

**The full  $\pi$ -calculus: simple and expressive**

## $\pi$ -Refresh. What we know so far?

- We have studied the asynchronous monadic  $\pi$ -calculus
  - no continuation on the output (asynchronous)  $\bar{u}\langle v\rangle.P$
  - only one value is communicated (monadic)  $\bar{u}\langle v\rangle$

|                           |                                                                                      |
|---------------------------|--------------------------------------------------------------------------------------|
| $P, Q ::=$                | processes                                                                            |
| $0$                       | nil process                                                                          |
| $P \mid Q$                | parallel composition of $P$ and $Q$                                                  |
| $(\nu a)P$                | generation of $a$ with scope $P$ (also called <i>restriction</i> )                   |
| $!P$                      | replication of $P$ , i.e. infinite parallel composition $P \mid P \mid P \mid \dots$ |
| $\bar{u}\langle v\rangle$ | output of $v$ on channel $u$                                                         |
| $u(x).P$                  | input of <i>distinct</i> variables $x$ on $u$ , with continuation $P$                |

## $\pi$ -calculus: simple, but expressive

- Why expressive:
  - encoding data structures
  - encoding polyadic communication with monadic primitives
  - encoding synchronous communication with asynchronous (Honda/Tokoro, Boudol)
  - encoding choice (Nestmann, Palamidessi)
  - encoding recursion
  - encoding Higher order functions

Today you will learn how to master that expressiveness ... !!!

## Synchronous $\pi$ -calculus

- It is time to add continuation on output:  $\bar{u}\langle v\rangle.P$
- We can define the synchronous calculus as follows:

$$\bar{a}\langle b\rangle.P \mid a(x).Q \longrightarrow P \mid Q\{b/x\}$$



Q: Can we simulate synchronous communication with asynchronous?

Hint: we need additional messages

## Basic Encoding Definition

- $\llbracket P \rrbracket = Q$  is a function (mapping) from  $P$  to  $Q$ .
- A good mapping should be homomorphic.
  - $\llbracket 0 \rrbracket = 0$
  - $\llbracket P \mid Q \rrbracket = \llbracket P \rrbracket \mid \llbracket Q \rrbracket$
  - $\llbracket (\nu a)P \rrbracket = (\nu a)\llbracket P \rrbracket$
  - $\llbracket !P \rrbracket = !\llbracket P \rrbracket$
- Question:
  - $\llbracket \bar{a}\langle b \rangle.P \rrbracket = \text{some asynchronous } \pi\text{-term}$
  - $\llbracket a(x).P \rrbracket = \text{some asynchronous } \pi\text{-term}$

# Synchronous $\pi$ -calculus

Synchronous



$$\llbracket \bar{u}\langle v \rangle . P \rrbracket = (\nu c)(\bar{u}\langle c \rangle \mid c(y).(\bar{y}\langle v \rangle \mid \llbracket P \rrbracket))$$

$$\llbracket u(x).P \rrbracket = u(y).(\nu d)(\bar{y}\langle d \rangle \mid d(x).\llbracket P \rrbracket)$$

Asynchronous



$$\text{where } y \notin fv(P), c \notin fn(P)$$

$$\text{where } y \notin fv(P), d \notin fn(P)$$

- Note:  $\llbracket P \rrbracket$  represents the formal notation for the encoding of  $P$

- Example:

$$\begin{aligned} \llbracket \bar{b}\langle e \rangle . P \rrbracket \mid \llbracket b(x).Q \rrbracket &\longrightarrow (\nu c)(c(y).(\bar{y}\langle e \rangle \mid \llbracket P \rrbracket) \mid (\nu d)(\bar{c}\langle d \rangle \mid d(x).\llbracket Q \rrbracket)) \\ &\longrightarrow (\nu d)(\bar{d}\langle e \rangle \mid \llbracket P \rrbracket \mid d(x).\llbracket Q \rrbracket) \longrightarrow \llbracket P \rrbracket \mid \llbracket Q \rrbracket\{e/x\} \end{aligned}$$

- How it works:
  - The channel  $u$  is used to exchange a private name  $c$
  - The meaning of  $u$  is that the receiver will be engaged in the rendez-vous with the sender
  - The sender confirms on  $c$  by sending the private channel  $d$
  - Now the actual transmission can occur on the channel  $d$

## Polyadic $\pi$ -calculus

- Monadic channels carry exactly one name:  $\bar{u}\langle v \rangle, u(x)$
- Polyadic channels carry a vector of names:

$$P ::= \begin{array}{ll} u(x_1, \dots, x_n).P & \text{input} \\ \bar{u}\langle v_1, \dots, v_n \rangle.P & \text{output} \end{array}$$

- We can communicate multiple values at the same time
- Reduction Rule:

$$\bar{a}\langle c_1, \dots, c_n \rangle.P \mid a(x_1, \dots, x_n).Q \longrightarrow P \mid Q\{\tilde{c}/\tilde{x}\}$$



Is there an encoding from polyadic to monadic channels?

## Let's Try

- For every complex problem there is a simple solution ... that is wrong :)

$$\llbracket u(x_1, \dots, x_n).P \rrbracket = u(x_1).u(x_2)\dots u(x_n).\llbracket P \rrbracket$$

$$\llbracket \bar{u}\langle v_1, \dots, v_n \rangle.P \rrbracket = \bar{u}\langle v_1 \rangle.\bar{u}\langle v_2 \rangle\dots \bar{u}\langle v_n \rangle.\llbracket P \rrbracket$$

- Why the above encoding is ... wrong ?
- **Hint:** encode two processes in parallel sending on the same channel

$$\llbracket a(x_1, x_2).P_1 \mid a(x_3, x_4).P_2 \mid \bar{a}\langle c_1, c_2 \rangle.P_3 \rrbracket$$

## Encoding Polyadic $\pi$ -calculus

$$\llbracket a(x_1, x_2).P_1 \rrbracket \mid \llbracket a(x_3, x_4).P_2 \rrbracket \mid \llbracket \bar{a}\langle c_1, c_2 \rangle.P_3 \rrbracket = \\ a(x_1).a(x_2).\llbracket P_1 \rrbracket \mid a(x_3).a(x_4).\llbracket P_2 \rrbracket \mid \bar{a}\langle c_1 \rangle.\bar{a}\langle c_2 \rangle.\llbracket P_3 \rrbracket = R$$

- This reduction is fine:

$$R \longrightarrow \llbracket P \rrbracket \{c_1/x_1, c_2/x_2\} \mid a(x_3, x_4).\llbracket P_2 \rrbracket \mid \llbracket P_3 \rrbracket$$

- But this is a mess:

$$R \longrightarrow a(x_2).\llbracket P_1 \rrbracket \{c_1/x_1\} \mid a(x_4).\llbracket P_2 \rrbracket \{c_2/x_3\} \mid \llbracket P_3 \rrbracket$$



Lets try again

Any Ideas?

## Another approach

- Use new binding
- We need private channel for each tuple

$$\llbracket u(x_1, \dots, x_n).P \rrbracket = u(\mathbf{z}).\mathbf{z}(x_1)\dots\mathbf{z}(x_n).\llbracket P \rrbracket$$

$$\llbracket \bar{u}\langle v_1, \dots, v_n \rangle.P \rrbracket = (\nu \mathbf{c})\bar{u}\langle \mathbf{c} \rangle.\bar{\mathbf{c}}\langle v_1 \rangle\dots\bar{\mathbf{c}}\langle v_n \rangle.\llbracket P \rrbracket$$

- We still haven't finished? We need a condition ...

## Let's put it all together

|              | Monadic                                  | Polyadic                                                               |
|--------------|------------------------------------------|------------------------------------------------------------------------|
| Asynchronous | $\bar{u}\langle v \rangle$<br>$u(x).P$   | $\bar{u}\langle v_1, \dots v_n \rangle$<br>$u(x_1, \dots x_n).P_1$     |
| Synchronous  | $\bar{u}\langle v \rangle.P$<br>$u(x).P$ | $\bar{u}\langle v_1, \dots v_n \rangle.P_1$<br>$u(x_1, \dots x_n).P_1$ |

## Adding more constructs ... Choice

In the asynchronous  $\pi$ -calculus there is no built-in choice operator  $+$ . Yet, we can represent *internal nondeterminism*.

$$P \oplus Q \stackrel{\text{df}}{=} (\nu a)(\bar{a} \mid a.P \mid a.Q) \quad \text{where } a \notin \text{fn}(P|Q)$$

There are two possible reductions:

- either  $P \oplus Q \longrightarrow P \mid (\nu a)a.Q$
- or  $P \oplus Q \longrightarrow (\nu a)a.P \mid Q$

Intuitively, since  $(\nu a)a.Q$  and  $(\nu a)a.P$  cannot reduce, the processes above are equivalent respectively to  $P$  and to  $Q$ .

## Branching and Selection

- Structured external choice in the polyadic synchronous  $\pi$ -calculus:

$$P ::= \dots \mid u \triangleright \{l_1 : P_1 \parallel \dots \parallel l_n : P_n\} \mid u \triangleleft l.P$$

- We have labels (ranged over  $l, l', \dots$ )
- The branching waits for the selector to select a label
- The reduction is:

$$a \triangleright \{l_1 : P_1 \parallel \dots \parallel l_n : P_n\} \mid a \triangleleft l_k.P \longrightarrow P_k \mid P \quad (1 \leq k \leq n)$$

Is there an encoding into the polyadic synchronous  $\pi$ -calculus

## Branching and selection



The encoding of branching and selection into the polyadic synchronous  $\pi$ -calculus is defined as follows.

- $\llbracket 0 \rrbracket = 0$ ,  $\llbracket P \mid Q \rrbracket = \llbracket P \rrbracket \mid \llbracket Q \rrbracket$ ,  $\llbracket (\nu a)P \rrbracket = (\nu a)\llbracket P \rrbracket$ ,  $\llbracket !P \rrbracket = !\llbracket P \rrbracket$ ,
- $\llbracket \bar{u}\langle \tilde{v} \rangle \rrbracket = \bar{u}\langle \tilde{v} \rangle$ ,  $\llbracket u(\tilde{x}).P \rrbracket = u(\tilde{x}).\llbracket P \rrbracket$ , and

- 

$$\llbracket u \triangleright \{l_1 : P_1 \parallel l_2 : P_2\} \rrbracket = u(x).(\nu c_1, c_2)(\bar{x}\langle c_1, c_2 \rangle \mid c_1.\llbracket P_1 \rrbracket \mid c_2.\llbracket P_2 \rrbracket)$$

$$\llbracket u \triangleleft l_1.P \rrbracket = (\nu c)(\bar{u}\langle c \rangle \mid c(z_1, z_2).\bar{z}_1.\llbracket P \rrbracket)$$

$$\llbracket u \triangleleft l_2.P \rrbracket = (\nu c)(\bar{u}\langle c \rangle \mid c(z_1, z_2).\bar{z}_2.\llbracket P \rrbracket)$$

- We still haven't finished? We need a condition ...

## Recursion

- We have
  - recursive definition  $A(\tilde{x}) \stackrel{\text{df}}{=} Q$
  - a process  $P$  which uses this definition, by calling  $A\langle\tilde{v}\rangle$
- How to encode that behaviour in the asynchronous  $\pi$ -calculus ?
- Theorem to the rescue:
  - Theorem: Any process involving recursive definitions is representable using replication, and conversely replication is redundant in the presence of recursion.
  - Tricky: we also need restriction

## Recursion vs Replication

Using replication and restriction we can encode recursive definitions. Suppose we have the recursive definition  $A(\tilde{x}) \stackrel{\text{df}}{=} Q$  and a process  $P$  which uses this definition, by calling  $A\langle\tilde{v}\rangle$ . We can encode this behaviour in the asynchronous  $\pi$ -calculus as follows:

1. choose a fresh channel name  $a$  not occurring in  $P$  or  $Q$ ;
2. let  $P_a$  and  $Q_a$  be  $P$  and  $Q$ , where each recursive call of the form  $A\langle\tilde{v}\rangle$  is replaced by an output process  $\bar{a}\langle\tilde{v}\rangle$ ;
3. replace  $P$  by  $(\nu a)(P_a \mid !a(\tilde{x}).Q_a)$

For example, consider the recursive definition and the reduction

$$\begin{aligned} \text{BufferNext}(x) &\stackrel{\text{df}}{=} x(y).(\bar{x}\langle y \rangle \mid \text{BufferNext}\langle y \rangle) \\ \bar{b}\langle c \rangle \mid \text{BufferNext}\langle b \rangle &\longrightarrow \bar{b}\langle c \rangle \mid \text{BufferNext}\langle c \rangle \end{aligned}$$

with their non-recursive version

$$\begin{aligned} \text{BufferNext}_a &\stackrel{\text{df}}{=} x(y).(\bar{x}\langle y \rangle \mid \bar{a}\langle y \rangle) \\ (\nu a)(\bar{b}\langle c \rangle \mid \bar{a}\langle b \rangle \mid !a(x).\text{BufferNext}_a) &\xrightarrow{*} (\nu a)(\bar{b}\langle c \rangle \mid \bar{a}\langle c \rangle \mid !a(x).\text{BufferNext}_a) \end{aligned}$$