MPRI Concurrency (course number 2-3) 2005-2006: \(\pi\)-calculus
2005-11-02

http://pauillac.inria.fr/~leifer/teaching/mpri-concurrency-2005/

James J. Leifer
INRIA Rocquencourt

James.Leifer@inria.fr
About the lectures

The MPRI represents a transition from student to researcher. So...

• Interrupting me with questions is good.
• Working through a problem without already knowing the answer is good.
• I’ll make mistakes. 8-)

About me

• 1995–2001: Ph.D. student of Robin Milner’s in Cambridge, UK
• 2001–2002: Postdoc in INRIA Rocquencourt, France
• 2002–: Research scientist in INRIA Rocquencourt, France
• November 2004: voted against W (who, despite this, was elected for the first time)
Books


Tutorials available online

- Joachim Parrow. “An introduction to the pi-calculus”.
  http://user.it.uu.se/~joachim/intro.ps
Today’s plan

• syntax

• reduction semantics and structural congruence

• labelled transitions

• bisimulation
Syntax

\[ P ::= \overline{xy}.P \quad \text{output} \]
\[ x(y).P \quad \text{input} \quad (y \text{ binds in } P) \]
\[ \nu x. P \quad \text{restriction (new)} \quad (x \text{ binds in } P) \]
\[ P \mid P \quad \text{parallel (par)} \]
\[ 0 \quad \text{empty} \]
\[ !P \quad \text{replication (bang)} \]
\[ ... \]

Significant difference from CCS: channels carry names.
Free names

The free names of $P$ are written $\text{fn}(P)$.

**Example:** $\text{fn}(0) = \emptyset$; $\text{fn}(\overline{xy}.z(y).0) = \{x, y, z\}$.

**Exercise:** Calculate $\text{fn}(z(y).\overline{xy}.0)$; $\text{fn}(\nu z.(z(y).\overline{xy}) | \overline{yz})$.

Formally:

\[
\begin{align*}
\text{fn}(\overline{xy}.P) &= \{x, y\} \cup \text{fn}(P) \\
\text{fn}(x(y).P) &= \{x\} \cup (\text{fn}(P) \setminus \{y\}) \\
\text{fn}(\nu x.P) &= \text{fn}(P) \setminus \{x\} \\
\text{fn}(P | P') &= \text{fn}(P) \cup \text{fn}(P') \\
\text{fn}(0) &= \emptyset \\
\text{fn}(!P) &= \text{fn}(P)
\end{align*}
\]

Alpha-conversion

We consider processes up to alpha-conversion: provided $y' \notin \text{fn}(P)$, we have

\[
\begin{align*}
x(y).P &= x(y').\{y'/y\}P \\
\nu y.P &= \nu y'.\{y'/y\}P
\end{align*}
\]

**Exercise:** Freshen all bound names: $\nu x.(x(x).\overline{xx}) | x(x)$
Reduction ($\rightarrow$)

We say that $P$ reduces to $P'$, written $P \rightarrow P'$, if this can be derived from the following rules:

\[
\begin{align*}
\bar{xy}.P \mid x(u).Q & \rightarrow P \mid \{y/u\}Q \quad \text{(red-comm)} \\
P & \rightarrow P' \\
\hline
P \mid Q & \rightarrow P' \mid Q \quad \text{(red-par)} \\
\hline
P & \rightarrow P' \\
\nu x.P & \rightarrow \nu x.P' \quad \text{(red-new)}
\end{align*}
\]

**Example:** $\nu x. (\bar{xy} \mid x(u).\bar{uz}) \rightarrow \nu x. (0 \mid \bar{yz})$

As currently defined, reduction is too limited:

\[
\begin{align*}
(\bar{xy} \mid 0) \mid x(u) & \not\rightarrow \\
\nu w.\bar{xy} \mid x(u) & \not\rightarrow
\end{align*}
\]
Structural congruence ($\equiv$)

The smallest equivalence relation such that:

\[
\begin{align*}
    P \mid (Q \mid S) & \equiv (P \mid Q) \mid S & \text{(str-associ)} \\
    P \mid Q & \equiv Q \mid P & \text{(str-commut)} \\
    P \mid 0 & \equiv P & \text{(str-id)} \\
    \nu x. \nu y. P & \equiv \nu y. \nu x. P & \text{(str-swap)} \\
    \nu x. 0 & \equiv 0 & \text{(str-zero)} \\
    \nu x. P \mid Q & \equiv \nu x. (P \mid Q) & \text{if } x \not\in \text{fn}(Q) & \text{(str-ex)} \\
    !P & \equiv P \mid !P & \text{(str-repl)}
\end{align*}
\]

And congruence rules:

\[
\begin{align*}
    P & \equiv P' & \text{(str-par-l)} \\
    P \mid Q & \equiv P' \mid Q & \text{(str-par-l)} \\
    \nu x. P & \equiv \nu x. P' & \text{(str-new)}
\end{align*}
\]

Note: we don’t close up by input or output prefixing.
Fixing reduction

We close reduction by structural congruence:

\[
P \equiv \longrightarrow \equiv P'
\]

\[
\frac{P \longrightarrow P'}{P \longrightarrow P'}
\]  \hspace{1cm} \text{(red-str)}

Exercise: Calculate the reductions of \( \nu y. (\overline{x} y \mid y(u).\overline{u} z) \mid x(w).\overline{w} v \) and \( \overline{x} y \mid \nu y. (x(u).\overline{u} w \mid y(v)) \)
Application of new binding: from polyadic to monadic channels

Let us extend our notion of monadic channels, which carry exactly one name, to polyadic channels, which carry a vector of names, i.e.

\[ P ::= x\langle y_1, \ldots, y_n \rangle . P \quad \text{output} \]
\[ x(y_1, \ldots, y_n) . P \quad \text{input } (y_1, \ldots, y_n \text{ bind in } P) \]

Is there an encoding from polyadic to monadic channels? We might try:

\[ \llbracket x\langle y_1, \ldots, y_n \rangle . P \rrbracket = xy_1 \ldots xy_n . \llbracket P \rrbracket \]
\[ \llbracket x(y_1, \ldots, y_n) . P \rrbracket = x(y_1) \ldots x(y_n) . \llbracket P \rrbracket \]

but this is broken! Can you see why? The right approach is use new binding:

\[ \llbracket x\langle y_1, \ldots, y_n \rangle . P \rrbracket = \nu z.(xz.\bar{z}y_1 \ldots \bar{z}y_n . \llbracket P \rrbracket) \]
\[ \llbracket x(y_1, \ldots, y_n) . P \rrbracket = x(z) . z(y_1) \ldots z(y_n) . \llbracket P \rrbracket \]

where \( z \notin \text{fn}(P) \) in both cases. (We also need some well-sorted assumptions.)
Application of new binding: from synchronous to asynchronous output

In distributed computing, sending and receiving messages may be asymmetric: we clearly know when we have received a message but not necessarily when a message we sent has been delivered. (Think of email.)

\[ P ::= \overline{x}y \quad \text{output} \]
\[ x(y).P \quad \text{input (}y\text{ binds in }P\text{)} \]

Nonetheless, one can always achieve synchronous sends by using an acknowledgement protocol:

\[ \llbracket \overline{x}y.P \rrbracket = \nu z.(\overline{x}\langle y, z \rangle \mid z().\llbracket P \rrbracket) \]
\[ \llbracket x(y).P \rrbracket = x(y, z).(\overline{z}\langle \rangle \mid \llbracket P \rrbracket) \]

provided \( z \not\in \text{fn}(P) \) in both cases.

But this is cheating since the encoding relies on being able to send tuples (e.g. \( \overline{x}\langle y, z \rangle \)). Can you see how to use only monadic communication?
Labels

The labels $\alpha$ are of the form:

\[
\alpha ::= xy \quad \text{output} \\
\text{\overline{x}(y)} \quad \text{bound output} \\
xy \quad \text{input} \\
\tau \quad \text{silent}
\]

The free names $\text{fn}(\alpha)$ and bound names $\text{bn}(\alpha)$ are defined as follows:

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$xy$</th>
<th>$\text{\overline{x}(y)}$</th>
<th>$xy$</th>
<th>$\tau$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{fn}(\alpha)$</td>
<td>${x, y}$</td>
<td>${x}$</td>
<td>${x, y}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\text{bn}(\alpha)$</td>
<td>$\emptyset$</td>
<td>${y}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
Labelled transitions \((P \xrightarrow{\alpha} P')\)

Labelled transitions are of the form \(P \xrightarrow{\alpha} P'\) and are generated by:

\[
\frac{\overline{xy}.P \xrightarrow{\overline{xy}} P}{P \xrightarrow{\alpha} P'} \text{ (lab-out)}
\]

\[
\frac{x(y).P \xrightarrow{xz} \{z/y\}P}{P \xrightarrow{\alpha} P'} \text{ (lab-in)}
\]

\[
\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \text{ if } \text{bn}(\alpha) \cap \text{fn}(Q) = \emptyset \text{ (lab-par-l)}
\]

\[
\frac{P \xrightarrow{\alpha} P'}{\nu y.P \xrightarrow{\alpha} \nu y.P'} \text{ if } y \notin \text{fn}(\alpha) \cup \text{bn}(\alpha) \text{ (lab-new)}
\]

\[
\frac{\nu y.P \xrightarrow{\nu y} P'}{P \xrightarrow{\overline{xy}} P'} \text{ if } y \neq x \text{ (lab-open)}
\]

\[
\frac{P \xrightarrow{\overline{xy}} P'}{Q \xrightarrow{xy} Q'} \text{ if } y \notin \text{fn}(Q) \text{ (lab-comm-l)}
\]

\[
\frac{P \xrightarrow{\overline{xy}} P'}{Q \xrightarrow{\sigma} Q'} \text{ if } y \notin \text{fn}(Q) \text{ (lab-comm-l)}
\]

\[
\frac{P \xrightarrow{\overline{xy}} P'}{Q \xrightarrow{\nu y.(P' \mid Q')} \text{ if } y \notin \text{fn}(Q) \text{ (lab-close-l)}
\]

\[
\frac{P \mid !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'} \text{ (lab-bang)}
\]

plus symmetric rules (lab-par-r), (lab-comm-r), (lab-close-r).
Labelled transitions and structural congruence

Theorem:
1. \( P \xrightarrow{\tau} P' \) iff \( P \equiv \tau P' \).
2. \( P \equiv \alpha \xrightarrow{\alpha} P' \) implies \( P \xrightarrow{\alpha} \equiv P' \).

Exercise: Why does the converse of the second not hold?

Exercise: Show that the following pair of processes are both in (\( \rightarrow \)) and (\( \tau \equiv \)):

\[
\nu z.\bar{x}z | x(u).y \bar{y}u \quad \nu z.\bar{y}z
\]

Fun with side conditions

Exercise: Show that the side condition on (\text{lab-par-l}) is necessary by considering the process \( \nu y.(\bar{x}y.y(u)) | \bar{z}v \) and an alpha variant.
Adding sum

\[ P ::= M \]
\[ P \parallel P \] parallel (par)
\[ \nu x. P \] restriction (new) \((x \text{ binds in } P)\)
\[ !P \] replication (bang)

\[ M ::= \overline{xy}. P \] output
\[ x(y). P \] input \((y \text{ binds in } P)\)
\[ M + M \] sum
\[ 0 \]

Changes:

- **structural congruence:** + is associative and commutative with identity 0.
- **reduction:** \((\overline{xy}. P + M) \parallel (x(u). Q + N) \longrightarrow P \parallel \{y/u\} Q\).
- **labelled transition:** \(M + \overline{xy}. P + N \overset{\overline{xy}}{\longrightarrow} P\)
  \(M + x(y). P + N \overset{xz}{\longrightarrow} \{z/y\} P\)
Exercises for next lecture

1. Define an encoding from the monadic synchronous \( \pi \)-calculus to the monadic asynchronous \( \pi \)-calculus.

2. Prove that if \( P \xrightarrow{\bar{xy}} P' \) then there exist \( P_0 \), \( P_1 \), and \( \vec{z} \) such that

\[
P \equiv \nu \vec{z}.(\bar{x}y.P_0 \mid P_1)
\]

\[
P' \equiv \nu \vec{z}.(P_0 \mid P_1)
\]

\[
\{x, y\} \cap \vec{z} = \emptyset
\]

NB: the notation \( \nu \vec{z}.P \) is merely a convenient way of expressing a series of new bindings:

\[
\nu \vec{z}.P = \begin{cases} 
P & \text{if } \vec{z} \text{ is empty} \\
\nu w.(\nu \vec{w}.P) & \text{if } \vec{z} = w \vec{w} 
\end{cases}
\]