overline{x}\}, C \in \mathcal{C} : C[P] \downharpoonleft_\mu \implies C[Q] \downharpoonleft_\mu$$

- barbed should-testing preorder:

$$P \sqsubseteq_{c,\text{should}} Q \text{ iff } \forall x \in \mathcal{N}, \mu \in \{x, \overline{x}\}, C \in \mathcal{C} : C[P] \Downarrow_\mu \implies C[Q] \Downarrow_\mu$$

- barbed testing preorder: $\sqsubseteq_c := \sqsubseteq_{c,\text{may}} \cap \sqsubseteq_{c,\text{should}}$
- barbed testing equivalences

$$\begin{aligned} \sqsupseteq\!\!\!\sqsubseteq_{c,\text{may}} &:= \sqsubseteq_{c,\text{may}} \cap (\sqsubseteq_{c,\text{may}})^{-1} \\ \sqsupseteq\!\!\!\sqsubseteq_{c,\text{should}} &:= \sqsubseteq_{c,\text{should}} \cap (\sqsubseteq_{c,\text{should}})^{-1} \\ \sqsupseteq\!\!\!\sqsubseteq_{c} &:= (\sqsubseteq_{c}) \cap (\sqsubseteq_{c})^{-1} \end{aligned}$$

Examples

Barbed testing equivalence is coarse:

$$(\underbrace{a(z).\mathbf{0}}_{A} \oplus \underbrace{b(z).\mathbf{0}}_{B}) \oplus \underbrace{c(z).\mathbf{0}}_{C} \quad \sqsubseteq_c \quad \underbrace{a(z).\mathbf{0}}_{A} \oplus (\underbrace{b(z).\mathbf{0}}_{B} \oplus \underbrace{c(z).\mathbf{0}}_{C})$$

Barbed may-testing is too coarse:

$$a(z).\mathbf{0} \sqsubseteq_{c,\text{may}} a(z).\mathbf{0} \oplus \mathbf{0}$$

Barbed should-testing is finer:

$$a(z).\mathbf{0} \not\sqsubseteq_{c,\text{should}} a(z).\mathbf{0} \oplus \mathbf{0}$$

Alternative Definitions of Barbed Testing

Theorem (Should-Testing includes May-Testing)

$\sqsubseteq_{c,\text{should}} \subset (\sqsubseteq_{c,\text{may}})^{-1}$ and thus $\sqsupseteq_c = \sqsupseteq_{c,\text{should}}$.

Theorem

$\sqsubseteq_{c,\text{should}} = \sqsubseteq_{c,\Downarrow_x} = \sqsubseteq_{c,\Downarrow}$ where

$$\begin{array}{lcl} P \sqsubseteq_{c,\text{should}} Q & := & \forall x \in \mathcal{N}, \mu \in \{x, \overline{x}\}, C \in \mathcal{C} : C[P] \Downarrow_\mu \implies C[Q] \Downarrow_\mu \\ P \sqsubseteq_{c,\Downarrow_x} Q & := & \forall C \in \mathcal{C} : C[P] \Downarrow_x \implies C[Q] \Downarrow_x \\ P \sqsubseteq_{c,\Downarrow} Q & := & \forall C \in \mathcal{C} : (\exists x : C[P] \Downarrow_x) \implies (\exists x : C[Q] \Downarrow_x) \end{array}$$

(analogous for barbed may-testing)

Proofs:

- (Fournet & Gonthier 2005) for the asynchronous π-calculus
- for Π also included in (S. & Schmidt-Schauß 2014, submitted)

Contextual Equivalence as Program Equivalence

Contextual Equivalence is a **general** notion of program equivalence for a lot of (and quite different) program calculi.

Required ingredients:

- expressions e
- contexts C (expressions with a hole)
- reduction relation $\rightarrow$
- predicate for successful termination

For any such calculus one can define

- May-convergence: $e\downarrow$ iff $\exists e : e \xrightarrow{*} e' \wedge e'$ is successful
- Should-convergence: $e\Downarrow$ iff $\forall e' : e \xrightarrow{*} e' \implies e'\downarrow$
- for $\xi \in \{\downarrow, \Downarrow\} : e_1 \leq_{c,\xi} e_2$ iff $\forall C : C[e_1]\xi \implies C[e_2]\xi$
- Contextual preorder: $\leq_c \;:=\; \leq_{c,\downarrow} \cap \leq_{c,\Downarrow}$
- Contextual equivalence: $\sim_c \;:=\; \leq_c \;\cap\; (\leq_c)^{-1}$

Advantages of Contextual Equivalence

- contextual equivalence is a **congruence** by definition
- contextual equivalence is usually the **coarsest** meaningful program equivalence
- having such a **common notion** of program equivalence makes program calculi (easier) **comparable**.

  For two calculi $calc_1, calc_2$:
  Does a translation $\psi : calc_1 \to calc_2$ exist, s.t.

  - ψ is adequate: $\psi(e_1) \sim_{c,calc_2} \psi(e_2) \implies e_1 \sim_{c,calc_1} e_2$
  - ψ is full-abstract: $\psi(e_1) \sim_{c,calc_2} \psi(e_2) \iff e_1 \sim_{c,calc_1} e_2$

  (see e.g. (Schmidt-Schauß, Niehren, Schwinghammer & S. 2008))

Back to the π-calculus

(strong) bisimilarity, barbed testing:

- instead of observing **success**, the **input/output capabilities** are observed
- other calculi **do not have channel names**, which makes them hard to compare to Π
- barbed testing is close to contextual equivalence, but:
  - $P{\upharpoonright^x}$ and $P \xrightarrow{sr} P'$ with $\neg(P'{\upharpoonright^x})$ is possible:

    $$(x(y).\mathbf{0} \mid \overline{x}\langle y\rangle.\mathbf{0}) \xrightarrow{sr} \mathbf{0}$$

  - hence ${\upharpoonright^x}$ **is not a notion of successful termination**.

The π-calculus with Stop

Our approach (S. & Schmidt-Schauß 2014, submitted):

- add a **syntactic constant** Stop to indicate success.
- contextual equivalence based on the new notion of success
- Stop can be seen as a **new programming primitive**:

  a process can **shutdown** the whole program

The π-calculus with Stop: Π_{Stop}

$$
\begin{array}{lcll}
P & ::= & \pi.P & \text{(action)} \\
 & | & P_1 \mid P_2 & \text{(parallel composition)} \\
 & | & !\,P & \text{(replication)} \\
 & | & \mathbf{0} & \text{(silent process)} \\
 & | & \nu x.P & \text{(name restriction)} \\
 & | & \mathsf{Stop} & \text{(success)} \\
 & & & \\
\pi & ::= & x(y) \mid \overline{x}\langle y\rangle &
\end{array}
$$

Further adaptions:

- contexts may also include Stop
- structural congruence: $\nu x.\mathsf{Stop} \equiv \mathsf{Stop}$
- standard reduction $\xrightarrow{sr}$ unchanged

Successful process: A process P is **successful** iff $P \equiv \mathsf{Stop} \mid P'$

Lemma: P successful and $P \xrightarrow{sr} P' \implies P'$ successful.

Contextual Equivalence in Π_{Stop}

- May-convergence: $P\downarrow$ iff $\exists P : P \xrightarrow{sr,*} P' \wedge P'$ is successful
- Should-convergence: $P\Downarrow$ iff $\forall P' : P \xrightarrow{sr,*} P' \implies P'\downarrow$

Notation:

- Must-Divergence: $P\Uparrow$ iff $\neg(P\downarrow)$
- May-Divergence: $P\uparrow$ iff $\neg P\Downarrow$

Contextual Preorder & Equivalence

- for $\xi \in \{\downarrow, \Downarrow\} : P_1 \leq_{c,\xi} P_2$ iff $\forall C : C[P_1]\xi \implies C[P_2]\xi$
- Contextual preorder: $\leq_c \;:=\; \leq_{c,\downarrow} \cap \leq_{c,\Downarrow}$
- Contextual equivalence: $\sim_c \;:=\; \leq_c \;\cap\; (\leq_c)^{-1}$

Conservativity

Theorem

Let P, Q be Stop-free processes. Then $P \sqsubseteq_c Q$ iff $P \sim_c Q$.

Consequences:

- Contextual equivalence in Π_{Stop} is compatible with existing process equivalences in Π.
- For Stop-free processes: $\approx^{\sigma}_{b,strong} \subset \approx^{\sigma}_{b} \subset \sqsubseteq_c = \sim_c$

Proof Tools: Context Lemma

Contexts $[\cdot] \mid R$ and name substitutions are sufficient to prove or disprove contextual equivalences, i.e.:

Context Lemma

For all processes P, Q:

- $P \leq_{c,\downarrow} Q$ iff $\forall \sigma, R$: $\sigma(P) \mid R\downarrow \implies \sigma(Q) \mid R\downarrow$
- $P \leq_c Q$ iff $\forall \sigma, R$:
  $(\sigma(P) \mid R\downarrow \implies \sigma(Q) \mid R\downarrow) \wedge (\sigma(P) \mid R\Downarrow \implies \sigma(Q) \mid R\Downarrow)$

Proof Tools: Action-Semantics

Labelled transitions: $P \xrightarrow{\alpha} Q$ with $\alpha \in Act = \{\overline{x}\langle y\rangle, x(y), \nu y.x(y)\}$:

Definition

- Open input: If $P \equiv \nu\mathcal{X}.(x(y).P_1 \mid P_2)$ (with $x \notin \mathcal{X}$) then $P \xrightarrow{\overline{x}\langle z\rangle} \nu\mathcal{X}.(P_1[z/y] \mid P_2)$ (for all $z \in \mathcal{N}$)
- Open output: If $P \equiv \nu\mathcal{X}.(\overline{x}\langle y\rangle.P_1 \mid P_2)$ with $x, y \notin \mathcal{X}$, then $P \xrightarrow{x(y)} \nu\mathcal{X}.(P_1 \mid P_2)$
- Bound output: If $P \equiv \nu\mathcal{X}, \nu y.(\overline{x}\langle y\rangle.P_1 \mid P_2)$ with $x \notin \mathcal{X}$, then $P \xrightarrow{\nu y.x(y)} \nu\mathcal{X}.(P_1 \mid P_2)$.

Proof Tools: Similarity

May-similarity

A binary relation η is an applicative $\downarrow$-simulation iff for all $(P, Q) \in \eta$:

- If P is successful, then $Q{\downarrow}$.
- If $P \xrightarrow{sr} P'$, then $\exists Q'$ with $Q \xrightarrow{sr,*} Q'$ and $(P', Q') \in \eta$.
- If P is not successful, for $\alpha \in Act$: $P \xrightarrow{\alpha} P'$, then $\exists Q'$ with $Q \xrightarrow{sr,*}\xrightarrow{\alpha} Q'$ and $(P', Q') \in \eta$.

Applicative $\downarrow$-similarity $\precsim_{b,\downarrow}$ is the largest applicative simulation.
Full applicative $\downarrow$-similarity $\precsim_{b,\downarrow}^{\sigma}$: $P \precsim_{b,\downarrow}^{\sigma} Q$ iff $\forall \sigma : \sigma(P) \precsim_{b,\downarrow} \sigma(Q)$

Theorem (Soundness)

$$\precsim_{b,\downarrow}^{\sigma} \quad \subset \quad \leq_{c,\downarrow}$$

Proof Tools: Similarity (2)

May-Divergence Similarity

A binary relation η is an applicative $\uparrow$-simulation iff for all $(P,Q) \in \eta$

- If $P\Uparrow$, then $Q\uparrow$.
- If $P \xrightarrow{sr} P'$, then $\exists Q'$ with $Q \xrightarrow{sr,*} Q'$ and $(P',Q') \in \eta$.
- If P is not must-divergent, then $\forall \alpha \in Act$: If $P \xrightarrow{\alpha} P'$ then $\exists Q'$ with $Q \xrightarrow{sr,*}\xrightarrow{\alpha} Q'$ and $(P',Q') \in \eta$.
- $Q \precsim_{b,\downarrow} P$

Applicative $\uparrow$-similarity $\precsim_{b,\uparrow}$ is the largest applicative $\uparrow$-simulation. Full applicative $\uparrow$-similarity: $P \precsim_{b,\uparrow}^{\sigma} Q$ iff $\forall \sigma : \sigma(P) \precsim_{b,\uparrow} \sigma(Q)$

Proof Tools: Similarity (3)

Theorem (Soundness)

- $(P \precsim^{\sigma}_{b,\downarrow} Q \land Q \precsim^{\sigma}_{b,\uparrow} P) \quad \Longrightarrow \quad P \leq_c Q$
- $P \precsim^{\sigma}_{b,\uparrow} Q \quad \Longrightarrow \quad Q \precsim^{\sigma}_{b,\downarrow} P$
- $(P \precsim^{\sigma}_{b,\uparrow} Q \ \land \ Q \precsim^{\sigma}_{b,\uparrow} P) \quad \Longrightarrow \quad P \sim_c Q$

- Note that $\precsim^{\sigma}_{b,\uparrow}$ is fine-grained, e.g.
  for $A := a(x).\mathbf{0}$, $B := b(x).\mathbf{0}$, $C := b(x).\mathbf{0}$:

  $$(A \oplus B) \oplus C \ \not\precsim^{\sigma}_{b,\uparrow} \ A \oplus (B \oplus C)$$

  although

  $$(A \oplus B) \oplus C \ \sim_c \ A \oplus (B \oplus C)$$

- Open problem: find a coarser sound similarity for $\uparrow$

Tools at Work: A Correct Program Transformation

Correctness of Deterministic Interaction

For all processes P, Q the following equation holds:

$$\nu x.(x(y).P \mid \overline{x}\langle z\rangle.Q)) \sim_c \nu x.(P[z/y] \mid Q)$$

Proof: Let

$$\mathcal{S} = \{(\sigma(\nu x.(x(y).P \mid \overline{x}\langle z\rangle.Q)), \sigma(\nu x.(P[z/y] \mid Q))) \mid \text{for all } x, y, z, P, Q, \sigma\} \cup \equiv$$

$\mathcal{S}$ and $\mathcal{S}^{-1}$ are applicative $\uparrow$-simulations.

Tools at Work: Some more laws

Theorem

For all processes P, Q the following equivalences hold:

1. $!\,P \sim_c \,!\,!\,P$.
2. $!\,P \mid !\,P \sim_c \,!\,P$.
3. $!\,(P \mid Q) \sim_c \,!\,P \mid !\,Q$.
4. $!\,\mathbf{0} \sim_c \mathbf{0}$.
5. $!\,\mathsf{Stop} \sim_c \mathsf{Stop}$.
6. $!\,(P \mid Q) \sim_c \,!\,(P \mid Q) \mid P$.
7. $x(y).\nu z.P \sim_c \nu z.x(y).P$ if $z \notin \{x, y\}$.
8. $\overline{x}\langle y\rangle.\nu z.P \sim_c \nu z.\overline{x}\langle y\rangle.P$ if $z \notin \{x, y\}$.

Proof: $\mathcal{S}_i \cup \precsim_{b,\uparrow}$ and $\mathcal{S}_i^{-1} \cup \precsim_{b,\uparrow}$ are applicative $\uparrow$-simulations where $\mathcal{S}_i := \{(R \mid l_i, R \mid r_i) \mid \text{for all } R, P, Q\}$, and l_i, r_i are the left and right hand side of the i^{th} equation.

Analyzing the Contextual Ordering

Theorem

1. If P, Q are two successful processes, then $P \sim_c Q$.
2. If P, Q are two processes with $P{\downarrow}, Q{\downarrow}$, then $P \sim_{c,\downarrow} Q$.
3. There are may-convergent processes P, Q with $P \not\sim_c Q$.
4. Stop is the greatest process w.r.t. $\leq_c$.
5. $\mathbf{0}$ is the smallest process w.r.t. $\leq_{c,\downarrow}$.
6. There is no smallest process w.r.t. $\leq_c$.

More Results

"Observing should-convergence is sufficient:"

Theorem

$\leq_{c,\Downarrow} \;=\; \leq_c \;\neq\; \leq_{c,\downarrow}$.

"Stop is not expressible in Π":

Theorem

There is no surjective translation $\psi : \Pi_{\mathsf{Stop}} \to \Pi$ s.t. for all P, Q: $P \leq_c Q \implies \psi(P) \sqsubseteq_c \psi(Q)$.

Conclusion

- Notion of contextual equivalence with may- and should convergence can also be used for the π-calculus
- Requires to add Stop
- Extension is conservative w.r.t. barbed testing equivalence
- Stop as a programming primitive

Further work

- Encodings of the π-calculus into other program calculi with contextual equivalence
- Extensions of the π-calculus with Stop: recursion, guarded sums, matching prefixes, etc.
- Coarser applicative $\uparrow$-simulation