The π-calculus with Stop

David Sabel

Goethe-Universität, Frankfurt am Main

January 21, 2014
Overview

1. The $\pi$-calculus

2. Process Equivalence in the $\pi$-calculus

3. The $\pi$-calculus with Stop
Introduction

- the \( \pi \)-calculus is a core language for concurrent processes
- is a message passing model
- the control flow of \( \pi \)-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 \( \pi \)-calculus to verify cryptographic protocols
  
  (Abadi & Gordon 1997, Abadi & Fournet 2001)

- \( \pi \)-calculus as a theoretical basis of business processes
  
  (Smith & Fingar, 2002)

- Representation of biochemical processes using the stochastic \( \pi \)-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

\[ P \parallel Q \]

“processes P and Q run concurrently”
Links

\[ x.P \quad \text{or} \quad \overline{x}.P \]

“\( P \) is linked to channel named \( x \)”
The $\pi$-calculus

Process Equivalence in the $\pi$-calculus

The $\pi$-calculus with Stop

Links

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

receive

send

“$P$ is linked to channel named $x$”
Communication

\[
\bar{x}.P \parallel x.Q
\]

“\(P\) (sender) and \(Q\) (receiver) can communicate”
Communication

\[ \overline{x}.P \parallel x.Q \rightarrow P \parallel Q \]

“\(P\) (sender) and \(Q\) (receiver) can communicate”

“\(P\) sent a message to \(Q\)”
Nondeterminism

\[
\overline{x}.P \parallel x.Q \parallel x.R
\]
Nondeterminism

\[ \overline{x}.P \parallel x.Q \parallel x.R \]
Nondeterminism

\[ P \parallel Q \parallel x.R \]

\[ \overline{x}.P \parallel x.Q \parallel x.R \]

\[ x \rightarrow P \parallel Q \parallel x.R \]
Nondeterminism

\[ \overline{x}.P \parallel x.Q \parallel x.R \]

\[ P \parallel Q \parallel x.R \]
Nondeterminism

\[
\begin{array}{c}
\overset{x}{P} | \overset{x}{Q} | \overset{x}{R} \\
\end{array}
\]
Nondeterminism

\[
\overline{x}.P \parallel x.Q \parallel x.R
\]

\[
\begin{align*}
P & \parallel Q \parallel x.R \\
& \hspace{2cm} \downarrow \hspace{2cm} \downarrow \\
P & \parallel x.Q \parallel R
\end{align*}
\]
The $\pi$-calculus

Process Equivalence in the $\pi$-calculus

The $\pi$-calculus with Stop

Messages

$$\overline{x}.P \parallel x.Q \rightarrow P \parallel Q$$
Messages

\[ \overline{x} \langle m \rangle \cdot P \parallel x.Q \rightarrow P \parallel Q \]

"P sends message m along x"
Messages

\[
\overline{x} \langle m \rangle . P \parallel x(y).Q \rightarrow P \parallel Q
\]

“\(P\) sends message \(m\) along \(x\)”
Messages

\[ \overline{x}\langle m \rangle.P | x(y).Q \rightarrow P | Q[m/y] \]

“\(P\) sends message \(m\) along \(x\)”
Approaches to Mobility

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

\[ P_1 \quad P_2 \quad P_3 \quad P_1 \quad P_2 \quad P_3 \]
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)
How to move links?

⇒ Send them as messages!

\[
P \xrightarrow{x} Q \]

\[
R \xrightarrow{y} \]

\[
\begin{align*}
&x(z).z(w).P' \mid x\langle y \rangle.Q' \mid y\langle u \rangle.R' \\
&P \mid Q \mid R
\end{align*}
\]
How to move links?

⇒ Send them as messages!

\[
\begin{align*}
\textcolor{red}{P} & \quad \xrightarrow{x} \quad \textcolor{green}{Q} \\
\textcolor{blue}{R} & \quad \xrightarrow{y} \\
\textcolor{red}{P} & \quad \textcolor{green}{Q} \quad \textcolor{blue}{R} \\
\end{align*}
\]

\[
\begin{align*}
\textcolor{red}{P} & \quad \xrightarrow{x(z).z(w).P'} \quad \textcolor{green}{Q} \quad \xrightarrow{\overline{y}\langle u'\rangle.R'} \\
\textcolor{blue}{R} & \quad \xrightarrow{y(w).P'} \quad \textcolor{green}{Q} \quad \xrightarrow{\overline{y}\langle u'\rangle.R'} \\
\end{align*}
\]

\[
\begin{align*}
(\textcolor{red}{P''} & \quad \textcolor{green}{Q'} \quad \textcolor{blue}{R''}) \\
\textcolor{red}{P''} & \quad \textcolor{green}{Q'} \quad \textcolor{blue}{R''}
\end{align*}
\]
Private Communication

\[ \nu x. P \]

“channel \( x \) is private for \( P \)”

Example: \[ \nu x. (x(y).P || \overline{x}\langle z\rangle.Q) || \overline{x}\langle z'\rangle.R \]

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

"! P means \( P \upharpoonright P \upharpoonright P \upharpoonright \ldots \)"

infinitely many parallel copies of \( P \)
Syntax of a minimalistic (synchronous) \( \pi \)-calculus

Syntax of \textbf{calculus \( \Pi \)} where \( x, y \in \mathcal{N} \) is a countably-infinite set of \textit{names}

\[
P ::= \pi.P \quad \text{(action)} \\
| P_1 \parallel P_2 \quad \text{(parallel composition)} \\
| !P \quad \text{(replication)} \\
| 0 \quad \text{(silent process)} \\
| \nu x.P \quad \text{(name restriction)}
\]

\[
\pi ::= x(y) \quad \text{input} \\
| \overline{x} \overline{y} \quad \text{output}
\]

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 [.] at process position
Structural Congruence $\equiv$

Smallest congruence on processes satisfying the following axioms

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

Remark:

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

Reduction rule for interaction:

\[ x(y).P \parallel \bar{x}(v).Q \xrightarrow{ia} P[v/y] \parallel Q. \]

Reduction contexts \( \mathcal{D} \):

\[ D \in \mathcal{D} ::= \cdot | D \parallel P | P \parallel D | \nu x.D \]

Standard reduction \( \xrightarrow{sr} \):

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

\[ P \xrightarrow{sr} Q \]

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 ⊕

\[ P \oplus Q := \nu x, y.(\overline{x}\langle y \rangle.P \parallel \overline{x}\langle y \rangle.Q \parallel x(z).0) \]

\[ (x, y \notin (\text{fn}(P) \cup \text{fn}(Q))) \]

\[ P \oplus Q \equiv \nu x, y.([x(z).0 \parallel \overline{x}\langle y \rangle.P] \parallel \overline{x}\langle y \rangle.Q) \]

\[ \xrightarrow{sr} \nu x, y.([P \parallel 0] \parallel \overline{x}\langle y \rangle.Q) \]

\[ \equiv P \parallel \nu x, y.(\overline{x}\langle y \rangle.Q) \]

“garbage”

Other reduction possibility:

\[ P \oplus Q \xrightarrow{sr} Q \parallel\nu x, y.(\overline{x}\langle y \rangle.P) \]

“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

- $\nu X.(x(y).P_1 \parallel P_2)$ with $x \notin X$ has an input capability.
- $\nu X.(\overline{x}(y).P_1 \parallel P_2)$ with $x \notin X$ has an output capability.

Two cases:

- $y \notin X$: The emitted name is free
- $y \in X$: 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**

- \( P \overset{x}{\rightarrow} \) iff \( P \equiv \nu X. (x(y).P' | P'') \) where \( x \notin X \),
- \( P \overset{x}{\rightarrow} \) iff \( P \equiv \nu X. (\overline{x}(y).P' | P'') \) where \( x \notin X \).

**May-barb and Should-barb**

For \( \mu \in \{ x, \overline{x} \} \):

- \( P \upharpoonleft_{\mu} \) (\( P \) may have a barb on \( x \)) iff \( \exists Q : P \xrightarrow{st,*} Q \land Q \overset{\mu}{\rightarrow} \), and
- \( P \upharpoonright_{\mu} \) (\( P \) should have a barb on \( x \)) iff \( \forall Q : P \xrightarrow{st,*} Q \implies Q \upharpoonleft_{\mu} \).

Notations:

- \( P \upharpoonleft_{\mu} \) iff \( \neg(P \upharpoonright_{\mu}) \)
- \( P \upharpoonright_{\mu} \) iff \( \neg(P \upharpoonleft_{\mu}) \)
Barbed Testing

For $\mu \in \{x, \bar{x}\}$, and $\xi \in \{\downmu, \downdownmu, \upmu, \upupmu\}$

- **barbed may-testing preorder:**

  $$P \sqsubseteq_{c,\text{may}} Q \text{ iff } \forall x \in \mathcal{N}, \mu \in \{x, \bar{x}\}, C \in \mathcal{C} : C[P] \downmu \Rightarrow C[Q] \downmu$$

- **barbed should-testing preorder:**

  $$P \sqsubseteq_{c,\text{should}} Q \text{ iff } \forall x \in \mathcal{N}, \mu \in \{x, \bar{x}\}, C \in \mathcal{C} : C[P] \downdownmu \Rightarrow C[Q] \downdownmu$$

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

- **barbed testing equivalences**

  $$\boxtimes_{c,\text{may}} := \sqsubseteq_{c,\text{may}} \cap (\sqsubseteq_{c,\text{may}})^{-1}$$
  $$\boxtimes_{c,\text{should}} := \sqsubseteq_{c,\text{should}} \cap (\sqsubseteq_{c,\text{should}})^{-1}$$
  $$\boxtimes_c := (\boxtimes_c) \cap (\boxtimes_c)^{-1}$$
Examples

Barbed testing equivalence is coarse:

\[
\begin{align*}
(a(z).0 \oplus b(z).0) \oplus c(z).0 & \sqsubset_c a(z).0 \oplus (b(z).0 \oplus c(z).0) \\
\end{align*}
\]

Barbed \textit{may}-testing is too coarse:

\[
\begin{align*}
a(z).0 & \sqsubset_{c, \textit{may}} a(z).0 \oplus 0 \\
\end{align*}
\]

Barbed \textit{should}-testing is finer:

\[
\begin{align*}
a(z).0 & \nsubseteq_{c, \textit{should}} a(z).0 \oplus 0 \\
\end{align*}
\]
Alternative Definitions of Barbed Testing

**Theorem (Should-Testing includes May-Testing)**

\[ \preceq_{c,\text{should}} \subset (\preceq_{c,\text{may}})^{-1} \text{ and thus } \square_c = \square_{c,\text{should}}. \]

**Theorem**

\[ \preceq_{c,\text{should}} = \preceq_{c,\parallel x} = \preceq_{c,\parallel} \text{ where} \]

\[
P \preceq_{c,\text{should}} Q \ := \ \forall x \in \mathcal{N}, \mu \in \{x, \overline{x}\}, C \in C : C[P] \parallel_{\mu} \rightarrow C[Q] \parallel_{\mu}
\]

\[
P \preceq_{c,\parallel x} Q \ := \ \forall C \in C : C[P] \parallel x \rightarrow C[Q] \parallel x
\]

\[
P \preceq_{c,\parallel} Q \ := \ \forall C \in C : (\exists x : C[P] \parallel x) \rightarrow (\exists x : C[Q] \parallel x)
\]

(analogous for barbed may-testing)

Proofs:

- (Fournet & Gonthier 2005) for the asynchronous \( \pi \)-calculus
- for \( \Pi \) also included in (S. & Schmidt-Schauß 2014, submitted)
**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 $\exists e : e \rightarrow^* e' \land e'$ is successful
- **Should-convergence:** $e \Downarrow$ iff $\forall e' : e \rightarrow^* e' \implies e' \downarrow$

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 \(\text{calc}_1, \text{calc}_2\):

Does a translation \(\psi : \text{calc}_1 \rightarrow \text{calc}_2\) exist, s.t.

- \(\psi\) is adequate: \(\psi(e_1) \sim_{c,\text{calc}_2} \psi(e_2) \implies e_1 \sim_{c,\text{calc}_1} e_2\)
- \(\psi\) is full-abstract: \(\psi(e_1) \sim_{c,\text{calc}_2} \psi(e_2) \iff e_1 \sim_{c,\text{calc}_1} e_2\)

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

- barbed testing is close to contextual equivalence, but:
  - \( P \xrightarrow{\langle x \rangle} \) and \( P \xrightarrow{sr} P' \) with \( \neg (P' \xrightarrow{\langle x \rangle}) \) is possible:

\[
(x(y).0 \mid \langle x \rangle \langle y \rangle .0) \xrightarrow{sr} 0
\]

- hence \( \xrightarrow{\langle x \rangle} \) is not a notion of successful termination.
The \( \pi \)-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 $\pi$-calculus with Stop: $\Pi_{\text{Stop}}$

$$P ::= \pi.P \quad \text{(action)}$$
$$\mid P_1 P_2 \quad \text{(parallel composition)}$$
$$\mid !P \quad \text{(replication)}$$
$$\mid 0 \quad \text{(silent process)}$$
$$\mid \nu x.P \quad \text{(name restriction)}$$
$$\mid \text{Stop} \quad \text{(success)}$$

$$\pi ::= x(y) | \overline{x}(y)$$

Further adaptations:

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

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

Lemma: $P$ successful and $P \xrightarrow{sr} P'$ $\implies$ $P'$ successful.
May-convergence: \( P \downarrow \) iff \( \exists P : P \xrightarrow{sr,*} P' \land P' \) 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: \( \leq_c := \leq_{c,\downarrow} \land \leq_{c,\downarrow} \)
- Contextual equivalence: \( \sim_c := \leq_c \land (\leq_c)^{-1} \)
Conservativity

Theorem

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

Consequences:

- Contextual equivalence in $\Pi_{\text{Stop}}$ is compatible with existing process equivalences in $\Pi$.
- For Stop-free processes: $\approx_{b,\text{strong}} \subset \approx_b \subset \oplus_c = \sim_c$
Proof Tools: Context Lemma

Contexts $[\cdot] \triangleright R$ 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 $\forall \sigma, R : \sigma(P) \triangleright R \downarrow \implies \sigma(Q) \triangleright R \downarrow$

- $P \leq_{c} Q$ iff $\forall \sigma, R :$
  
  \begin{align*}
  (\sigma(P) \triangleright R \downarrow \implies \sigma(Q) \triangleright R \downarrow) \land (\sigma(P) \triangleright R \downarrow \implies \sigma(Q) \triangleright R \downarrow)
  \end{align*}
Proof Tools: Action-Semantics

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

**Definition**

- **Open input:** If $P \equiv \nu X.(x(y).P_1 \parallel P_2)$ (with $x \notin X$) then
  
  $P \xrightarrow{\overline{x}(z)} \nu X.(P_1[z/y] \parallel P_2)$ (for all $z \in N$)

- **Open output:** If $P \equiv \nu X.(\overline{x}(y).P_1 \parallel P_2)$ with $x, y \notin X$, then
  
  $P \xrightarrow{x(y)} \nu X.(P_1 \parallel P_2)$

- **Bound output:** If $P \equiv \nu X, \nu y.(\overline{x}(y).P_1 \parallel P_2)$ with $x \notin X$, then
  
  $P \xrightarrow{\nu y.x(y)} \nu X.(P_1 \parallel P_2)$. 

Proof Tools: Similarity

**May-similarity**

A binary relation $\eta$ 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 $Q \xrightarrow{sr,*} \alpha \xrightarrow{} Q'$ and $(P', Q') \in \eta$.

Applicative $\downarrow$-similarity $\preccurlyeq_{b,\downarrow}$ is the largest applicative simulation.

Full applicative $\downarrow$-similarity $\preccurlyeq_{b,\downarrow}$: $P \preccurlyeq_{b,\downarrow} Q$ iff $\forall \sigma: \sigma(P) \preccurlyeq_{b,\downarrow} \sigma(Q)$

**Theorem (Soundness)**

$\preccurlyeq_{b,\downarrow} \subseteq \preccurlyeq_{c,\downarrow}$
May-Divergence Similarity

A binary relation $\eta$ 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 $Q \xrightarrow{sr,*} \alpha \xrightarrow{} Q'$ and $(P', Q') \in \eta$.
- $Q \prec b,\downarrow P$

Applicative $\uparrow$-similarity $\prec b,\uparrow$ is the largest applicative $\uparrow$-simulation. Full applicative $\uparrow$-similarity: $P \prec b,\uparrow Q$ iff $\forall \sigma : \sigma(P) \prec b,\uparrow \sigma(Q)$
Proof Tools: Similarity (3)

**Theorem (Soundness)**

- \((P \preceq b, \downarrow Q \land Q \preceq b, \uparrow P) \implies P \preceq_c Q\)
- \(P \preceq b, \uparrow Q \implies Q \preceq b, \downarrow P\)
- \((P \preceq b, \uparrow Q \land Q \preceq b, \uparrow P) \implies P \sim_c Q\)

Note that \(\preceq\) is fine-grained, e.g.

for \(A := a(x).0\), \(B := b(x).0\), \(C := b(x).0\):

\[
(A \oplus B) \oplus C \preceq b, \uparrow A \oplus (B \oplus C)
\]

although

\[
(A \oplus B) \oplus C \sim_c A \oplus (B \oplus C)
\]

Open problem: find a coarser sound similarity for \(\uparrow\)
Correctness of Deterministic Interaction

For all processes $P, Q$ the following equation holds:

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

Proof: Let

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

$S$ and $S^{-1}$ 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. $!P | !!P \sim_c !P$.
3. $!(P | Q) \sim_c !P | !Q$.
4. $!0 \sim_c 0$.
5. $!\text{Stop} \sim_c \text{Stop}$.
6. $!(P | Q) \sim_c !{(P | Q)} | P$.
7. $x(y).vz.P \sim_c vz.x(y).P$ if $z \notin \{x, y\}$.
8. $\overline{x}(y).vz.P \sim_c vz.\overline{x}(y).P$ if $z \notin \{x, y\}$.

Proof: $S_i \cup \preceq_b, \uparrow$ and $S_i^{-1} \cup \preceq_b, \uparrow$ are applicative $\uparrow$-simulations where $S_i := \{(R | l_i, R | r_i) | \text{ for all } R, P, Q\}$, and $l_i, r_i$ are the left and right hand side of the $i^{th}$ equation.
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 $P \not\succeq_c Q$.
4. Stop is the greatest process w.r.t. $\leq_c$.
5. $0$ 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 \( \Pi \)”:

**Theorem**
There is no surjective translation \( \psi : \Pi_{\text{Stop}} \rightarrow \Pi \) s.t. for all \( P, Q \):
\[ P \leq_c Q \implies \psi(P) \subseteq_c \psi(Q). \]
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 $\pi$-calculus into other program calculi with contextual equivalence
- Extensions of the $\pi$-calculus with Stop: recursion, guarded sums, matching prefixes, etc.
- Coarser applicative $\uparrow$-simulation