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

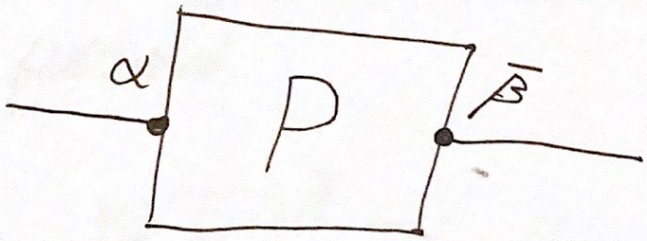
Preliminaries:

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.

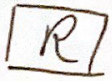
We define processes via their interactions with processes over channels.

Channels are a static property of a process.

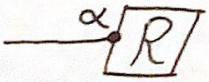


Example: checkout at grocery store.

1. nil
   - a closed checkout aisle.

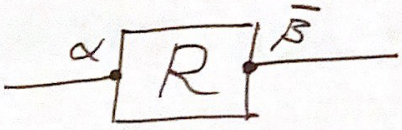


2. receive
   - the cashier receives 5$ from customer.



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

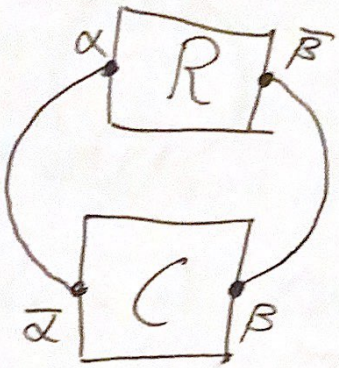
3. send
   - the cashier gives the customer what they purchased.



$$\bar{\beta}\,gum.R' \xrightarrow{\beta\ gum} R'$$

4. Parallel (Par)

- A cashier and customer communicating



$$(\alpha(x).\ \bar{\beta}\,gum.\ R' \mid \bar{\alpha}\,\$5.\ \beta(y).\ C')$$

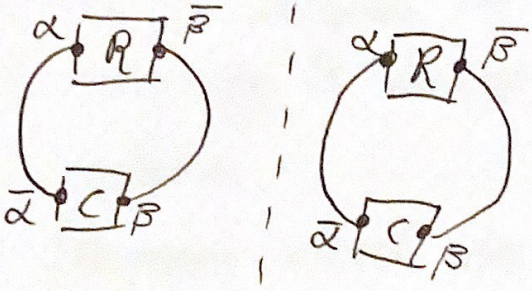
$$\xrightarrow{\alpha\,\$5} (\bar{\beta}\,gum.\ R'[x \leftarrow \$5] \mid \beta(y).\ C')$$

$$\xrightarrow{\beta\,gum} (R'[x \leftarrow \$5] \mid C'[y \leftarrow gum])$$

Note: From the perspective of an outside observer, they don't interact with the larger system. These 'unobserved' actions are called 'silent' actions labeled with τ-transitions ($\xrightarrow{\tau}$). This distinction is mostly useful for theoretical purposes.

5. Restriction

- Cashiers & customers in different aisles.



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

Not the same.

6. Summation - Non-Determinism

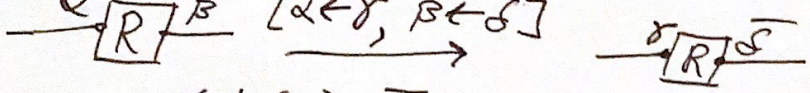
- The cashier might ask if you have a rewards card.

$\alpha(x).R_1 + \alpha(x).R_2$

$\xrightarrow{\alpha\$} R_2[x \leftarrow \$5]$

7. Relabelling

- When a channel must be renamed do so explicitly.



$(\alpha(x).\bar{\beta}\,gum.R')[\alpha \leftarrow \gamma, \beta \leftarrow \gamma)$

$\rightarrow (\gamma(x).\bar{\delta}\,gum.R'[\ldots])$

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

Problem: where did the Truth go?

In the λ-calculus, the Truth is Observational Equivalence. Its 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.

Milner defines three such relations for CCS.

Example:

The "strong equivalence relation:

$P \sim Q$ iff

1. if $P \xrightarrow{\mu v} P'$, then for some Q', $Q \xrightarrow{\mu v} Q'$ and $P' \sim Q'$.
2. if $Q \xrightarrow{\mu v} Q'$, then for some P', $P \xrightarrow{\mu v} P'$ and $Q' \sim P'$.

1.	$\bar{\beta}5.P'$	$\bar{\alpha}5.Q'$	X
2.	$\bar{\beta}5.P'$	$\bar{\beta}5.P'$	✓
3.	$\bar{\beta}5(\bar{\alpha}6.nil \mid \alpha(y).\bar{\gamma}y.nil)$	$\bar{\beta}5.\tau.\bar{\gamma}6.nil$	✓
4.	$\bar{\beta}5(\bar{\alpha}6.nil \mid \alpha(y).\bar{\gamma}y.nil)$	$\bar{\beta}5.\bar{\gamma}6.nil$	X

To equate 4:

$$\xRightarrow{\mu v} = \xrightarrow{\tau} \xrightarrow{\tau} \dots \xrightarrow{\mu v} \xrightarrow{\tau} \dots \xrightarrow{\tau}$$

$P \approx Q$ iff

1. if $P \xRightarrow{\mu v} P'$, then for some Q', $Q \xRightarrow{\mu v} Q'$ and $P' \approx Q'$.
2. if $Q \xRightarrow{\mu v} Q'$, then for some P', $P \xRightarrow{\mu v} P'$ and $Q' \approx P'$.

Flow: this is <u>not</u> a congruence 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. "... Its primitive constructs deserve re-examination.
   - Are there unnecessary operators?
   - Are other operators able to create a richer calculus?

Intermediate Step - Ufferberg & Nielsen, 1986

Solution: allow processes to pass channels over channels.
Extended CCS (ECCS).

Two disjoint sets of variables:
- 'value' variables: $x, y, z \ldots$
- name / channel variables: $a, b, c \ldots$

Still use α, β to represent actual channels.
Example: Walkie-Talkies.

$$(\nu\beta)(\overline{\alpha}\beta.\beta(x).P' \mid \alpha(talk).\overline{talk}\text{ "hi"}.Q') \mid \overline{\gamma}\text{ "hello?"}.R'$$

$$\xrightarrow{\alpha\beta} (\nu\beta)(\beta(x).P' \mid \overline{\beta}\text{ "hi"}.Q'[talk \leftarrow \beta]) \mid \overline{\gamma}\text{ "hello?"}.R'$$

$$\xrightarrow{\beta\text{"hi"}} (\nu\beta)(P'[x \leftarrow \text{"hi"}] \mid Q'[talk \leftarrow \beta]) \mid \overline{\gamma}\text{ "hello?"}.R'$$

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.

π-Calculus Reflection

Characteristic Examples

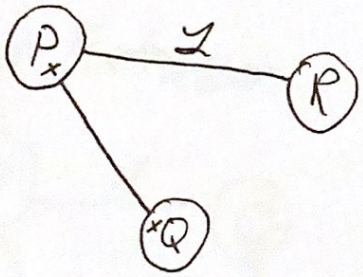
1. Passing Channels

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

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

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

2. Restricting Channels



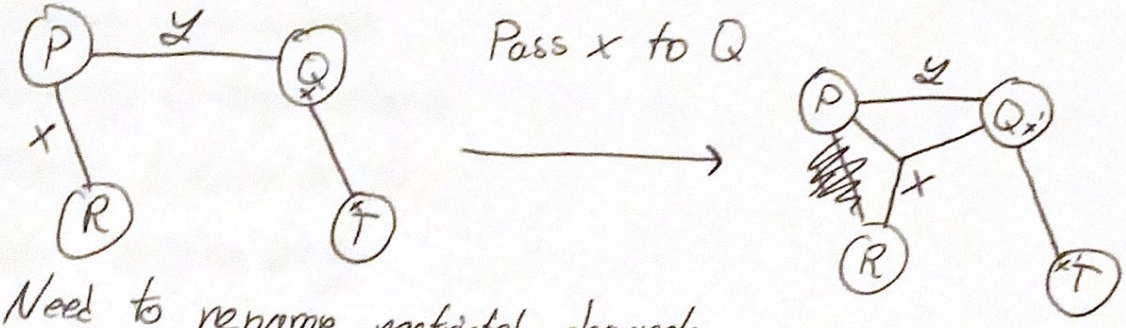
$$(\nu x)(\bar{x}5.P' \mid x(y).Q') \mid R$$

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

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

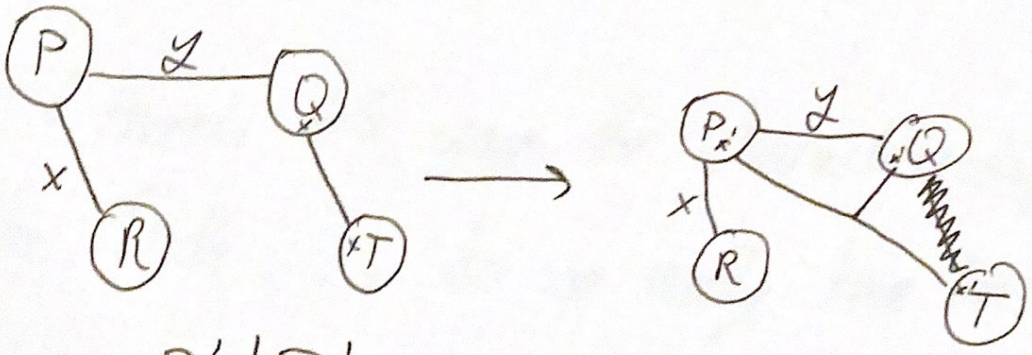


Need to rename restricted channels.

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

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

4. Scope Extrusion



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

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

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
- open bisimilarity

. . .

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

π-Calculus Reflection

A relatively small step (compared to the preceding one) is what puts the calculus over the edge.

The π-calculus goes on to dominate the world of process calculi.

Flaw: the world of concurrent communication is chaotic, and the π-calculus offers no way of remedying that.

- Race Conditions
- Non-determinism
- Dead-locking

Example: the molecular action:

$$(\nu x)(\bar{x}5.\bar{x}6.P' \mid x(y).x(z).Q' \mid x(y).x(z).R')$$

Values are NOT broadcast over channels.

$$\xrightarrow{x5} (\nu x)(\bar{x}6.P' \mid x(z).Q'[y \leftarrow 5] \mid x(y).x(z).R')$$

$$\xrightarrow{x6} (\nu x)(P' \mid x(z).Q'[y \leftarrow 5] \mid x(z).R'[y \leftarrow 6])$$

We've failed! A race condition has occurred.

One possible solution: send a channel of communication first.

$P = (\nu w)(\bar{x}w.\bar{w}5.\bar{w}6.P')$

$Q = x(w).w(y).w(z).Q'$

$R = x(w).w(y).w(z).R'$

The π-calculus is expressive and powerful, but intricate.

3. Milner 1991 — A Sorting Discipline

Solution: add a sorting discipline to the π-calculus (analogous to a typing discipline).

A sorting discipline lets the programmer specify how a process must interact with a channel.

Intermediate Step: Polyadic π-calculus

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

$P = \bar{x}\langle 5, 6 \rangle . P'$

$Q = x(y, z) . Q'$

$R = x(y, z) . R'$

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

The Sorting Discipline

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

A channel has a sort S. Our sorting contains the sorts of channels.

{ SENDER: [LEFT, RIGHT], LEFT: [], RIGHT: [] }

Note: we need names because our types are recursive.

Inference Rules:

We show that a process P obeys the sorts of the channels it interacts with.

Preliminary:

- $[x_1, x_2 \ldots x_n] \rightarrow [\tilde{x}]$
- A judgement $\Gamma \vdash P$ should be read as:
  "Process P is well-behaved under gamma."

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

$$\frac{\Gamma(x) = [\Gamma(\tilde{b})] \quad \Gamma \vdash P}{\Gamma \vdash \bar{x}\langle\tilde{b}\rangle.P} \quad [T\text{-}OUT]$$

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

Note: Processes used to have type []. This was dropped later upon realizing that all processes that checked had "one type."

Reflection

The sorting guarantees no arity runtime errors.
Is this what we want?

This discipline is <u>unsatisfactory.</u> We've learned relatively little about how to use channels.

Imagine if modern programming languages only let the programmer specify the arities of functions?

Example: office printer.
job, paper ∈ channels.

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

If another process reads on the job channel, they can steal a printing job!

Pierce & Sangiorgi 1994 – I/O Types

Solution: specify the read-write interface of a channel.

Preliminaries:

Polarities = $\{i, o\}$

$r = \{i\}$

$w = \{o\}$

$b = \{i, o\}$

Lowercase p ranges over polarities.

A type is written as:

$$S = \{ \text{NAME}: p[S_1, S_2, \ldots] \}$$

For example, take the office printer:

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

A client Q will receive the job channel and specify its interface, write access only:

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

We need a way to show that a sort obeys the interface of a channel. For that, we introduce subsorting.

We specify that:

$b <: r$

$b <: w$

Intuition: a sort is a subsort of another if the interface is supported and the children obey the same relationship.

Example subsort rule:

$$\frac{p <: r \qquad \text{for each } i,\ \Sigma \vdash S_i <: T_i}{\Sigma \vdash p[\tilde{S}] <: r[\tilde{T}]}$$

Sorting rules with I/O & subsorting

Two rules are modified:

$$\frac{\Gamma(x) <: r[\tilde{s}] \qquad \Gamma, [\tilde{b}] : [\tilde{s}] \vdash P}{\Gamma \vdash x(\tilde{b} : \tilde{s}).P} \quad [T\text{-}IN]$$

$$\frac{\Gamma(x) <: w[\Gamma(\tilde{b})] \qquad \Gamma \vdash P}{\Gamma \vdash \bar{x}\langle \tilde{b} \rangle.P} \quad [T\text{-}OUT]$$

Reflection on I/O Types

We've specified a critical component of a channel's interface. But we're still missing another frequent part of a protocol.

Consider a webserver:

- it receives any number of requests, and a channel on which to send a response (in the π-calculus)
- it always sends <u>one</u> response per request.

A channel's interface also includes how many times we must use a channel.

It would be great if our discipline could catch the programmer when they forget to send a reply on a 300 request.

Kobayashi, Pierce, Turner 1998 - Linear Types

Objective: allow the programmer to specify the usage of channels on top of the I/O interface and the actors.

Linear type system will be leveraged to achieve this.

Introduction to Linear Types

Linear types let us specify & enforce the usage of resources. We assign to each resource a 'multiplicity' that indicates how many times it must be used.

This prevents two other features in type systems:

1. Weakening (Dropping):

(first x y) => x NOT ALLOWED

$\Gamma, x:T \vdash x:T$ ✗

$x:T \vdash x:T$ ✓

We say 'Γ unlimited' to specify an environment with no limited resources.

2. Contraction (Duplication):

(define (double x) (+ x x)) NOT ALLOWED

$$\frac{\Gamma \vdash v \in V \quad \Gamma \vdash w \in W}{\Gamma \vdash \langle v, w \rangle \in V \times W}$$ ✗

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

Preliminaries:

m is a metavariable ranging over multiplicities (in our case, 1 or ω).

A sort/type takes the form:

$$p[S_1 \dots S_n]^m$$

Combination of types:

$$p[S_1 \dots S_n]^\omega + q[S_1 \dots S_n] = (p \cup q)[S_1 \dots S_n]^\omega$$

$$p[S_1 \dots S_n]^1 + q[S_1 \dots S_n] = (p \cup q)[S_1 \dots S_n]^1$$

if $p \cap q = \emptyset$

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

Typing Rules:

$$\frac{\Gamma, \tilde{z}: \tilde{S} \vdash P \qquad \Gamma \text{ unlimited}}{\Gamma, x: r[\tilde{S}]^m \vdash x(z).P} \; [\Gamma\text{-IN}]$$

$$\frac{\Gamma \text{ unlimited}}{\Gamma, x: w[\tilde{S}]^m, z: \tilde{S} \vdash \bar{x}\langle \tilde{y} \rangle.P} \; [\Gamma\text{-OUT}]$$

$$\frac{\Gamma_1 \vdash P \qquad \Gamma_2 \vdash Q}{\Gamma_1, \Gamma_2 \vdash P \mid Q} \; [\Gamma\text{-PAR}]$$

Reflection on Linear Types

Returning to the web server: we've begun to create something more powerful than just the interface of a channel – we're beginning to specify protocols.

The element of typing in the π-calculus plays a big role in the future of work on communicating and mobile processes (Andreas' Talk).

Summary

- The key component to the π-calculus that makes it work is 'mobility,' the ability for the communication between processes to change over time.

- Our notion of the Truth has changed in this work. Processes simulate one another, and can thus be equated . . . somehow.

- We can specify our communication channels across multiple dimensions:
  - their arities
  - their read/write capabilities
  - the ~~total~~ quantity of their usage.

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.