

## Pi-calculus

### syntax and reduction semantics

Francesco Zappa Nardelli

INRIA Rocquencourt, MOSCOVA research team

francesco.zappa\_nardelli@inria.fr

MPRI Concurrency course with:

Pierre-Louis Curien (PPS), Roberto Amadio (PPS), Catuscia Palamidessi (INRIA Futurs)

MPRI - Concurrency

October 20, 2006

---

## Challenges (idiosyncratic survey)

- **Local concurrency:**  $\pi$ -calculus, Join, Pict, ...
- Mobile computations: JoCaml, Nomadict Pict, ...
- Marshalling: choice of distributed abstractions, and trust assumptions: Acute, HashCaml, ...
- Dynamic (re)binding and evaluation strategies: exchanging values between programs
- Type equality between programs: run-time type names, type-safe and abstraction-safe interaction (and type equality within programs)
- Typed interaction handles: establishing shared expression-level names between programs
- Version change: type safety in the presence of dynamic linking. Controlling dynamic linking. Dynamic update
- Semantics for real-world network abstractions, TCP, UDP, Sockets
- Security: security policies, executing untrusted code, protocols, language based
- Module structure again: first-class/recursive/parametric modules. Exposing interfaces to other programs via communication

2

---

## High-level programming languages

For non-distributed, non-concurrent programming, they are pretty good. We have ML (SML/OCaml), Haskell, Java, C#, with:

- type safety
- rich concrete types — data types and functions
- abstraction mechanisms for program structuring — ML modules and abstract types, type classes and monads, classes and objects, ...

But this is only within **single executions** of **single, sequential** programs.

What about distributed computation?

1

---

## Local concurrency

**Local:** within a single failure domain, within a single trust domain, low-latency interaction.

- Pure (implicit parallelism or skeletons — parallel map, etc.)
- Shared memory
  - mutexes, cvars (**incomprehensible, uncomposable, common**)
  - transactional (Venari, STM Haskell/Java, AtomCaml, ...)
- Message passing
  - semantic choices: asynchronous/synchronous, different synchronisation styles (CSP/CCS, Join, ...), input-guarded/general nondeterministic choice, ...
  - cf Erlang [AVWW96], Telescript, Facile [TLK96,Kna95], Obliq [Car95], CML [Rep99], Pict [PT00], JoCaml [JoC03], Alice [BRS+05], Esterel [Ber98], ...

3

---

## In these lectures...

- Simplify by considering just interaction.
- What are the equations of interactions?
- Find a logic for interaction.
- Find new/correct paradigms for programming.
- What's about distribution?
- Mobility?
- Security?

Understand some key concepts behind concurrency theory  
(from a programming language perspective).

---

4

---

## CCS, synchronisation

In CCS, a system *evolves* when two threads *synchronise* over the same name:

$$\bar{b}.P \parallel b.Q \rightarrow P \parallel Q$$

We will focus on **reductions** for the time being (that is, forget about LTSs until next lecture). Summary...

---

5

---

## CCS, reduction semantics

We define **reduction**, denoted  $\rightarrow$ , by

$$a.P \parallel \bar{a}.Q \rightarrow P \parallel Q$$

$$\frac{P \rightarrow P'}{P \parallel Q \rightarrow P' \parallel Q} \quad \frac{P \rightarrow P'}{(\nu x)P \rightarrow (\nu x)P'} \quad \frac{P \equiv P' \rightarrow Q' \equiv Q}{P \rightarrow Q}$$

where, the **structural congruence** relation, denoted  $\equiv$ , is defined as:

$$P \parallel Q \equiv Q \parallel P \quad (P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R)$$

$$P \parallel \mathbf{0} \equiv P \quad !P \equiv P \parallel !P \quad (\nu a)P \parallel Q \equiv (\nu a)(P \parallel Q) \text{ if } a \notin \text{fn}(Q)$$

**Theorem**  $P \rightarrow Q$  iff  $P \xrightarrow{\tau} \equiv Q$ .

---

6

---

## Value passing

Names can be interpreted as *channel names*: allow channels to carry values, so instead of pure outputs  $\bar{a}.P$  and inputs  $a.P$  allow e.g.:  $\bar{a}\langle 15, 3 \rangle.P$  and  $a(x, y).Q$ .

Value 6 being sent along channel  $x$ :

$$\bar{x}\langle 6 \rangle \parallel x(u).\bar{y}\langle u \rangle \rightarrow (\bar{y}\langle u \rangle)\{^6/u\} = \bar{y}\langle 6 \rangle$$

Restricted names are different from all others:

$$\begin{aligned} \bar{x}\langle 5 \rangle \parallel (\nu x)(\bar{x}\langle 6 \rangle \parallel x(u).\bar{y}\langle u \rangle) &\rightarrow \bar{x}\langle 5 \rangle \parallel (\nu x)(\bar{y}\langle 6 \rangle) \\ &\equiv \\ \bar{x}\langle 5 \rangle \parallel (\nu x')(\bar{x}'\langle 6 \rangle \parallel x'(u).\bar{y}\langle u \rangle) &\rightarrow \bar{x}\langle 5 \rangle \parallel (\nu x'')(\bar{y}\langle 6 \rangle) \end{aligned}$$

(note that we are working with alpha equivalence classes).

---

7

---

## Exercise

Program a server that increments the value it receives.

$$!x(u).\bar{x}\langle u + 1 \rangle$$

Argh!!! This server exhibits exactly the problems we want to avoid when programming concurrent systems:

$$\begin{aligned} \bar{x}\langle 3 \rangle.x(u).P \parallel \bar{x}\langle 7 \rangle.x(v).Q \parallel !x(u).\bar{x}\langle u + 1 \rangle &\rightarrow \dots \\ \dots &\rightarrow P\{8/u\} \parallel Q\{4/u\} \parallel !x(u).\bar{x}\langle u + 1 \rangle \end{aligned}$$

---

8

---

## The $\pi$ -calculus

1. A name received on a channel can then be used itself as a channel name for output or input — here  $y$  is received on  $x$  and the used to output 7:

$$\bar{x}\langle y \rangle \parallel x(u).\bar{u}\langle 7 \rangle \rightarrow \bar{y}\langle 7 \rangle$$

2. A restricted name can be sent outside its original scope. Here  $y$  is sent on channel  $x$  outside the scope of the  $(\nu y)$  binder, which must therefore be moved (with care, to avoid capture of free instances of  $y$ ). This is *scope extrusion*:

$$\begin{aligned} (\nu y)(\bar{x}\langle y \rangle \parallel y(v).P) \parallel x(u).\bar{u}\langle 7 \rangle &\rightarrow (\nu y)(y(v).P \parallel \bar{y}\langle 7 \rangle) \\ &\rightarrow (\nu y)(P\{7/v\}) \end{aligned}$$

---

10

---

## Ideas...

Allow those values to include channel names.

A new implementation for the server:

$$!x(u, r).\bar{r}\langle u + 1 \rangle$$

This server prevents confusion provided that the return channels are distinct.

How can we guarantee that the return channels are distinct?

The restriction operator we have is overly restrictive...

---

9

---

## The (simplest) $\pi$ -calculus

Syntax:

$P, Q$	$::=$	<b>0</b>	nil
		$P \parallel Q$	parallel composition of $P$ and $Q$
		$\bar{c}\langle v \rangle.P$	output $v$ on channel $c$ and resume as $P$
		$c(x).P$	input from channel $c$
		$(\nu x)P$	new channel name creation
		$!P$	replication

Free names (alpha-conversion follows accordingly):

$$\begin{aligned} \text{fn}(\mathbf{0}) &= \emptyset & \text{fn}(P \parallel Q) &= \text{fn}(P) \cup \text{fn}(Q) \\ \text{fn}(\bar{c}\langle v \rangle.P) &= \{c, v\} \cup \text{fn}(P) & \text{fn}(c(x).P) &= (\text{fn}(P) \setminus \{x\}) \cup \{c\} \\ \text{fn}((\nu x)P) &= \text{fn}(P) \setminus \{x\} & \text{fn}(!P) &= \text{fn}(P) \end{aligned}$$

---

11

---

## π-calculus, reduction semantics

Structural congruence:

$$\begin{aligned}
 P \parallel 0 &\equiv P & P \parallel Q &\equiv Q \parallel P \\
 (P \parallel Q) \parallel R &\equiv P \parallel (Q \parallel R) & !P &\equiv P \parallel !P \\
 (\nu x)(\nu y)P &\equiv (\nu y)(\nu x)P \\
 P \parallel (\nu x)Q &\equiv (\nu x)(P \parallel Q) \text{ if } x \notin \text{fn}(P)
 \end{aligned}$$

Reduction rules:

$$\bar{c}(v).P \parallel c(x).Q \rightarrow P \parallel Q\{v/x\}$$

$$\frac{P \rightarrow P'}{P \parallel Q \rightarrow P' \parallel Q} \quad \frac{P \rightarrow P'}{(\nu x)P \rightarrow (\nu x)P'} \quad \frac{P \equiv P' \rightarrow Q' \equiv Q}{P \rightarrow Q}$$

---

12

---

## Example: polyadic with monadic

Let us extend our notion of monadic channels, which carry exactly one name, to polyadic channels, which carry a vector of names, i.e.

$$\begin{array}{l}
 P ::= \bar{x}\langle y_1, \dots, y_n \rangle.P \quad \text{output} \\
 \quad \mid x(y_1, \dots, y_n).P \quad \text{input}
 \end{array}$$

with the main reduction rule being:

$$\bar{x}\langle y_1, \dots, y_n \rangle P \parallel x\langle z_1, \dots, z_n \rangle.Q \rightarrow P \parallel Q\{y_1, \dots, y_n / z_1, \dots, z_n\}$$

Is there an **encoding** from polyadic to monadic channels?

---

14

---

## Expressiveness

A small calculus (and the semantics only involves name-for-name substitution, not term-for-variable substitution), but very expressive:

- encoding data structures
- encoding functions as processes (Milner, Sangiorgi)
- encoding higher-order π (Sangiorgi)
- encoding synchronous communication with asynchronous (Honda/Tokoro, Boudol)
- encoding polyadic communication with monadic (Quaglia, Walker)
- encoding choice (or not) (Nestmann, Palamidessi)
- ...

---

13

---

## Polyadic with monadic, ctd.

We might try:

$$\begin{aligned}
 [[\bar{x}\langle y_1, \dots, y_n \rangle.P]] &= \bar{x}\langle y_1 \rangle. \dots \bar{x}\langle y_n \rangle. [[P]] \\
 [[x\langle y_1, \dots, y_n \rangle.P]] &= x(y_1). \dots x(y_n). [[P]]
 \end{aligned}$$

but this is broken! Why?

The right approach is use new binding:

$$\begin{aligned}
 [[\bar{x}\langle y_1, \dots, y_n \rangle.P]] &= (\nu z)(\bar{x}\langle z \rangle. \bar{z}\langle y_1 \rangle. \dots \bar{z}\langle y_n \rangle. [[P]]) \\
 [[x\langle y_1, \dots, y_n \rangle.P]] &= x(z). z(y_1). \dots z(y_n). [[P]]
 \end{aligned}$$

where  $z \notin \text{fn}(P)$  (why?). (We also need some well-sorted assumptions.)

---

15

---

## Recursion

Alternative to replication: **recursive definition** of processes.

Recursive definition:

$$K = (\tilde{x}).P$$

Constant application:

$$K[\tilde{a}]$$

Reduction rule:

$$\frac{K = (\tilde{x}).P}{K[\tilde{a}] \rightarrow P\{\tilde{a}/\tilde{x}\}}$$

---

16

---

## Data as processes: booleans

Consider the truth-values {True, False}. Consider the abstractions:

$$T = (x).x(t, f).\bar{t}\langle \rangle \quad \text{and} \quad F = (x).x(t, f).\bar{f}\langle \rangle$$

These represent a *located copy* of a truth-value at  $x$ . The process

$$R = (\nu t)(\nu f)\bar{b}\langle t, f \rangle.(t().P \parallel f().Q)$$

where  $t, f \notin \text{fn}(P, Q)$  can test for a truth-value at  $x$  and behave accordingly as  $P$  or  $Q$ :

$$R \parallel T[b] \rightarrow\rightarrow P \parallel (\nu t, f)f().Q$$

The term obtained **behaves** as  $P$  because the thread  $(\nu t, f)f().Q$  is deadlocked.

---

18

---

## Recursion vs. Replication

**Theorem** Any process involving recursive definitions is representable using replication, and conversely replication is redundant in presence of recursion.

The proof requires some techniques we have not seen, but...

Intuition: given

$$F = (\tilde{x}).P$$

where  $P$  may contain recursive calls to  $F$  of the form  $F[\tilde{z}]$ , we may replace the RHS with the following process abstraction containing no mention of  $F$ :

$$(\tilde{x}).(\nu f)(\bar{f}\langle \tilde{x} \rangle \parallel !f(\tilde{x}).P')$$

where  $P'$  is obtained by replacing every occurrence of  $F[\tilde{z}]$  by  $\bar{f}\langle \tilde{z} \rangle$  in  $P$ , and  $f$  is **fresh** for  $P$ .

---

17

---

## Data as processes: integers

Using a unary representation.

$$[[k]] = (x).x(z, o).(\bar{o}\langle \rangle)^k.\bar{z}\langle \rangle$$

where  $(\bar{o}\langle \rangle)^k$  abbreviates  $\bar{o}\langle \rangle.\bar{o}\langle \rangle \dots \bar{o}\langle \rangle$  ( $k$  occurrences).

Operations on integers can be expressed as processes. For instance,

$$\text{succ} = (x, y).!x(z, o).\bar{o}\langle \rangle.\bar{y}\langle z, o \rangle$$

Which is the role of the final output on  $z$ ? (Hint: omit it, and try to define the test for zero).

---

19

---

## Another representation for integers

```
type Nat = zero | succ Nat
```

Define:

```
[[zero]] = (x).!x(z, s).z̄⟨⟩  
[[succ]] = (x, y).!x(z, s).s̄⟨y⟩
```

and for each  $e$  of type Nat:

```
[[succ e]] = (x).(νy)([[succ]] [x, y] || [[e]] [y])
```

*This approach generalises to arbitrary datatypes.*

---

20

---

## A step backward: defining a language

Recipe:

1. define the *syntax* of the language (that is, specify what a program is);
2. define its *reduction semantics* (that is, specify how programs are executed);
3. define when *two terms are equivalent* (that is, hum...?!).

Share and enjoy the new language...

---

21

---

## Equivalent?

Suppose that  $P$  and  $Q$  are equivalent (in symbols:  $P \simeq Q$ ).

Which properties do we expect?

**Preservation under contexts** For all contexts  $C[-]$ , we have  $C[P] \simeq C[Q]$ ;

**Same observations** If  $P \downarrow x$  then  $Q \downarrow x$ , where  $P \downarrow x$  means that we can observe  $x$  at  $P$  (or  $P$  can do  $x$ );

**Preservation of reductions**  $P$  and  $Q$  must mimic their reduction steps (that is, they realise the same nondeterministic choices).

---

22

---

## Formally

A relation  $\mathcal{R}$  between processes is

**preserved by contexts:** if  $P \mathcal{R} Q$  implies  $C[P] \mathcal{R} C[Q]$  for all contexts  $C[-]$ .

**barb preserving:** if  $P \mathcal{R} Q$  and  $P \downarrow x$  imply  $Q \downarrow x$ , where  $P \downarrow x$  holds if there exists  $P'$  such that  $P \rightarrow^* P'$  and  $P' \downarrow x$ , while

$$P \equiv (\nu \tilde{n})(\bar{x}\langle y \rangle.P' \parallel P'') \text{ or } P \equiv (\nu \tilde{n})(x(u).P' \parallel P'') \text{ for } x \notin \tilde{n};$$

**reduction closed:** if  $P \mathcal{R} Q$  and  $P \rightarrow P'$ , imply that there is a  $Q'$  such that  $Q \rightarrow^* Q'$  and  $P' \mathcal{R} Q'$  ( $\rightarrow^*$  is the reflexive and transitive closure of  $\rightarrow$ ).

---

23

---

## Reduction-closed barbed congruence

Let **reduction barbed congruence**, denoted  $\simeq$ , be the largest symmetric relation over processes that is preserved by contexts, barb preserving, and reduction closed.

Remark: reduction barbed congruence is a **weak** equivalence: the number of internal reduction steps is not important in the bisimulation game imposed by “reduction closed”.

---

24

---

## Some equivalences (?)

Compare the processes

1.  $P = \bar{x}\langle y \rangle$  and  $Q = \mathbf{0}$
2.  $P = \bar{a}\langle x \rangle$  and  $Q = \bar{a}\langle z \rangle$
3.  $P = (\nu x)\bar{x}\langle \cdot \rangle.R$  and  $Q = \mathbf{0}$
4.  $P = (\nu x)(\bar{x}\langle y \rangle.R_1 \parallel x(z).R_2)$  and  $Q = (\nu x)(R_1 \parallel R_2\{y/z\})$

Argh... we need other **proof techniques** to show that processes are equivalent!

Remark: we can reformulate *barb preservation* as “if  $P \mathcal{R} Q$  and  $P \Downarrow x$  imply  $Q \Downarrow x$ ”. This is sometimes useful...

---

25

---

## Example: local names are different from global names

Show that in general

$$(\nu x)!P \not\approx !(\nu x)P$$

Intuition: the copies of  $P$  in  $(\nu x)!P$  can interact over  $x$ , while the copies of  $(\nu x)P$  cannot.

We need a process that interacts with another copy of itself over  $x$ , but that cannot interact with itself over  $x$ . Take

$$P = \bar{x}\langle \cdot \rangle \oplus x().\bar{b}\langle \cdot \rangle$$

where  $Q_1 \oplus Q_2 = (\nu w)(\bar{w}\langle \cdot \rangle \parallel w().Q_1 \parallel w().Q_2)$ .

We have that  $(\nu x)!P \Downarrow b$ , while  $!(\nu x)P \not\Downarrow b$ .

---

26

---

## Exercises

1. Compare the transitions of  $F[u, v]$ , where  $F = (x, y).x(y).F[y, x]$  to those of its encoding in the recursion free calculus (use replication).
2. Consider the pair of mutually recursive definitions

$$\begin{aligned} G &= (u, v).(u()).H[u, v] \parallel k().H[u, v] \\ H &= (u, v).v().G[u, v] \end{aligned}$$

Write the process  $G[x, y]$  in terms of replication (you have to invent the technique to translate mutually recursive definitions yourself).

3. Implement a process that negates at location  $a$  the truth-value found at location  $b$ . Implement a process that sums of two integers (using both the representations we have seen).
4. Design a representation for lists using  $\pi$ -calculus processes. Implement list append.

---

27

---

## References

### Books

- Robin Milner, Communicating and mobile systems: the  $\pi$ -calculus. (CUP,1999).
- Robin Milner, Communication and concurrency. (Prentice Hall,1989).
- Davide Sangiorgi, David Walker, The  $\pi$ -calculus: a theory of mobile processes. (CUP, 2001).

### Tutorials available online:

- Robin Milner, The polyadic pi-calculus: a tutorial. Technical Report ECS-LFCS-91-180, University of Edinburgh.
- Joachim Parrow, An introduction to the pi-calculus. <http://user.it.uu.se/~joachim/intro.ps>
- Peter Sewell. Applied pi — a brief tutorial. Technical Report 498, University of Cambridge. <http://www.cl.cam.ac.uk/users/pes20/appi.ps>