CSCI.6500/4500 Distributed Computing over the Internet—Programming Distributed Computing Systems (Varela)—Sections 3.1,3.2,3.4

Instructor: Carlos Varela
Rensselaer Polytechnic Institute
Spring 2021
**Π Calculus Syntax**

\[ \alpha ::= \overline{cx} \quad \text{Write } x \text{ on channel } c \]

\[ c(x) \quad \text{Read } x \text{ on channel } c \]

\[ \tau \quad \text{No interaction} \]

\[ P, Q ::= 0 \quad \text{Empty process} \]

\[ \alpha.P \quad \text{Prefixed process} \]

\[ P \mid Q \quad \text{Concurrent composition of } P \text{ and } Q \]

\[ P + Q \quad \text{Nondeterministic choice of } P \text{ or } Q \]

\[ (\nu c)P \quad \text{New channel } c, \text{ scope restricted to } P \]

\[ \text{if } x = y \text{ then } P \quad \text{Conditional execution (match)} \]

\[ \text{if } x \neq y \text{ then } P \quad \text{Conditional execution (mismatch)} \]

\[ A(y_1, \ldots, y_n) \quad \text{Process invocation} \]

\[ \Delta ::= A(x_1, \ldots, x_n) \triangleq P \quad \text{Process declaration} \]
Exercise: two-way communication

Reduce the following $\pi$-calculus expressions:

$$(\nu x)\bar{r}x.x(y).\bar{y}.Q \mid r(e).\bar{e}z.z.P$$
Reference Cell in the \( \pi \) calculus

A reference cell can be defined in the \( \pi \) calculus as follows:

\[
\begin{align*}
\text{Ref}(g, s, i) & \triangleq (\nu l)(\bar{l}i \mid \text{GetServer}(l, g) \mid \text{SetServer}(l, s)) \\
\text{GetServer}(l, g) & \triangleq !g(c).l(v).(\bar{c}v \mid \bar{l}v) \\
\text{SetServer}(l, s) & \triangleq !s(c, v').l(v).(\bar{c} | \bar{l}v')
\end{align*}
\]

A client of the reference cell:

\[
(\nu c)\bar{s}\langle c, v \rangle.c.(\nu d)\bar{g}d.d(e).P
\]

- We use process declarations (e.g., GetServer), scope restriction (e.g., \((\nu l)\)), and replication (\(!P\)).
- What about \(s(c, v')\) and \(\bar{s}\langle c, v \rangle\)?
Exercise: sending multiple values over a channel

How would you define \( a(u, v) \) and \( \bar{a}(u, v) \) so that:

\[
\bar{a}(u, v).P \quad | \quad a(x, y).Q \quad \xrightarrow{\tau}^* \quad P \mid Q\{u, v/x, y\}
\]

send two values over channel \( a \) to the same receiving process

receive two values over channel \( a \)

to the same receiving process
Sending multiple values over a channel

The $\pi$ calculus syntax is *monadic*, i.e., it only allows for one value to be communicated over a channel. A useful extension to the syntax to enable multiple values to be communicated when writing to or reading from a channel is the following:

\[
\alpha ::= \ c\langle x_1, \ldots, x_n \rangle \quad \text{Write } x_1, \ldots, x_n \text{ on channel } c
\]
\[
c(x_1, \ldots, x_n) \quad \text{Read } x_1, \ldots, x_n \text{ on channel } c
\]

The intent is that two processes communicate as follows:

\[
\bar{c}\langle x_1, \ldots, x_n \rangle . P \mid c(y_1, \ldots, y_n) . Q \xrightarrow{\tau}^* P \mid Q\{x_1, \ldots, x_n/y_1, \ldots, y_n\}
\]
Sending multiple values over a channel

Notice that we cannot simply use the following translation into the monadic calculus:

\[
\begin{align*}
\bar{c}\langle x_1, \ldots, x_n \rangle.P & \triangleq \bar{c}x_1.\bar{c}x_2.\cdots.\bar{c}x_n.P \\
{c}(y_1, \ldots, y_n).Q & \triangleq {c}(y_1)\cdot{c}(y_2)\cdot\cdots\cdot{c}(y_n).Q
\end{align*}
\]

Assuming the above translation, reduce the following expression:

\[
\bar{a}\langle u, v \rangle.P \mid a(x, y).Q \mid a(x, y).R
\]
Sending multiple values over a channel

\[
\bar{a}\langle u, v \rangle . P \mid a(x, y).Q \mid a(x, y).R
\]

\[
\equiv \bar{a}u.\bar{a}v . P \mid a(x).a(y).Q \mid a(x).a(y).R
\]

\[
\quad\quad\xrightarrow{\tau}\quad\bar{a}v . P \mid a(y).Q\{u/x\} \mid a(x).a(y).R
\]

\[
\quad\quad\xrightarrow{\tau}\quad P \mid a(y).Q\{u/x\} \mid a(y).R\{v/x\}
\]

- Notice that we underline the prefixes of the communicating processes for clarity.
- Did we get the expected behavior?
Polyadic \( \pi \) calculus

Instead, we must create a private communication link that is subsequently used to communicate all values over:

\[
\bar{c}\langle x_1, \ldots, x_n \rangle.P \triangleq (\nu d)\bar{c}d.\bar{d}x_1.\bar{d}x_2.\cdots.\bar{d}x_n.P \quad (1)
\]

The channel variable name \( d \) must be fresh, i.e., \( d \notin \text{fn}(P) \cup \{c, x_1, \ldots, x_n\} \), to ensure that we do not capture free variable occurrences thereby erroneously changing the meaning of process \( \bar{c}\langle x_1, \ldots, x_n \rangle.P \).

We can then decode the polyadic \( \pi \) calculus reading process expression \( c(y_1, \ldots, y_n).Q \) into:

\[
c(y_1, \ldots, y_n).Q \triangleq c(d).d(x_1).d(x_2).\cdots.d(x_n).Q \quad (2)
\]

where \( d \) is fresh.
Polyadic $\pi$ calculus example

Using polyadic to monoadic $\pi$ calculus translations (1) and (2):

$$\bar{a}\langle u, v \rangle . P \mid a(x, y). Q \mid a(x, y). R$$

$$\equiv (\nu w)(\bar{a}w.\bar{u}u.\bar{w}v.P) \mid a(z).z(x).z(y). Q \mid a(z).z(x).z(y). R$$

$$\equiv (\nu w)(\bar{a}w.\bar{u}u.\bar{w}v.P \mid a(z).z(x).z(y). Q) \mid a(z).z(x).z(y). R$$

$$\xrightarrow{\tau} (\nu w)(\bar{w}u.\bar{w}v.P \mid w(x).w(y).Q) \mid a(z).z(x).z(y). R$$

$$\xrightarrow{\tau} (\nu w)(\bar{w}v.P \mid w(y).Q\{u/x\}) \mid a(z).z(x).z(y). R$$

$$\xrightarrow{\tau} (\nu w)(P \mid Q\{u, v/x, y\}) \mid a(z).z(x).z(y). R$$

$$\equiv P \mid Q\{u, v/x, y\} \mid a(x, y). R$$
Exercise 3.6.11

1. Reduce the following expression:

\[
\text{Ref}(r, w, i) \mid (\nu c)\overline{w}\langle c, v \rangle.c.(\nu d)\overline{r}d.d(e).P
\]

2. How can you change the process expression in Exercise (1) to ensure that no other interacting processes may interfere with process 
   \(P\) receiving the value \(v\) over channel \(d\) in variable \(e\) from the reference cell?
Operational Semantics

The semantics of a concurrency model explains precisely the behavior of programs using a language following the model.

• An operational semantics of the $\pi$ calculus can be expressed in terms of a *labelled transition system*, where process expressions can evolve, modeling computation over time, according to a set of well defined structural inference rules.

• When the system evolves from a state $P$, into a state $Q$, according to the label $\alpha$, we write:

\[ P \xrightarrow{\alpha} Q \]
Operational Semantics

• In the \( \pi \) calculus, the label \( \alpha \) denotes interaction with the environment surrounding the process execution.
  • For example
    \[
    \bar{a}x.P \xrightarrow{\bar{a}x} P
    \]
    denotes that the \( \pi \) calculus expression \( \bar{a}x.P \) can evolve as \( P \) if the environment allows for the value \( x \) to be sent over channel \( a \).
  
• More generally, we can use a prefix \( \alpha \) to range over possible actions that a \( \pi \) calculus process may evolve with, and write:
  \[
  \alpha.P \xrightarrow{\alpha} P
  \]
Operational Semantics

- Notice that the input prefix can make a process evolve in infinitely many possible ways according to its environment. For example:
  
  \[
  a(b).\overline{bc}.0 \xrightarrow{a(u)} \overline{uc}.0 \xrightarrow{\overline{uc}} 0
  \]

  \[
  a(b).\overline{bc}.0 \xrightarrow{a(v)} \overline{vc}.0 \xrightarrow{\overline{vc}} 0
  \]
\( \pi \) calculus Operational Semantics

**PREFIX**

\[ \alpha . P \xrightarrow{\alpha} P \]

**STRUCT**

\[ P' \equiv P \quad P \xrightarrow{\alpha} Q \quad Q \equiv Q' \]

\[ P' \xrightarrow{\alpha} Q' \]

**SUM**

\[ P \xrightarrow{\alpha} P' \]

\[ P + Q \xrightarrow{\alpha} P' \]

**MATCH**

\[ \text{if } x = x \text{ then } P \xrightarrow{\alpha} P' \]

**MISMATCH**

\[ \text{if } x \neq y \text{ then } P \xrightarrow{\alpha} P' \]
\(\pi\) calculus Operational Semantics

**PAR**
\[
P \xrightarrow{\alpha} P' \quad \text{bn}(\alpha) \cap \text{fn}(Q) = \emptyset \\
P \mid Q \xrightarrow{\alpha} P' \mid Q
\]

**COM**
\[
P \xrightarrow{a(x)} P' \quad Q \xrightarrow{\bar{a}u} Q' \\
P \mid Q \xrightarrow{\tau} P'\{u/x\} \mid Q'
\]

**RES**
\[
P \xrightarrow{\alpha} P' \quad x \notin \alpha \\
(\nu x)P \xrightarrow{\alpha} (\nu x)P'
\]
The semantic rules are composed of two parts:

- the *preconditions* under which each transition can occur, which are written above the horizontal line, and
- the *consequent* of the rule that specifies how a process expression can evolve into another one, which is written below the horizontal line.
**π calculus Operational Semantics Example**

The transition above, \( a(b).\overline{bc}.0 \xrightarrow{a(u)} \overline{uc}.0 \), is a direct consequence of applying the PREFIX and STRUCT rules as follows:

\[
a(b).\overline{bc}.0 \equiv a(u).\overline{uc}.0 \quad \text{PREFIX} \quad \frac{a(u).\overline{uc}.0 \xrightarrow{a(u)} \overline{uc}.0}{a(b).\overline{bc}.0 \rightarrow \overline{uc}.0} \quad \text{STRUCT}
\]
\[ \pi \text{ calculus Operational Semantics (PREFIX and STRUCT)} \]

- The **PREFIX** rule has no pre-conditions and specifies that any \( \alpha . P \) expression can evolve as \( P \) using a transition with label \( \alpha \).
- The **STRUCT** rule formalizes the intuition that structural congruence is a syntactic form of equivalence of processes. The rule specifies that if \( P \xrightarrow{\alpha} Q \) then any expression \( P' \), structurally congruent to \( P \), can also evolve into any expression \( Q' \), structurally congruent to \( Q \). For example, one may derive the following:

\[
\begin{align*}
\bar{a}x.P \mid 0 & \equiv \bar{a}x.P \\
\bar{a}x.P \xrightarrow{\bar{a}x} P & \quad \text{PREFIX} \quad P \equiv P \mid 0 \\
\bar{a}x.P \mid 0 & \xrightarrow{\bar{a}x} P \mid 0 \quad \text{STRUCT}
\end{align*}
\]
\[ \pi \text{ calculus Operational Semantics (STRUCT)} \]

More interestingly, the STRUCT rule helps us avoid needing dual rules for many \( \pi \) calculus primitives. For example, it is not necessary to have a dual \( \text{SUM} \) rule:

\[ \text{SUM}_2 \quad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'} \]

since it can be inferred from STRUCT and SUM:

\[
\begin{align*}
P + Q &\equiv Q + P \\
&\xrightarrow{Q \xrightarrow{\alpha} Q'} Q + P \xrightarrow{\alpha} Q' \\
&\xrightarrow{Q' \equiv Q'} P + Q \xrightarrow{\alpha} Q'
\end{align*}
\]

If \( P \xrightarrow{\alpha} P' \) and \( Q \xrightarrow{\alpha'} Q' \), then \( P + Q \) can evolve in two possible ways: \( P + Q \xrightarrow{\alpha} P' \) (by SUM rule), or \( P + Q \xrightarrow{\alpha'} Q' \) (by the inferred \( \text{SUM}_2 \) rule). This is exactly the intent of the \( + \) operator in the \( \pi \) calculus: non-deterministic choice between \( P \) and \( Q \).
The \texttt{PAR} rule specifies independent progress for a process $P$ interacting with the environment when it is part of a concurrent process composition expression $P \mid Q$, and the \texttt{COM} rule specifies how both processes $P$ and $Q$ can make progress synchronously as a result of a communication between them over a shared channel $a$. The precondition $\text{bn}(\alpha) \cap \text{fn}(Q) = \emptyset$ of the \texttt{PAR} rule is necessary to prevent erroneously capturing free variable occurrences in the process $Q$ as a result of applying both the rules \texttt{PAR} and \texttt{COM} as follows:

\[
\begin{align*}
a(x).P \xrightarrow{a(x)} P & \quad \text{PAR} \quad (a(x).P) \mid Q \xrightarrow{a(x)} P \mid Q \\
\left( (a(x).P) \mid Q \right) \mid \overline{\alpha u}.R \xrightarrow{\tau} (P \mid Q)\{u/x\} \mid R & \quad \text{COM}
\end{align*}
\]

Therefore, ensuring that $x$ (bound in $a(x)$) does not appear free in $Q$ is critical to the correctness of the substitution $(P \mid Q)\{u/x\}$. 

\[\pi\textit{ calculus Operational Semantics (PAR and COM)}\]
\[ \pi \text{ calculus Operational Semantics (RES)} \]

The RES rule specifies the behavior of processes that create a new channel \( x \), namely \((\nu x)P\).

The RES rule allows *internal* progress within \( P \), as long as the communication with the environment does not have anything to do with the restricted channel \( x \).

If the label \( \alpha \) is \( \tau \), then there is no interaction with the environment, and RES applies trivially:

\[
\begin{align*}
(\nu x)(a(b).P \mid \bar{a}c.Q) & \xrightarrow{\tau} (\nu x)(P\{c/b\} \mid Q) \\
\end{align*}
\]

Notice that any of \( a, b, \) and \( c \) could have been \( x \); and the RES rule would still apply.
The two less clear cases are when $\alpha$ is $a(b)$ or $\bar{a}b$, namely, how should processes $(\nu x)a(b).P$ and $(\nu x)\bar{a}b.P$ evolve?

According to the RES rule, the only way that these processes can evolve is if $x \neq a$ and $x \neq b$. When $a = x$, we have the expressions $(\nu x)x(b).P$ and $(\nu x)\bar{x}b.P$, which should be semantically equivalent to 0, since we are trying to communicate over a channel name that is private. Therefore, by construction no rule should apply. When $b = x$, we have the expressions $(\nu x)a(x).P$ and $(\nu x)\bar{a}x.P$. We do not want to allow:

$$(\nu x)a(x).P \xrightarrow{a(x)} (\nu x)P$$

since the input $a(x)$ from the environment allows for any arbitrary name to be input over channel $a$. However, if $x$ itself is sent over channel $a$, any free occurrences of $x$ in $P$ would become erroneously captured by the $(\nu x)$ using this transition.
Finally, we have the expression \((\nu x)\overline{a}x.P\). In this case, we want the semantics to capture the intuitive notion that the expression

\[ a(b).P \mid (\nu x)\overline{a}x.Q \]

can evolve as

\[ (\nu x)(P\{x/b\} \mid Q) \]

assuming that \(x \notin \text{fn}(P) \cup \{a\}\).
\[ \pi \text{ calculus Operational Semantics (RES)} \]

The PREFIX, COM, RES and STRUCT rules could be used to make this derivation as follows:

\[
\begin{align*}
x \notin \text{fn}(P) \cup \{a\} \\
\frac{}{a(b).P \mid (\nu x)\bar{a}x.Q \equiv (\nu x)(a(b).P \mid \bar{a}x.Q)} \quad (4)
\end{align*}
\]

\[
\begin{align*}
&\quad \quad (3) \\
\frac{(\nu x)(a(b).P \mid \bar{a}x.Q) \xrightarrow{\tau} (\nu x)(P\{x/b\} \mid Q)}{a(b).P \mid (\nu x)\bar{a}x.Q \xrightarrow{\tau} (\nu x)(P\{x/b\} \mid Q)} \quad \text{STRUCT (5)}
\end{align*}
\]