The Polyadic π-calculus

- Sending/receiving more names in a communication
- Syntax:

$$P \quad ::= \quad \mathbf{0} \mid a(\widetilde{x}).P \mid \overline{a}\langle\widetilde{b}\rangle.P \mid (\nu a)P \mid P_1|P_2 \mid !P \mid \ [a=b]P$$

  where "tilde" denotes a (possibly empty) sequence of names
  - if every sequence has length 1, we have the original π-calc
  - if every sequence has length 0, we have CCS
- The reduction rule then becomes

$$(\text{R-Com}) \quad a(\widetilde{x}).P \mid \overline{a}\langle\widetilde{b}\rangle.Q \longmapsto P[\widetilde{b}/\widetilde{x}] \mid Q$$

  where $[\widetilde{b}/\widetilde{x}]$ denotes the component-wise replacement of names b for names x
- PROBLEM: what if $\widetilde{x}$ and $\widetilde{b}$ have different lengths?

- Consider the analogy
  - input ⟷ function definition
  - output ⟷ function invocation
- In any programming language, invoking a function with the wrong number (and type) of parameters is a (type) error, usually caught by the compiler at compile time
- We want to develop a type system that can catch these kinds of errors. Formally:

$$\text{(E-Com)}\ \frac{}{a(\widetilde{x}).P \mid \overline{a}\langle\widetilde{b}\rangle.Q \uparrow}\ {\scriptstyle |\widetilde{x}|\neq|\widetilde{b}|} \qquad \text{(E-Par)}\ \frac{P \uparrow}{P \mid Q \uparrow}$$

$$\text{(E-Res)}\ \frac{P \uparrow}{(\nu a)P \uparrow} \qquad \text{(E-Struct)}\ \frac{P \uparrow \quad P \equiv Q}{Q \uparrow}$$

- Intuitively, P ↑ should be read as: “In P there is a communication attempt on the same channel with an arity mismatch among the input and the output”.

A first attempt to statically catch run-time errors

- The natural idea is to associate every channel with a type that describes the legal communications along the channel
- Hence, a typing environment Γ is a (partial) mapping from the set of names *N* to the set of types *T*
- Let's choose $T = \mathbb{N}$

  → Γ(a) states the number of names that must be passed in every communication along a
- Some of the typing rules should be:

$$\frac{\Gamma(a) = |\widetilde{b}| \qquad \Gamma \vdash P}{\Gamma \vdash \overline{a}\langle\widetilde{b}\rangle.P} \qquad \frac{\Gamma(a) = |\widetilde{x}| \qquad \Gamma \vdash P}{\Gamma \vdash a(\widetilde{x}).P}$$

$$\Gamma \vdash \mathbf{0} \qquad \frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \mid Q}$$

- Can $a\langle b, c\rangle.P \mid a(x).Q$ be typed?
  - This amounts to finding a typing environment Γ such that we can infer $\Gamma \vdash \overline{a}\langle b, c\rangle.P \mid a(x).Q$
  - To type $a\langle b,c\rangle.P$, it must be that $\Gamma(a) = 2$
  - To type $a(x).Q$, we would need $\Gamma(a) = 1$
  - Impossible, since Γ is a function
- Hence, the given process is not typeable and rejected by the type system, as desired

  → indeed, $a\langle b, c\rangle.P \mid a(x).Q$ ↑

- Can we type $a(x).x(z) \mid a\langle c\rangle.c\langle b, b\rangle$?
- Consider Γ to be such that $\Gamma(a) = 1$ $\Gamma(x) = 1$ $\Gamma(c) = 2$
- Then,

$$\dfrac{\dfrac{\Gamma(a) = 1 = |x| \qquad \dfrac{\Gamma(x) = 1 = |z| \quad \Gamma \vdash \mathbf{0}}{\Gamma \vdash x(z).\mathbf{0}}}{\Gamma \vdash a(x).x(z)} \qquad \dfrac{\Gamma(a) = 1 = |c| \qquad \dfrac{\Gamma(c) = 2 = |b, b| \quad \Gamma \vdash \mathbf{0}}{\Gamma \vdash \bar{c}\langle b, b\rangle.\mathbf{0}}}{\Gamma \vdash \bar{a}\langle c\rangle.\bar{c}\langle b, b\rangle}}{\Gamma \vdash a(x).x(z) \mid \bar{a}\langle c\rangle.\bar{c}\langle b, b\rangle}$$

- PROBLEM: after the first communication we obtain

  $a(x).x(z) \mid a\langle c\rangle.c\langle b, b\rangle \rightarrow c(z) \mid c\langle b, b\rangle$

  that is clearly *not* typeable (as shown before)
- Hence, a well-typed process has reduced to an ill-typed one

  → this is not desirable from a type system!

The Type System for Correct Communications

- We cannot only keep track of the number of the arguments of a communication (as the choice of $T = \mathbb{N}$ entails)
- Also the type of every exchanged name matters!

  → This is similar to what happens in any language, where not only the number of the arguments of a function, but also their types matter
- We need more informative types, that we now recursively define:

  $$T ::= [T1,\ldots,Tk] \quad \text{with } k \geq 0$$
- The base case of the recursion is when k = 0 (assigned to a channel where no data is exchange)
- A typing environment Γ will be represented as a sequence of name-to-type associations n1 : T1, . . . , nk : Tk such that ni ≠ nj whenever i ≠ j.
- dom(Γ) = {n1,...,nk}

Typing Rules

$$\text{(T-Nil)}\quad \frac{}{\Gamma \vdash \mathbf{0}} \qquad \text{(T-Par)}\quad \frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad \text{(T-Rep)}\quad \frac{\Gamma \vdash P}{\Gamma \vdash !P}$$

$$\text{(T-Match)}\quad \frac{\Gamma \vdash P}{\Gamma \vdash [a = b]P} \qquad \text{(T-Res)}\quad \frac{\exists T.\Gamma, a : T \vdash P}{\Gamma \vdash (\nu a)P}\ a \notin dom(\Gamma)$$

$$\text{(T-Out)}\quad \frac{\Gamma(a) = [T_1, \ldots, T_k] \qquad \forall i = 1, \ldots, k.\Gamma(b_i) = T_i \qquad \Gamma \vdash P}{\Gamma \vdash \overline{a}\langle b_1, \ldots, b_k \rangle.P}$$

$$\text{(T-In)}\quad \frac{\Gamma(a) = [T_1, \ldots, T_k] \quad \Gamma, x_1 : T_1, \ldots, x_k : T_k \vdash P}{\Gamma \vdash a(x_1, \ldots, x_k).P}\ \{x_1, \ldots, x_k\} \cap dom(\Gamma) = \emptyset$$

Checking $\Gamma \vdash a(x).x(z).\mathbf{0} \mid a\langle c\rangle.c\langle b, b\rangle.\mathbf{0}$

$$\dfrac{\Gamma(a) = [T] \qquad \dfrac{(\Gamma, x : T)(x) = [T_1] \qquad \Gamma, x : T, z : T_1 \vdash \mathbf{0}}{\Gamma, x : T \vdash x(z).\mathbf{0}}}{\Gamma \vdash a(x).x(z).\mathbf{0}}$$

$$\dfrac{\Gamma(a) = [T] \qquad \Gamma(c) = T \qquad \dfrac{\Gamma(c) = [T', T'] \qquad \Gamma(b) = T' \qquad \Gamma \vdash \mathbf{0}}{\Gamma \vdash \bar{c}\langle b, b\rangle.\mathbf{0}}}{\Gamma \vdash \bar{a}\langle c\rangle.\bar{c}\langle b, b\rangle.\mathbf{0}}$$

The first inference would imply $T = [T_1]$,
whereas the second one would imply $T = [T', T']$

Hence, we cannot find a Γ that allows us to build a derivation for $\Gamma \vdash a(x).x(z).\mathbf{0} \mid \bar{a}\langle c\rangle.\bar{c}\langle b, b\rangle.\mathbf{0}$.

Soundness of the Type System

Theorem 9 (Subject Reduction) *If* $\Gamma \vdash P$ *and* $P \longmapsto P'$, *then* $\Gamma \vdash P'$.

Theorem 10 (Type Safety) *If* $\exists \Gamma . \Gamma \vdash P$, *then* $P \not\downarrow$.

we say that a process P is *typeable* if there exists Γ such that $\Gamma \vdash P$.

Corollary 11 *If* P *is typeable, then* $P' \not\downarrow$, *for every* P' *such that* $P \Longmapsto P'$.

Encoding Poly-π into Monadic-π

- First attempt:
  - translate the 2-ary process $a\langle b, c\rangle.P \mid a(x, y).Q$ as

    $a\langle b\rangle.a\langle c\rangle.P \mid a(x).a(y).Q$
  - The problem here is if we consider process $a\langle b,c\rangle.P \mid a(x,y).Q \mid a(x,y).R$ (whose encoding would then be $a\langle b\rangle.a\langle c\rangle.P \mid a(x).a(y).Q \mid a(x).a(y).R$)
  - we could have that name b is caught by Q and name c is caught by R

    → in the original process both names can be caught by only one between Q and R!
- Second attempt:
  - translate the 2-ary process $a\langle b, c\rangle.P \mid a(x, y).Q$ as

    $a\langle b\rangle.b\langle c\rangle.P \mid a(x).x(y).Q$
  - this solves the problem with process $a\langle b, c\rangle.P \mid a(x, y).Q \mid a(x, y).R$
  - New problem: $a\langle b, c\rangle.P \mid a(x, y).Q \mid b(z).R$

    → again, it is possible that b is caught by Q and c is caught by R
- SOLUTION: create a new channel, send it over a and then use it to deliver the arguments of the polyadic communication

  → Turn $a\langle b, c\rangle.P \mid a(x, y).Q$ into $(vd)a\langle d\rangle.d\langle b\rangle.d\langle c\rangle.P \mid a(z).z(x).z(y).Q$ where d and z are fresh names.

The encoding

$$
\begin{aligned}
(\!|\mathbf{0}|\!) &\triangleq \mathbf{0} \\
(\!|(\nu a)P|\!) &\triangleq (\nu a)(\!|P|\!) \\
(\!|P \mid Q|\!) &\triangleq (\!|P|\!) \mid (\!|Q|\!) \\
(\!|[a=b]P|\!) &\triangleq [a=b](\!|P|\!) \\
(\!|!P|\!) &\triangleq\ !\,(\!|P|\!)
\end{aligned}
$$

$$
\begin{aligned}
(\!|\overline{a}\langle a_1,...,a_n\rangle.P|\!) &\triangleq (\nu d)\overline{a}\langle d\rangle.\overline{d}\langle a_1\rangle.\cdots.\overline{d}\langle a_n\rangle.(\!|P|\!) \quad \text{for } d \notin fn(a,a_1,..,a_n,P) \\
(\!|a(x_1,...,x_n).P|\!) &\triangleq a(z).z(x_1).\cdots.z(x_n).(\!|P|\!) \quad \text{for } z \notin fn(a,x_1,..,,x_n,P)
\end{aligned}
$$

On the Properties of an Encoding

- *Computational cost of performing an action:* a single polyadic communication yields n+1 monadic communications
  → this is very close to the best we can imagine, since every n-ary communication requires at least n monadic communications
- The most important thing we have to check is that the source process and its encoding ‘‘behave in the same way”
- Operational correspondence:
  - *If P → P', then* $\langle [P] \rangle \Rightarrow \langle [P'] \rangle$
    if a polyadic process P reduces to some P', the corresponding monadic process reduces (in some steps) to the encoding of P'
    → the behavior of the source process is preserved by its translation
  - *if* $\langle [P] \rangle \rightarrow Q$, *then there exists P' s.t.* $P \Rightarrow P'$ *and* $Q \Rightarrow \langle [P'] \rangle$
    if the encoding of P reduces to some process Q (that, in general, is not an encoding), then Q must always converge to the encoding some reduct of an encoding eventually reduces to the encoding of some polyadic process.
    → the translation does not add any new behavior
- The encoding enjoys many other good properties
  → polyadic and monadic π-calculus have the same expressive power
- REMARK: the encoding just shown satisfies all these good properties only we restrict to well-typed polyadic processes
  → for non-typeable processes, no encoding exists that satisfies a ‘reasonable’ set of properties.

Asynchronous π-calculus

- Communication in π-calculus:
  - blocking input: the continuation process P is suspended until the input prefix synchronizes with some output
    → necessary, since P cannot proceed until it knows the real message
  - blocking output: process Q is suspended until its message is received by some other process
- Blocking outputs are less natural. Examples:
  - exchange of e-mails/text-messages
  - information put on a web-page
  - ...
- *Asynchronous communication*: the sender and the receiver do not synchronize to communicate
  → Sending and receiving a message happen in two different moments

On Modelling Asynchrony

- First way: replace rule (R-Com) with two rules:

$$\text{(R-Snd)} \quad \overline{a}\langle b\rangle.P \longmapsto \overline{a}\langle b\rangle \mid P \qquad \text{(R-Rcv)} \quad a(x).P \mid \overline{a}\langle b\rangle \longmapsto P[{}^{b}\!/_{x}]$$

  - This is the solution that better models reality
  - However, it is not used in practice for a 'philosophical' reason: we would have two basic axioms that model the single basic computation step of the π-calculus
- Second way: modify the syntax to exclude output prefixes:

$$P ::= \mathbf{0} \mid a(x).P \mid \overline{a}\langle b\rangle \mid P_1|P_2 \mid (\nu n)P \mid !P \mid [a=b]P$$

  - Output are not blocking by construction
  - Only one axiom for communication (in place of (R-Com):

$$a(x).P \mid \overline{a}\langle b\rangle \longmapsto P[{}^{b}\!/_{x}]$$

Enconding Synchrony into Asynchrony

How can we encode $a(x).P \mid a\langle b\rangle.Q$?

- First attempt:

  $$a(x).(c\langle d\rangle \mid P) \;\mid\; a\langle b\rangle \mid c(y).Q$$

  - Problems:
    - $a(x).(c\langle d\rangle \mid P) \mid a\langle b\rangle \mid c(y).Q \mid c(z).R$
      $\rightarrow\rightarrow P[b/x] \mid c(y).Q \mid R[d/z]$
    - $a(x).(c\langle d\rangle \mid P) \mid a\langle b\rangle \mid c(y).Q \mid c\langle d\rangle.R$
      $\rightarrow a(x).(c\langle d\rangle \mid P) \mid a\langle b\rangle \mid Q[d/y] \mid R$
  - So, the ack channel c must be restricted
- Second attempt:

  $$a(y).y(x).(y\langle d\rangle \mid P) \;\mid\; (\nu c)(a\langle c\rangle \mid c\langle b\rangle \mid c(x).Q)$$

  - Problem: Q can be unblocked by itself
- Third attempt:

  $$a(y).(y\langle d\rangle \mid y(x).P) \;\mid\; (\nu c)(a\langle c\rangle \mid c(x).(c\langle b\rangle \mid Q))$$

  - Problem: P can be consume the ack

The encoding

$$
\begin{array}{rcll}
[\![\mathbf{0}]\!] & \triangleq & \mathbf{0} & \\
[\![P_1|P_2]\!] & \triangleq & [\![P_1]\!] \mid [\![P_2]\!] & \\
[\![!P]\!] & \triangleq & ![\![P]\!] & \\
[\![(\nu n)P]\!] & \triangleq & (\nu n)[\![P]\!] & \\
[\![[a=b]P]\!] & \triangleq & [a=b][\![P]\!] & \\
[\![\overline{a}\langle b\rangle.P]\!] & \triangleq & (\nu c)(\overline{a}\langle c\rangle \mid c(y).(\overline{y}\langle b\rangle \mid [\![P]\!])) & \text{for } c, y \notin fn(P) \\
[\![a(x).P]\!] & \triangleq & a(z).(\nu d)(\overline{z}\langle d\rangle \mid d(x).[\![P]\!]) & \text{for } d, z \notin fn(P)
\end{array}
$$

- Also this encoding satisfies operational correspondence and many other good properties

  → Synchnony and Asynchrony have the same expressive power

Higher Order π-calculus

- a π-calculus where messages can be processes as well
- This very naturally models scenarios where executable code is sent during an interaction:
  - e-mail with attachments
  - java applets downloaded from a web page
  - remote code evaluation
- Let X be a generic process variable chosen from the set V, disjoint from N
- The syntax:

$$\begin{array}{lcl} P & ::= & \mathbf{0} \mid P_1|P_2 \mid (\nu n)P \mid [a=b]P \mid a(\theta).P \mid \overline{a}\langle\eta\rangle.P \mid X \\ \eta & ::= & b \mid P \\ \theta & ::= & x \mid X \end{array}$$

- Communication rule:

$$a(\theta).P \mid \overline{a}\langle\eta\rangle.Q \longmapsto P[{}^{\eta}/_{\theta}] \mid Q$$

- EXAMPLE: a(X).(X|X) | a⟨R⟩ → R|R
- REMARK: a type system is necessary to statically rule out erroneous terms like a(X).(X|X) | a⟨b⟩ or a(x).x(y) | a⟨P⟩

Encoding Replication

- *Duplicator:* $D_a = a(X).(X \mid a\langle X\rangle)$
- *Replicator:* $R(P) = (\nu a)(D_a \mid a\langle P|D_a\rangle)$ for a fresh
- $R(P) \rightarrow (\nu a)((P|D_a) \mid a\langle P|D_a\rangle)$

  $\equiv P \mid (\nu a)(D_a \mid a\langle P|D_a\rangle)$

  $= P \mid R(P)$

- The only difference between R(P) and !P is that every activation of a copy of P here requires a reduction, whereas, by using (S-Rep) and (R-Struct), in a single reduction many copies can be simultaneously activated

Encoding HOπ into π

- Given a higher-order process P , let us consider a partial function $\phi : V \rightarrow N$ such that $im(\phi) \cap n(P) = \varnothing$
- Encoding:

$$
\begin{array}{rcll}
[\![a(X).P]\!] & \triangleq & a(x).[\![P]\!] & \text{for } x = \varphi(X) \\
[\![\overline{a}\langle P\rangle.Q]\!] & \triangleq & (\nu n)\overline{a}\langle n\rangle.([\![Q]\!] \mid\, !n().[\![P]\!]) & \text{for } n \notin fn(P,Q) \cup im(\varphi) \\
[\![X]\!] & \triangleq & \overline{x}\langle\rangle & \text{for } x = \varphi(X)
\end{array}
$$

where n() and x⟨⟩ denote and input/output with a useless argument

EXAMPLE: a(X).(X | X) | a⟨P⟩ → P | P

By encoding it, we obtain:

a(x).(x⟨⟩ | x⟨⟩) | (νn)a⟨n⟩.(0 | !n().P)

→ (νn)(n⟨⟩ | n⟨⟩ | !n().P)
→ (νn)(n⟨⟩ | !n().P | P)
→ (νn)(!n().P | P | P)
≡ (νn)(!n().P) | P | P
~ P | P since (νn)(!n().P) ~ **0**

- As clear from the previous example, operational correspondence for this encoding holds up-to bisimilarity:
  - If $P \rightarrow P'$, then $[P] \Rightarrow\sim [P']$
  - If $[P] \rightarrow Q$, then there exists P' such that $P \Rightarrow P'$ and $Q \Rightarrow\sim [P']$
- Other good properties are satisfied, and so also first horder and higher order have the same expressive power