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

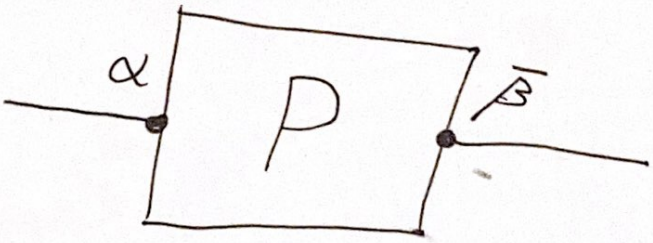
Preliminaries:

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.

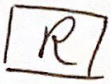
We define processes via their interactions with processes over channels.

Channels are a static property of a process.

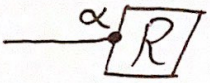


Example: Checkout at grocery store.

1. nil
   - a closed checkout aisle.

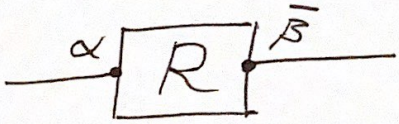


2. receive
   - the cashier receives 5$ from customer.



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

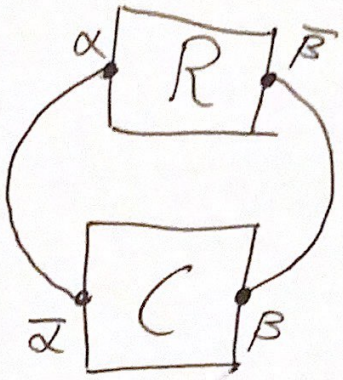
3. send
   - the cashier gives the customer what they purchased.



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

4. Parallel (Par)

- A cashier and customer communicating



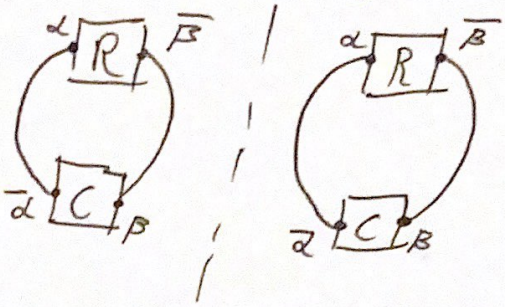
$$(\alpha(x).\ \bar{\beta}\,gum.\ R' \mid \bar{\alpha}\,\$5.\ \beta(y).\ C')$$

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

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

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.

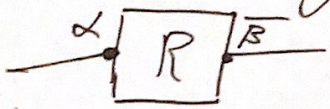
5. Restriction
- Cashiers & customers in different aisles.



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

Not the same!

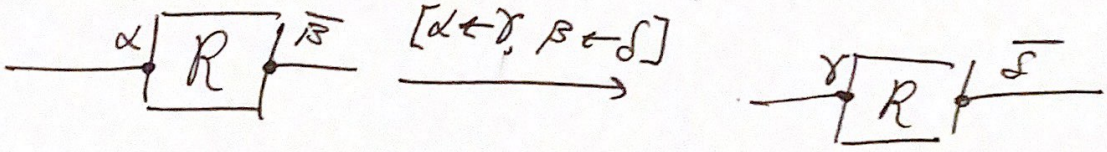
6. Summation / Non-Determinism
- The cashier might ask if you have a rewards card.



$\alpha\$5 \quad \alpha(x).R_1 + \alpha(x).R_2$

$\rightarrow R_2[x \leftarrow \$5]$

7. Relabelling
- Channels must be renamed in the surface syntax.



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

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

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.

Milner defines three such relations for CCS.

Example:

The 'strong equivalence' relation:

$P \sim Q$ iff

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

Examples:

1.	$\bar{\beta}5.P'$	$\alpha 42.Q'$	✗
2.	$\bar{\beta}5.P'$	$\bar{\beta}5.P'$	✓
3.	$\bar{\beta}5(\alpha 6.nil \mid \alpha(y).\bar{\gamma}y.nil)$	$\bar{\beta}5.\tau.\bar{\gamma}6.nil$	✓
4.	$\bar{\beta}5(\alpha 6.nil \mid \alpha(y).\bar{\gamma}y.nil)$	$\bar{\beta}5.\bar{\gamma}6.nil$	✗

We need a new relation to relate 4.

$\xRightarrow{\mu\nu} \stackrel{def}{=} \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{\tau} \cdots \xrightarrow{\mu\nu} \xrightarrow{\tau} \cdots \xrightarrow{\tau}$

$\xRightarrow{\mu\nu} = \xrightarrow{\tau^k} \xrightarrow{\mu\nu} \xrightarrow{\tau^k}$

$P \approx Q$ iff

1. if $P \stackrel{\mu\nu}{\Rightarrow} P'$, then for some Q', $Q \stackrel{\mu\nu}{\Rightarrow} Q'$ and $P' \approx Q'$.
2. if $Q \stackrel{\mu\nu}{\Rightarrow} Q'$, then for some P', $P \stackrel{\mu\nu}{\Rightarrow} P'$ and $Q' \approx P'$.

Flaw: Not a congruence relation.

Reflections on CCS

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

2. "... its primitive constructs deserve re-examination."
   - Are there other operators that make the calculus more expressive?
   - Are there unneeded operators?

Intermediate Step - Ulfberg & 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.

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

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

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

ECCS Reflection

The capacity for the links between processes to change dynamically is called 'mobility' (of channels).

↳ Channels no longer unidirectional.

This also means shrinking the calculus. We no longer need relabelling, now that we've defined substitution over channels.

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

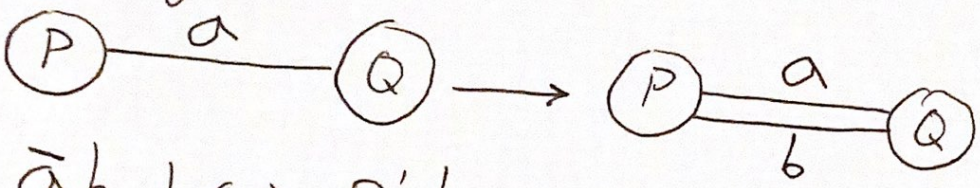
Small improvement: distinguishing between channel and value variables is unnecessary to achieve the same expressiveness.

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

Characteristic Examples

We do not use greek letters for channels anymore

1. Passing Channels

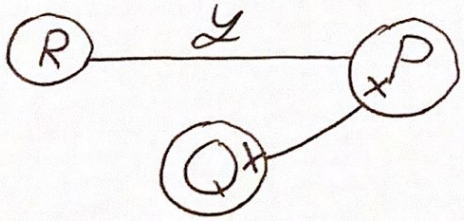


$\bar{a}b.\ b(x).\ P' \mid a(chan).\ \overline{chan}\,5.\ Q'$

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

$\xrightarrow{b5}\ P'[x \leftarrow 5] \mid Q'[chan \leftarrow b]$

2. Restricted Channels

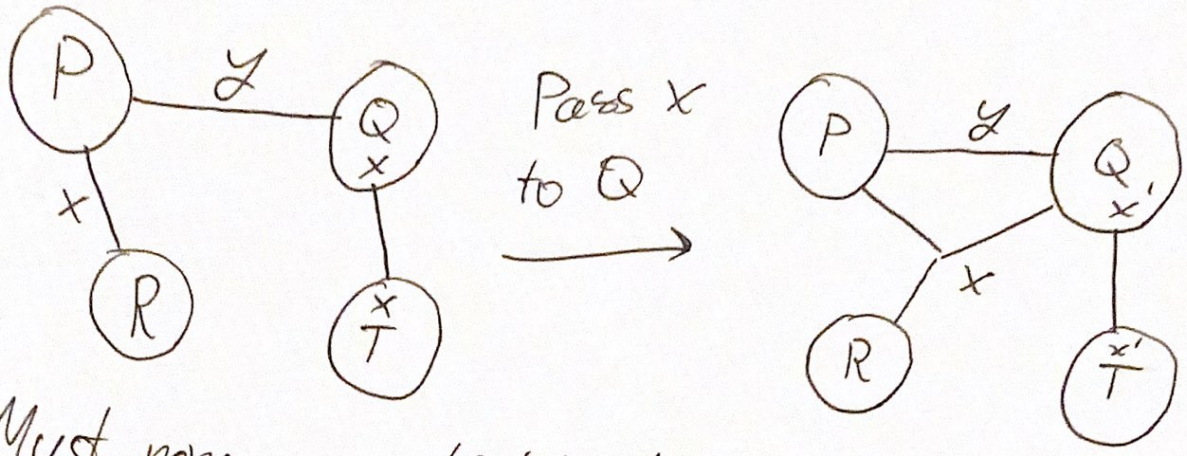


$(\nu x)\,(\bar{x}5.\ P' \mid x(y).\ Q') \mid R$

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

Restriction introduces channels like lambdas introduce variables.

3. Scope Intrusion



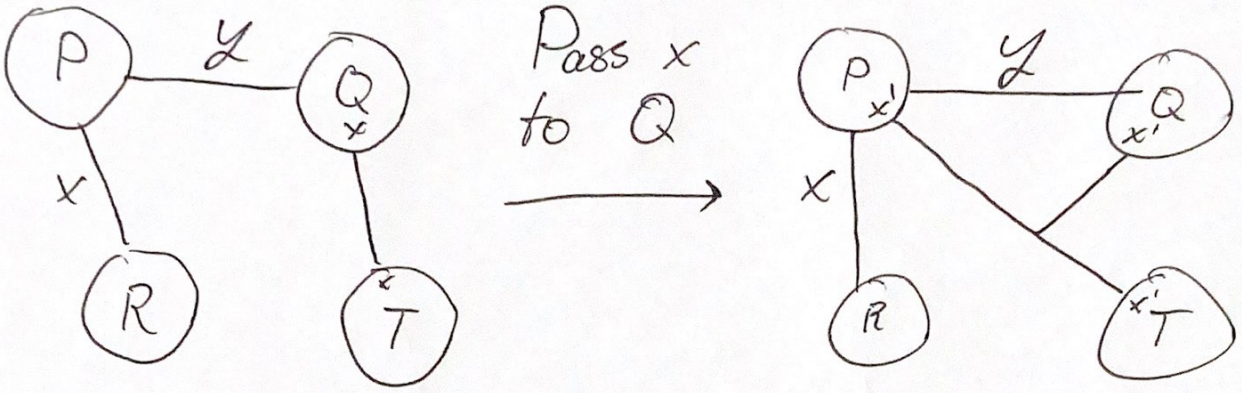
Must rename restricted channels.

$$
\begin{aligned}
&(\overline{y}x.P' \\
&\mid R \\
&\mid (\nu x) \\
&\quad (y(z).Q' \\
&\quad \mid T))
\end{aligned}
$$

$\xrightarrow{yx}$

$$
\begin{aligned}
&(P' \\
&\mid R \\
&\mid (\nu x') \\
&\quad (Q'[x \leftarrow x'][z \leftarrow x] \\
&\quad \mid T[x \leftarrow x']))
\end{aligned}
$$

4. Scope Extrusion



Must rename restricted channels.

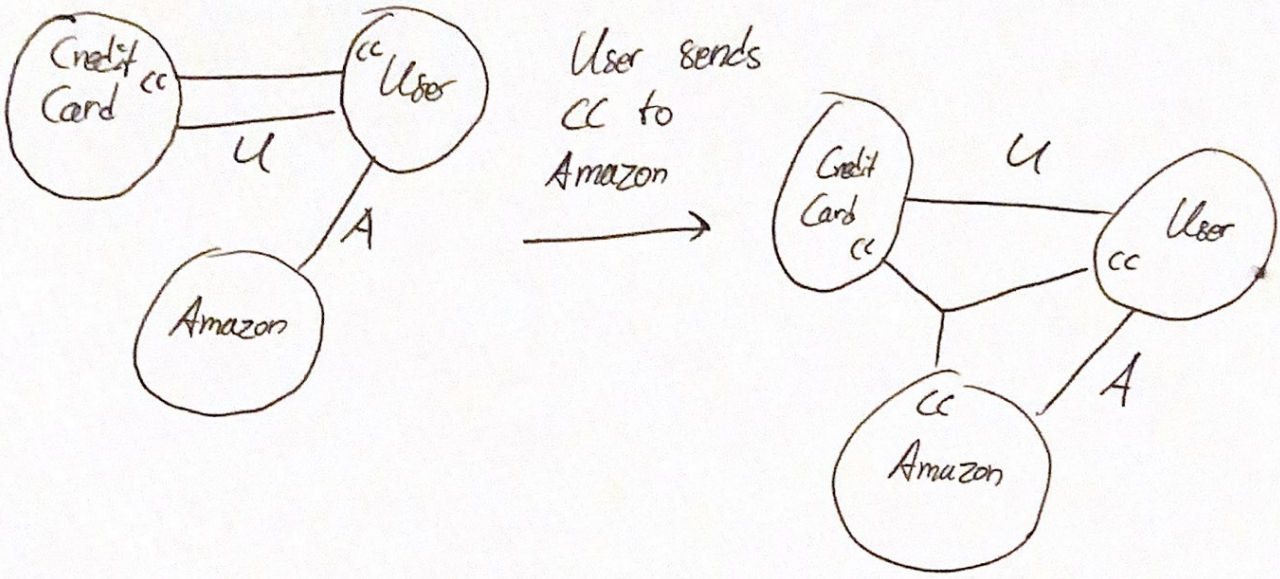
$$
\begin{aligned}
&(y(z).P' \\
&\mid R \\
&\mid (\nu x) \\
&\quad (\bar{y}x.Q' \\
&\quad \mid T))
\end{aligned}
$$

$\xrightarrow{yx'}$

$$
\begin{aligned}
&((\nu x') \\
&\quad (P'[z \leftarrow x'] \\
&\quad \mid Q'[x \leftarrow x'] \\
&\quad \mid T[x \leftarrow x']) \\
&\mid R)
\end{aligned}
$$

Must hoist the channel restriction up so that x' is in scope for P.

Another Example: Amazon Purchase



CreditCard = ()(cc) ($\bar{u}$cc. cc(charge). CreditCard')

User = u(cc). $\bar{A}$book. $\bar{A}$cc. User'

Amazon = A(product). A(cc). $\bar{cc}$price_of_product. Amazon'

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 puts the π-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
- Deadlocking

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.

Example sorting:

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

Add type annotation:

$(\nu x : [\text{LEFT}, \text{RIGHT}])\,(\ldots)$

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:

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

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

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

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

Reflection

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

This discipline is <u>censa fis factory.</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). $\overline{\text{paper}}$<x>. 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 capabilities of each channel.

Preliminaries:

Polarities = $\{i, o\}$

$r = \{i\}$

$w = \{o\}$

$b = \{i, o\}$

Lowercase p, q range over polarities.

A sort now looks like: $p[S_1, S_2, \ldots S_n]$

Office Printer:

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

A client will receive the job channel with only write access:

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

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 relationship

Example subsort rule:

Judgement $\Sigma \vdash S <: T$ means "Sort S is a subsort of Sort T under Σ."

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

Sorting rules with I/O & subsorting

Two rules are modified:

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

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

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 web server:

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

It would be great if our typechecker could let the implementer know that they forgot to send a reply in a particular case.

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 actions.

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

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

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

Inference Rules:

m is either 1 or ω.
Write a type $p[S_1 \dots S_n]^m$.

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

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

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

How do we combine types from a split context?

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

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

$$p[S_1 \dots S_n]^1 + q[S_1 \dots S_n]^1 = (p \cup q)[S_1 \dots S_n]^1$$
$$\text{if } p \cap q = \emptyset$$

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 (Architects' Talk).

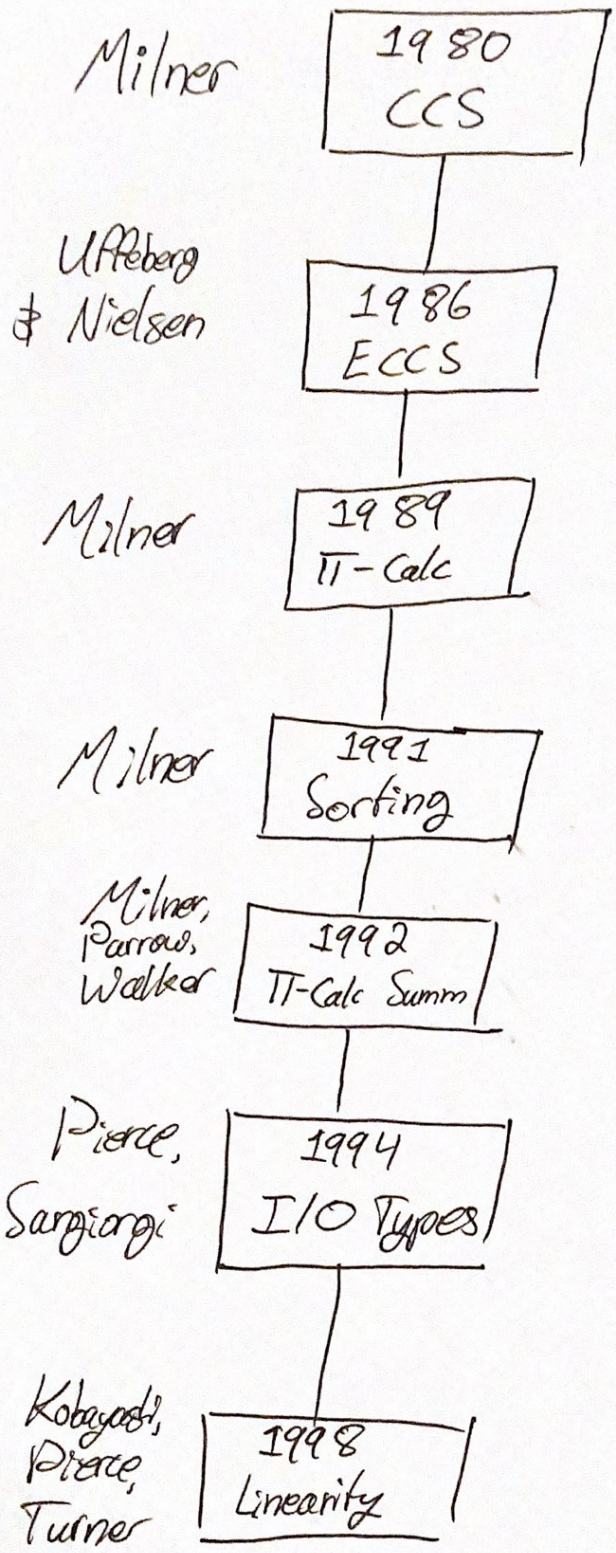
Summary

What have we achieved?

- Our original goal — one process calculus to rule them all.
- The key to the model is mobility of channels.
- We've defined a new notion of the Truth for processes.
- We've introduced discipline into our world.

Shortcomings:

- There is no notion of the definitive Π-calculus.
- Nobody programs in the Π-calculus.


Milner
1980
CCS
Uffeberg & Nielsen
1986
ECCS
Milner
1989
π-Calc
Milner
1991
Sorting
Milner, Parrow, Walker
1992
π-Calc Summ
Pierce, Sangiorgi
1994
I/O Types
Kobayashi, Pierce, Turner
1998
Linearity