The $\pi$-Calculus

Overview

- Syntax
- Reaction, actions, and transitions

References

- Robin Milner, “Communicating and Mobil Systems”
- Davide Sangiorgi and David Walker, “The $\pi$-calculus: A Theory of Mobile Processes”
- Robin Milner, Joachim Parrow, David Walker, “A Calculus of Mobile Processes, Part I+II”
From CCS to $\pi$-Calculus

- The $\pi$-calculus is a model of concurrent computation based upon the notion of *naming*.

- The $\pi$-calculus is a calculus in which the topology of communication can evolve dynamically during evaluation.

- In the $\pi$-calculus communication links are identified by *names*, and computation is represented purely as the communication of names across links.

- The $\pi$-calculus is an extension of the process algebra CCS, following the work by Engberg and Nielsen who added mobility to CCS while preserving its algebraic properties.
The π-Calculus - Basic Ideas

- The most primitive in the π-calculus is a name, Names, infinitely many, are \( x, y, \ldots \in \mathbb{N} \); they have no structure.

- In the π-calculus we only have one other kind of entity: a process. We use \( P, Q, \ldots \) to range over processes.

- Processes evolve by performing actions. The capabilities for actions are expressed via prefixes, of which there are four kinds:

\[
\pi ::= \overline{x}y | x(z) | \tau | [x = y]\pi
\]
Prefixes

- **Output prefix:**
  
  The output prefix $\bar{x}y$ denotes a process’s capability to send the name ‘y’ via the name ‘x’.

- **Input prefix:**

  The input prefix $x(z)$ denotes a process’s capability to receive any name, say ‘y’, via the name ‘x’. The role of ‘z’ is that of placeholder that will be substituted with the received name.

- **Silent prefix:**

  The silent prefix $\tau$ denotes an unobservable action. As in CCS, the silent prefix can be thought of as expressing an internal action of a process.

- **Match prefix:**

  The match prefix $[x = y]\pi$ is a conditional capability, that is, $\pi$ if $x$ and $y$ are the same name.
The $\pi$-Calculus - Syntax

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

$$P ::= \Sigma_{i \in I} \pi_i.P_i \mid P_1 \mid P_2 \mid \nu x P \mid !P$$

where $I$ is any finite indexing set. The processes $\Sigma_{i \in I} \pi_i.P_i$ are called summations or sums. We often write simply $M_1 + M_2$ for $\Sigma_{i \in \{1,2\}} \pi_i.P_i$. Note, however, that summation binds more tightly than composition, restriction, and replication. If $I = \emptyset$, then $\Sigma_{i \in \emptyset} \pi_i.P_i$ is the empty sum, written $0$. 


Restriction

In the restriction $\nu x \ P$, the scope of the name $x$ is restricted to $P$. Components of $P$ can use $x$ to interact with one another, but not with other processes.

The process $\nu x ((x(z).\overline{zy}.0 + \overline{wv}.0) | \overline{xu}.0)$ has only two capabilities: to send ‘$v$’ via ‘$w$’, and to evolve invisibly as an effect of an interaction between its components via ‘$x$’.

The scope of a restriction may change as the result of an interaction between processes. Unlike restriction in concurrent processes (or CCS), names in $\pi$-calculus can be transmitted. In the case of a restricted name this means that this name may extrude its original scope.
Replication

The replication \(!P\) can be thought of as an infinite composition \(P \mid P \mid \ldots\) or, equivalently, a process satisfying the equation \(!P = P \mid !P\).

Replication is an operator that makes it possible to express infinite behaviors.

The process \(!x(z). !y z. 0\) can receive names via ‘x’ repeatedly, and can repeatedly send via ‘y’ any name it does receive.
Link Passing – Free Channel

The agent $P$, with $x \notin \text{fn}(P')$, has a link ‘$x$’ to $R$, and wishes to pass ‘$x$’ along its link ‘$y$’ to $Q$. $Q$ is willing to receive it.

$$\bar{y}x.P \mid y(z).Q \mid R \xrightarrow{\tau} P' \mid \{x/z\}Q' \mid R$$

$Q''$ in the diagram is $\{x/z\}Q'$. 
Link Passing – Private Channel

The agent P, with \( x \notin fn(P') \), has a link ‘x’ to R, and wishes to pass ‘x’ along the private ‘y’ to Q. The scope of the private name include both P and Q.

\[
vy (\overline{yx}.P | y(z).Q) | R \xrightarrow{\tau} vy (P' | \{x/z\}Q') | R
\]

\[
v_y (\overline{yx}.P | y(z).Q) = \tau.v_y (P' | \{x/z\}Q')
\]
Scope Intrusion

The agent $P$, with $x \in \text{fn}(P')$, has a link ‘$x$’ to $R$, and wishes to pass ‘$x$’ along its link ‘$y$’ to $Q$. $Q$ is willing to receive it, but already possesses a private link ‘$x$’ to $S$. The latter needs to be renamed to avoid the capture of the received free name ‘$x$’ in $S$.

$$\bar{y}.x.P | v(x(y(z).Q|S)) | R \xrightarrow{\tau} P' | v(x'(\{x/z\}\{x'/x\}Q'|\{x'/x\}S)) | R$$
The agent $P$, with $x \in \text{fn}(P')$, has a link ‘$x$’ to $R$, but this link is private. $P$ wishes to pass ‘$x$’ along its link ‘$y$’ to $Q$. $Q$ is willing to receive it, and possesses no ‘$x$’-link. This situation is exactly that of a program $P$, with a local variable modeled by a storage register $R$, passing $R$ to a procedure $Q$, which takes its parameter by reference, not by value.
Now suppose both Q and P' possess no 'x'-link.

\[ \forall x (\bar{y}x.P \mid R) \mid y(z).Q \xrightarrow{\tau} \forall x (P' \mid R \mid \{x/z\}Q') \]

However, there is a general law:

\[ \forall x (P_1 \mid P_2) = P_1 \mid \forall x P_2 \]

So

\[ \forall x (P' \mid R \mid \{x/z\}Q') = P' \mid \forall x (R \mid \{x/z\}Q') \]
In each of $x(z).P$ and $\nu z P$, the name ‘$z$’ occurs bound with scope $P$.

An occurrence of a name in a process is free if it is not bound.

We write $fn(P)$ for the set of names that have free occurrences in $P$.

$$fn((\bar{z}y.0 + \bar{w}v.0) | \bar{x}u.0) = \{z, y, w, v, x, u\}$$

$$fn(vx ((x(z).\bar{z}y.0 + \bar{w}v.0) | \nu u \bar{x}u.0)) = \{y, w, v\}$$
Substitution

The process, $\sigma P$, obtained by applying the name substitution $\sigma$ to $P$ is defined as follows:

- $\sigma 0 = 0$
- $\sigma (\pi.P) = \sigma \pi. \sigma P$
- $\sigma (M_1 + M_2) = \sigma M_1 + \sigma M_2$
- $\sigma (P_1 | P_2) = \sigma P_1 | \sigma P_2$
- $\sigma (\nu z P) = \nu z \sigma P$
- $\sigma (!P) = !\sigma P$

Note $\sigma (\sum_{i \in I} \pi_i.P_i) = \sum_{i \in I} \sigma \pi_i. \sigma P_i$. 
Process Context

- An occurrence of 0 in a process is degenerate if it is the left or right term in a sum $M_1 + M_2$, and non-degenerate otherwise.

- A $\pi$-calculus process context $C$ is obtained when the hole $\bullet$ replaces a non-degenerate occurrence of 0 in a process term given by the $\pi$-calculus grammar.

Examples:

- $\nu z ([\bullet] | !z(w).wz.0)$

- $x(z).!vw (\tilde{z}w.[\bullet] + y(v).0)$

An equivalence relation $S$ on $\pi$-calculus processes is a congruence if $(P, Q) \in S$ implies $(C[P], C[Q]) \in S$ for every context $C$. 
Structural Congruence Rules

- Change of bound names (alpha-conversion)
- Axioms:

\[ \forall x. P \equiv x. P \]
\[ M_1 + (M_2 + M_3) \equiv (M_1 + M_2) + M_3 \]
\[ M_1 + M_2 \equiv M_2 + M_1 \]
\[ M + 0 \equiv M \]
\[ P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3 \]
\[ P_1 \mid P_2 \equiv P_2 \mid P_1 \]
\[ P \mid 0 \equiv P \]
\[ \forall z \forall w. P \equiv \forall w \forall z. P \]
\[ \forall z. 0 \equiv 0 \]
\[ \forall z. (P_1 \mid P_2) \equiv P_1 \mid \forall z. P_2 \text{ if } z \notin \text{fn}(P_1) \]
\[ \forall \neg P \equiv P \mid \forall \neg P \]
A $\pi$-calculus process expression $\nu \tilde{x} (M_1 | \ldots | M_n | R_1 | \ldots | R_m)$, where each $M_i$ is a non-empty summation, is said to be in standard form.

**Theorem:**
Every $\pi$-calculus process is structurally congruent to a standard form.

**Proof:**
For any restriction ‘$\nu$ a’ inside a summation, we can bring it to the outermost by using alpha-conversion (if necessary) followed by the rule $\nu$ a $(P|Q) \equiv \nu$ a P|Q in conjunction with some of the laws of structural congruence.
Reaction Rules

The reaction relation $\rightarrow$ over $P^{\pi}$ contains exactly those transitions that can be inferred from the rules in the following table:

- **TAU**: $\tau.P + M \rightarrow P$

- **INTER**: $(x(z).P + M) | (\overline{xy}.Q + N) \rightarrow \{y/z\}P | Q$

- **PAR**: $P \rightarrow P'$
  
  $P|Q \rightarrow P'|Q$

- **RES**: $P \rightarrow P'$
  
  $\nu a P \rightarrow \nu a P'$

- **STRUCT**: $P \equiv Q$
  
  $P \rightarrow P'$
  
  $P' \equiv Q'$
  
  $Q \rightarrow Q'$