A Fully Abstract Domain Model for the π-Calculus

Ian Stark

BRICS

Department of Computer Science

University of Aarhus

Denmark

July 1996

The Issue

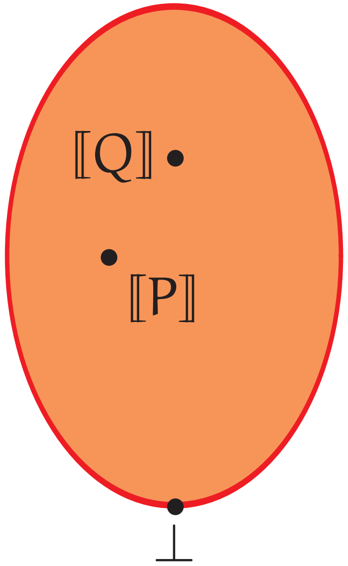
Languages like CCS and the π-calculus provide an algebraic approach to concurrency: structured operational semantics for process behaviour, and bisimulation for equational reasoning about equivalence between processes.

Scott's domain theory provides a mathematical foundation for models of computation: in particular, complete partial orders can express approximations to potentially infinite computation.

Can we usefully employ one to describe the other?

Abramsky's 'Domain Equation for Bisimulation'

$$D \;\cong\; P^0\Big(\sum_{a \in Act} D\Big).$$



This expresses an SCCS process as the collection of actions it can take, and the processes that it may then become.

The interpretation is *compositional*:

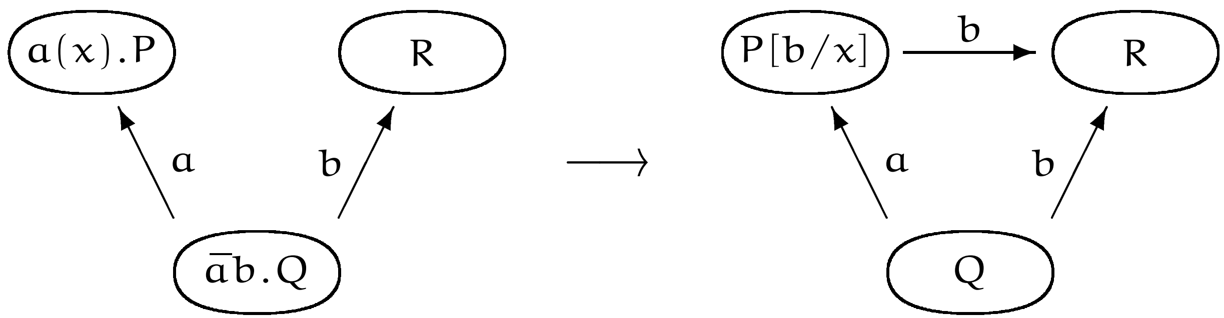
$$prefix : Act \times D \to D \qquad [\![a.P]\!] = prefix(a, [\![P]\!]) \quad \ldots$$

and *fully abstract* for the finitary part of bisimulation:

$$P \sim^F Q \iff [\![P]\!] = [\![Q]\!].$$

A Calculus of Mobile Processes

In the π-calculus, processes pass values that are themselves channel names. This leads to changes in connectivity and allows processes to dynamically reconfigure:



This is much more flexible than CCS: both the λ-calculus and a variety of dynamic distributed systems can be encoded in the π-calculus.

π-Calculus Syntax and Semantics

Processes are built up using a variety of operations:

$$
\begin{array}{ccccc}
0 & P + Q & P \mid Q & \bar{x}y.P & \tau.P \\
\nu x\, P & [x = y]P & [x \neq y]P & x(y).P & !P\,.
\end{array}
$$

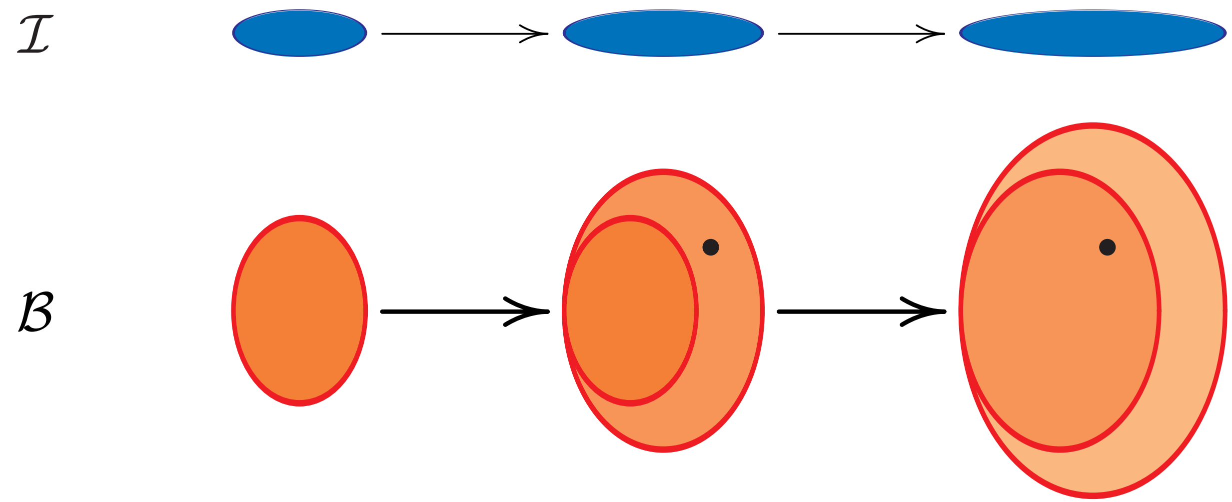
Behaviour is expressed by transitions:

$$
\bar{x}y.P \xrightarrow{\bar{x}y} P \qquad \frac{P \xrightarrow{\bar{x}y} P'}{\nu y\, P \xrightarrow{\bar{x}(y)} P'} \qquad \frac{P \xrightarrow{x(y)} P' \quad Q \xrightarrow{\bar{x}(y)} Q'}{P \mid Q \xrightarrow{\tau} \nu y\,(P \mid Q)} \ldots
$$

We consider the *strong*, *late* semantics, with notions of bisimilarity $P \mathrel{\dot{\sim}} Q$ and equivalence $P \sim Q$ between processes.

Indexed domains

Any π-calculus process is defined over some finite set of free names, which may change as the process performs input and output. We model this with domains that vary according to $\mathcal{I}$, an index category of finite name sets and injections between them.



The properties of *functorality* and *naturality* ensure consistency as the current name set changes over time.

Category $\mathcal{C}$

We take a particular functor category $\mathcal{C}$, with index $\mathcal{I}$ and a base $\mathcal{B}$ of bifinite domains without bottom. This has:

- $A \times B$, $A \to B_\perp$, $P(A)$ for pairing, functions, powerdomain;
- $A \otimes B$, $A \multimap B_\perp$ for privacy and non-interference;
- an object of names N, being the inclusion $\mathcal{I} \hookrightarrow \mathcal{B}$.

In particular an element of $(N \multimap A)s$ is a function that takes any fresh name $x \notin s$ uniformly to an element of $A(s + \{x\})$.

The domain equations

The object Pi is defined as the solution in $\mathcal{C}$ of these domain equations:

$$\begin{aligned}
\mathrm{Pi} &\cong 1 + P(\mathrm{Pi}_\perp + \mathrm{In} + \mathrm{Out}) && 0 \text{ or } \tau.P \text{ or } \ldots \\
\mathrm{In} &\cong N \times (N \to \mathrm{Pi}_\perp) && x(y).P \\
\mathrm{Out} &\cong N \times (N \times \mathrm{Pi}_\perp + (N \multimap \mathrm{Pi}_\perp)) && \overline{x}y.P \text{ or } \overline{x}(z).P.
\end{aligned}$$

An element of $\mathrm{Pi}_\perp s$ is a process with free names in s, expressed as the set of actions it can take and the processes it may then become.

Processes as elements

For each operation of the π-calculus there is a corresponding map, defined abstractly by expanding the equation for Pi:

$$\uplus : Pi_\perp \times Pi_\perp \longrightarrow Pi_\perp \qquad out : N \times N \times Pi_\perp \longrightarrow Pi_\perp \quad \ldots$$

These give a compositional interpretation of processes as domain elements:

$$(\![P + Q]\!)_s = (\![P]\!)_s \uplus (\![Q]\!)_s \qquad (\![\bar{x}y.P]\!)_s = out_s(x, y, (\![P]\!)_s) \quad \ldots$$

Thus any process P with names from s is interpreted by an element $(\![P]\!)_s \in Pi_\perp s$.

Two important operations

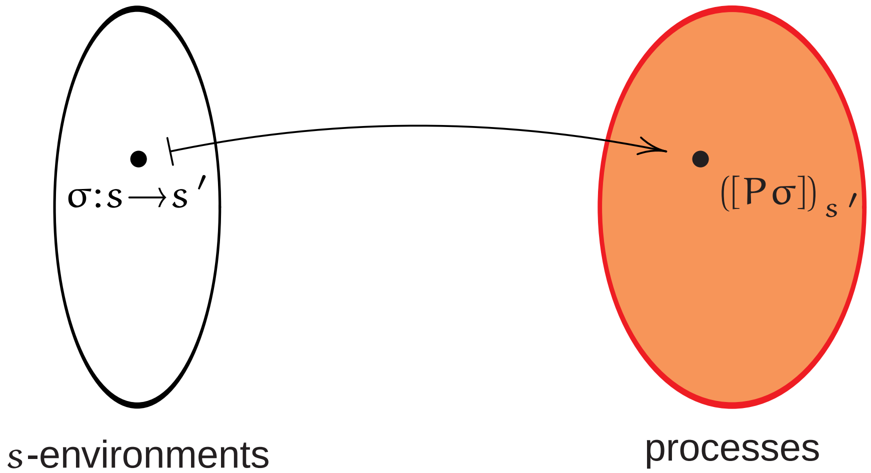
Two particularly significant maps in $\mathcal{C}$:

$$new : (\mathrm{N} \multimap \mathrm{Pi}_\perp) \longrightarrow \mathrm{Pi}_\perp$$
$$par : \mathrm{Pi}_\perp \times \mathrm{Pi}_\perp \longrightarrow \mathrm{Pi}_\perp \,.$$

This new captures name restriction: it takes an agent expecting a name to a process, by providing a fresh private name. The map par interprets parallel composition as interleaving.

Processes as morphisms

$$\mathsf{N}^s \xrightarrow{\quad [\![P]\!]_s \quad} \mathsf{Pi}_\perp$$



This broadens the interpretation of a process, to account for behaviour at all possible name instantiations.

Full abstraction

If P is a π-calculus process then its interpretation in $\mathcal{C}$ both preserves and reflects transitions:

$$P \xrightarrow{\bar{x}y} Q \quad \Longrightarrow \quad \mathit{out}_s(x, y, (\![Q]\!)_s) \in (\![P]\!)_s \qquad \textit{etc.}$$

$$\mathit{tau}_s(q) \in (\![P]\!)_s \quad \Longrightarrow \quad \exists Q \,.\, P \xrightarrow{\tau} Q \quad \& \quad (\![Q]\!)_s = q \qquad \textit{etc.}$$

It follows that the model is fully abstract for bisimulation and equivalence between processes:

$$(\![P]\!)_s = (\![Q]\!)_s \iff P \stackrel{\cdot}{\sim} Q$$

$$[\![P]\!]_s = [\![Q]\!]_s \iff P \sim Q \,.$$

Thus $\mathcal{C}$ can be used to prove equivalences between specific processes, and to verify algebraic laws for the π-calculus.

Applications and extensions

The model

- verifies structural rules ($P + Q \equiv Q + P$), the expansion law, and all other algebraic laws for the π-calculus;
- can represent notions of privacy and non-interference between processes, as in $\mathrm{Pi}_\perp \otimes \mathrm{Pi}_\perp \subseteq \mathrm{Pi}_\perp \times \mathrm{Pi}_\perp$.

Possible extensions are

- variants on the π-calculus, other kinds of bisimilarity;
- a domain logic for mobile processes;
- indexing other models for concurrency.

Summary

A category of domains indexed by $\mathcal{I}$ is a suitable setting in which to construct $\mathtt{Pi}_\perp$, a recursively defined domain that provides a denotational semantics for the π-calculus.

The symmetric monoidal closed structure $(1, \otimes, \multimap)$ on the category is particularly important: it provides abstract notions of ‘independence’ between processes and ‘freshness’ of names.

The interpretation of processes in the category captures exactly their transition behaviour, strong late bisimulation and strong late equivalence.

Summary

Q. Is there a domain model of the π-calculus?

A. Yes, and it is both compositional and fully abstract.

Q. What makes it work?

A. A functor category: domains that vary according to the current set of names.

Q. What does the $(1, \otimes, \multimap)$ structure do?

A. It provides abstract notions of 'independence' between processes and 'freshness' of names.