The Pi-Calculus

2 March 2021

0. What motivated the search for a process calculus?
1. What did prehistoric models look like?
2. How did the pi-calculus improve upon them?
3. What is The Truth for a process calculus?
4. How do you add discipline to this world?

Pre-History

In the late 70's / early 80's, researchers had a model of computation (λ-calculus), but no model for concurrent communication.

Researchers wanted to reason about processes running independently and in communication with each other.

Many 'process calculi' emerged:
- Actors
- CSP
- Petri Nets
- ...

Many models, little agreement.

1. Milner 1980 - CCS

Calculus of Communicating Systems (CCS).

Two Ideas:

1. A process can be defined based on how an observer interacts with it. To build complex systems, you can take an observer and model it as a process that can be observed.

2. Processes communicate over channels → pathways over which data can be sent between processes.

   ↳ NOT a new idea.

Intro to CCS

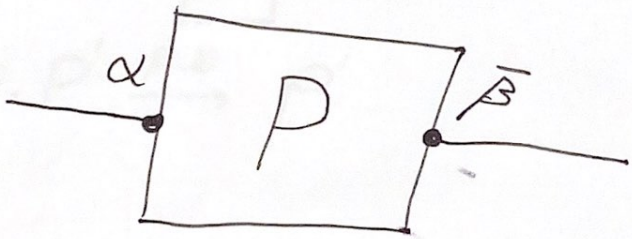
Preliminaries:

Channels are defined by a set of names, unique symbols that differ from other values. We use $\alpha, \beta, \gamma, \delta \ldots$ and range over channels with μ.

Denote processes with P, Q, R.

We define processes via their interactions with processes over channels.

Channels are a static property of a process.



Example: checkout at a grocery store.

1. nil
   - a closed checkout aisle.

2. receive
   - the cashier receives a payment from customer.

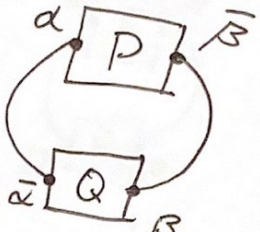
$$\alpha(x).P' \xrightarrow{\alpha 5} P'[x \leftarrow 5]$$

3. send
   - the cashier hands you your purchase.

$$\bar{\beta} 10.P' \xrightarrow{\bar{\beta} 10} P'$$

4. Parallel (Par)

- a cashier and customer communicating.



$$(\alpha(x).\bar{\beta}2.x.\ P' \mid \bar{\alpha}5.\beta(y).\ Q')$$

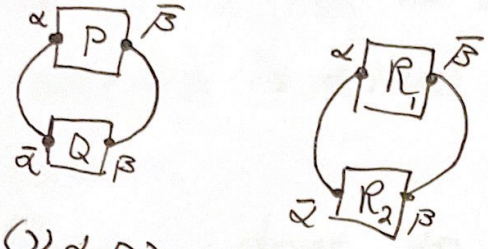
$$\xrightarrow{\alpha 5} (\bar{\beta}10.P'[x \leftarrow 5] \mid \beta(y).Q')$$

$$\xrightarrow{\beta 10} (P'[x \leftarrow 5] \mid Q'[y \leftarrow 10])$$

Note: From the perspective of the observer, they don't interact with the system. These are sometimes called τ-transitions ($\xrightarrow{\tau}$). This distinction is only useful for theoretical purposes.

5. Restriction

- cashiers & customers in different aisles.



$(\nu\,\alpha\beta)\,(\ldots) \mid (\nu\,\alpha\beta)\,(\ldots)$

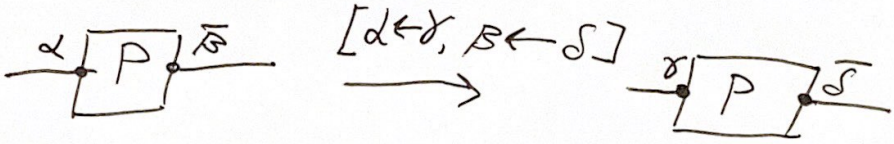
6. Summation

- a cashier might respond in multiple ways.

$\xrightarrow{\gamma 3} \alpha(x).P_1 + \gamma(y).P_2$

$P_2[y \leftarrow 3]$

7. Relabelling



$(\alpha(x).\bar{\beta}2{\cdot}x.P')[\alpha \leftarrow \gamma,\ \beta \leftarrow \delta]$

$\rightarrow \quad (\gamma(x).\bar{\delta}2{\cdot}x.P'[\ldots])$

Relabelling is a part of the syntax. Channels are static, and renaming & substitution isn't defined over them.

Problem: where did the Truth go?

In the λ-calculus, the Truth is Observational Equivalence. It's meaning is derived from expressions reducing to values.

Processes don't do so; what do we do?

Insight: two processes are equivalent if they simulate each other.

A process simulates another process if it has the same interface (channels) and, after a transition, continues to simulate the other process.

This is called a bisimulation / bisimilarity relation.

Reflections on CCS

Milner was unhappy for two reasons:

1. Channels are static. In the real world, communication links change dynamically with time.

2. Could a smaller calculus be developed?

Intermediate Step - Ufeberg & Nielsen, 1986

Solution: allow processes to pass channels over channels.

Extended CCS (or ECCS).

Two disjoint sets of variables:

- value variables: $x, y, z \ldots$
- name / channel variables: a, b, c

Example: Walkie-Talkies

Two processes would like to communicate over a private talk channel.

$$\bar{\alpha}\beta.\beta(x).P' \mid \alpha(talk).\overline{talk}\,5.Q'$$

$$\xrightarrow{\alpha\beta} \quad \beta(x).P' \mid \bar{\beta}\,5.Q'[talk \leftarrow \beta]$$

$$\xrightarrow{\beta 5} \quad P'[x \leftarrow 5] \mid Q'[talk \leftarrow \beta]$$

ECCS Reflection

The capacity for the links between processes to change dynamically is called 'mobility.'
↳ channels no longer unidirectional.

This also meant shrinking the calculus. We no longer need relabelling, since we've defined substitution and renaming over channel variables.

2. Milner, Parrow, Walker 1992 - π-calculus

Small improvement: distinguishing between channel and value variables is unnecessary to achieve the same expressiveness.

The relatively small change of removing the distinction is what produces the π-calculus.

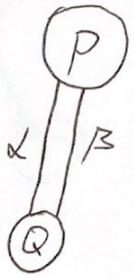
Characteristic Examples

1. Passing Channels

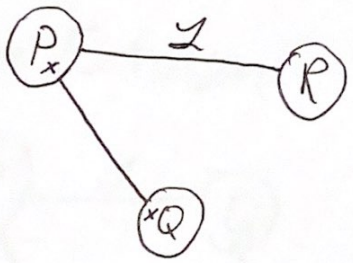
$$\bar{\alpha}\beta.\beta(x).P' \mid \alpha(chan).\overline{chan}\,5.Q'$$

$$\xrightarrow{\alpha\beta} \beta(x).P' \mid \bar{\beta}5.Q'[chan \leftarrow \beta]$$

$$\xrightarrow{\beta 5} P'[x \leftarrow 5] \mid Q'[chan \leftarrow \beta]$$



2. Restricting Channels



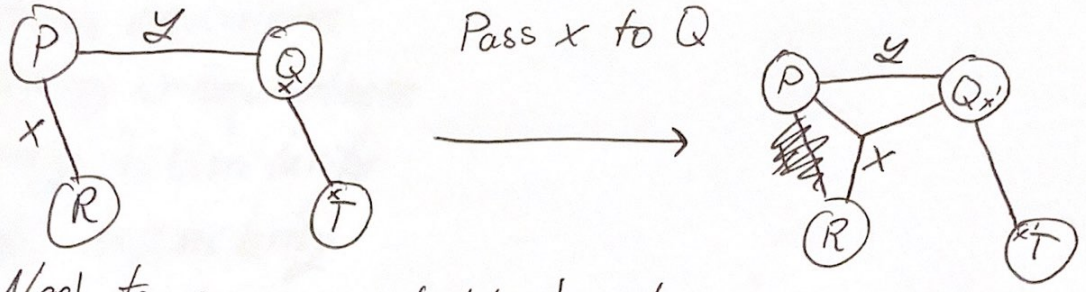
$$(\nu x)(\bar{x}5.P' \mid x(y).Q') \mid R$$

$$\xrightarrow{x5} (\nu x)(P' \mid Q'[y \leftarrow 5]) \mid R$$

A well-defined process (like a λ-calculus program) has all channels explicitly introduced by restriction.

Characteristic Examples 2: Breaking the Scope Barrier

3. Scope Intrusion

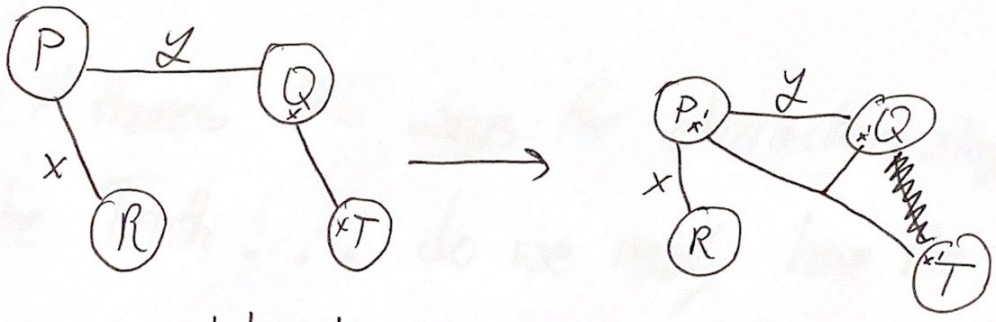


Need to rename restricted channels.

$$\overline{y}x.P' \mid R \mid (\nu x)(y(z).Q' \mid T)$$

$$\xrightarrow{yx} P' \mid R \mid (\nu x)(Q'[x \leftarrow x'][z \leftarrow x] \mid T[x \leftarrow x'])$$

4. Scope Extrusion



$$y(z).P' \mid R \mid (\nu x)(\overline{y}x.Q' \mid T)$$

$$\xrightarrow{yx'} P'[z \leftarrow x'] \mid R \mid (\nu x')(Q'[x \leftarrow x'] \mid T[x \leftarrow x'])$$

Final Note on The Truth:

Milner et al. introduce five bisimilarity relations
- strong bisimilarity
- strong equivalence
- strong D-Equivalence
- early bisimilarity
- late bisimilarity

And there's more where that came from:
- barbed congruence
- strong barbed congruence

. . .

If there's 25 ways for characterizing the Truth, . . . do we really have the Truth?

3. Milner 1993 - Adding Discipline

The π-calculus is powerful, but chaotic.

Example: the 'molecular action.'

$u, v \in$ values

$x \in$ channels

$$\bar{x}u.\bar{x}v.P' \mid x(y).x(z).Q' \mid x(y).x(z).R'$$

Values are NOT broadcasted over channels.

$$\xrightarrow{xu} \bar{x}v.P' \mid x(z).Q'[y \leftarrow u] \mid x(y).x(z).R'$$

$$\xrightarrow{xv} P' \mid x(z).Q'[y \leftarrow u] \mid x(z).R'[y \leftarrow v]$$

The values can get split between processes Q and R.

How can we ensure they're received by the same process?

Solution: send a channel first.

$x, w \in$ channels.

$$\bar{x}w.\bar{w}u.\bar{w}v.P' \mid x(w).w(u).w(v).Q' \mid x(w).w(u).w(v).R'$$

$\xrightarrow{xw}$ $$\bar{w}u.\bar{w}v.P' \mid w(u).w(v).Q' \mid x(w).w(u).w(v).R'$$

$\xrightarrow{wu}$ $$\bar{w}v.P' \mid w(v).Q' \mid x(w).w(u).w(v).R'$$

$\xrightarrow{wv}$ $$P' \mid Q' \mid x(w).w(u).w(v).R'$$

Expressive & powerful, but intricate.

Intermediate Step: Polyadic π-calculus

Instead of sending single values, send and receive tuples of arbitrary arity.

$$P = \bar{x}\langle u, v \rangle . P'$$

$$Q = x(u, v) . Q'$$

$$R = x(u, v) . R'$$

- Good: can pass multiple values at a time.
- Bad: Arity errors.

How do we add discipline?

A Sorting Discipline

sorts are analogous to types in λ-calculus.

Each channel is associated with a sort. A sort is mapped to an 'object sort,' or a list of sorts.

Intuition: we express the arities of each channel, and the arities of the values passed over it.

$$S = \{\text{SENDER}: [\text{LEFT}, \text{RIGHT}],\ \text{LEFT}: [],\ \text{RIGHT}: []\}$$

$$(\nu\, x: \text{SENDER},\ u: \text{LEFT},\ v: \text{RIGHT})\ (\ldots)$$

Note: we need names because our types are recursive.

Inference Rules:

We show that processes are well-typed. A process must be well-defined.

A closed process has sort $[\,]$.

A process with a free channel has the sort of that channel.

$$\frac{\Gamma \vdash x : S \quad \Gamma \vdash P : ob(S)}{\Gamma \vdash x.P : [\,]} \quad [T\text{-}IN]$$

$$\frac{\Gamma \vdash x : S \quad \Gamma \vdash P : ob(S)}{\Gamma \vdash \bar{x}.P : [\,]} \quad [T\text{-}OUT]$$

$$\frac{\Gamma \vdash P : [\,] \quad \Gamma \vdash Q : [\,]}{\Gamma \vdash (P \mid Q) : [\,]} \quad [T\text{-}PAR]$$

$$\frac{\Gamma, x : S \vdash P : [\,]}{\Gamma \vdash (\nu\, x : S)\, P : [\,]} \quad [T\text{-}RESTR]$$

Reflection

The sorting discipline guarantees no arity runtime errors.

Is this what we want?

This discipline is unsatisfactory. We haven't learned how to use channels.

Example: what if type systems for modern programming languages only let the programmer specify the arities of functions?

Arity-checking is insufficient.

Pierce & Sangiorgi 1994 - I/O Types

Motivation: the office printer.

job, paper $\in$ channels

$$Printer = job(x).\overline{paper}\langle x\rangle.Printer$$

If somebody else is reading on the job channel, they can steal the data somebody else wanted to print.

Solution: specify what direction the channel can be used for a process.

Preliminaries:

Polarities = $\{i, o\}$

$r = \{i\}$

$w = \{o\}$

$b = \{i, o\}$

e.g. $r[T_1, T_2, \ldots T_n]$

↳ a sort for a channel of arity n that can only be read.

Printer example:

$(\nu\, job: r[S], paper: w[S])\,(job(x).\overline{paper}\,x.\,Printer)$

A client Q will receive the job channel & specific interface:

$Q = recv(job: w[S]).\ \overline{job}\ 5.\ Q'$

The Sub-typing relation:

We need a way to show that a sort obeys the interface of a channel

$b <: r$

$b <: w$

$<:$ A sort S is a subsort of T iff:

1. iff $T = b[T_1 \ldots T_n]$ then $S = b[S_1 \ldots S_n]$ and <u>both</u> $S_i <: T_i$ <u>and</u> $T_i <: S_i$.

2. iff $T = r[T_1 \ldots T_n]$ then $S = I[S_1 \ldots S_n]$, $I <: r$, and for all $1 \leq i \leq n$, $S_i <: T_i$.

3. iff $T = w[T_1 \ldots T_n]$ then $S = I[S_1 \ldots S_n]$, $I <: w$, and for all $1 \leq i \leq n$, $T_i <: S_i$.

Example Rule: (Σ is the subsort context)

$$\frac{I <: r \qquad \text{for each } i, \Sigma \vdash S_i <: T_i}{\Sigma \vdash I[S_1 \ldots S_n] <: r[T_1 \ldots T_n]}$$

Typing processes with I/O Types:

Two modified rules:

$$\frac{\vdash \Gamma(a) <: \Gamma[S_1 \ldots S_n] \quad \Gamma, b_1 : S_1 \ldots b_n : S_n \vdash P : ()}{\Gamma \vdash a(b_1 : S_1 \ldots b_n : S_n).P : ()} \; [\Gamma\text{-IN}]$$

$$\frac{\vdash \Gamma(a) <: \mathrm{W}[\Gamma(b_1) \ldots \Gamma(b_n)] \quad \Gamma \vdash P : ()}{\Gamma \vdash \bar{a}\langle b_1 \ldots b_n \rangle.P : ()}$$

Kobayashi, Pierce, Turner 1998 - Linear Pi-Calculus

Linear Types for Channels

Consider a web server.

- it receives any number of requests, and a channel on which to send a response.
- it <u>always</u> sends one response.

The interface / specification of our protocol doesn't just include a channel's direction — it also includes how frequently we can use a channel.

How do we introduce this to the π-calculus?

Introduction to Linear Types

Linear types let us specify the usage of resources. We assign to a resource a 'multiplicity' and that resource must be 'used' that number of times.

More specifically, narrowing and contraction are disallowed with a linear type.

To express this, inference rules are expressed as the sum of two separate environments, to highlight that the resource is used correctly between the two.

$$\frac{\Gamma_1 \vdash v \in V \qquad \Gamma_2 \vdash w \in W}{\Gamma_1 + \Gamma_2 \vdash \langle v, w \rangle \in V \times W}$$

Preliminary:

m is a metavariable ranging over multiplicities $(1, \omega)$.

e.g. $r[T_1 \ldots T_n]^1$

Combination of types:

$$I_1[T_1 \ldots T_n]^\omega + I_2[T_1 \ldots T_n]^\omega = (I_1 \cup I_2)[T_1 \ldots T_n]^\omega$$

$$I_1[T_1 \ldots T_n]^1 + I_2[T_1 \ldots T_n]^1 = (I_1 \cup I_2)[T_1 \ldots T_n]^1$$
$$\text{if } I_1 \cap I_2 = \emptyset$$

The capabilities of our channels must be split: a linear channel can be read from 1 process and written from 1 process, but no more.

Typing Rules:

$$\frac{\Gamma_1 \vdash P \qquad \Gamma_2 \vdash Q}{\Gamma_1 + \Gamma_2 \vdash P \mid Q} \quad [E\text{-}PAR]$$

$$\frac{\Gamma \text{ unlimited}}{\Gamma + x : w[T_1 \ldots T_n]^m + y : T_1 \ldots T_n \vdash \bar{x}[y]} \quad [E\text{-}OUT]$$

$$\frac{\Gamma, z : T_1 \ldots T_n \vdash P \qquad \Gamma \text{ unlimited}}{\Gamma + x : r[T_1 \ldots T_n]^m + x(z).P} \quad [E\text{-}IN]$$

$$\frac{\Gamma, x : I[T_1 \ldots T_n]^m \vdash P}{\Gamma \vdash (\nu x : I[T_1 \ldots T_n]^m)\, P} \quad [E\text{-}NEW]$$

$$\frac{\Gamma \vdash P_1 \qquad \Gamma \vdash P_2}{\Gamma + b : Bool \vdash \text{if } b \text{ then } P_1 \text{ else } P_2} \quad [E\text{-}IF]$$

Summary

- The key component to the process calculus is mobility, the ability for the communication between processes to evolve dynamically.

- Our notion of the Truth has changed — processes simulate one another, and can thus be equated ... somehow.

- We can specify our processes across multiple dimensions, particularly by specifying the interface of a channel.

In Retrospect

The π-calculus has fallen short in two ways:

1. There is no notion of the single π-calculus. For every paper on the subject, you have a different calculus, with varying central operators, and different notions of bisimulation.

2. Nobody programs in the π-calculus. Channels are a common component of process calculi. The π-calculus describes concurrent communication via symbol-pushing, and that doesn't line up with real concurrent programs.