

# Interpreting the Full $\lambda$ -Calculus in the $\pi$ -Calculus

Xiaojuan Cai

Joint work with Yuxi Fu

BASICS Lab

October 12, 2009



- 1 Technical background
- 2 Encoding the full  $\lambda$ -calculus into the  $\pi$ -calculus
  - Interpretation via tree structure
  - Tree structure in the  $\pi$ -calculus
- 3 How does the encoding work?
- 4 Conclusion













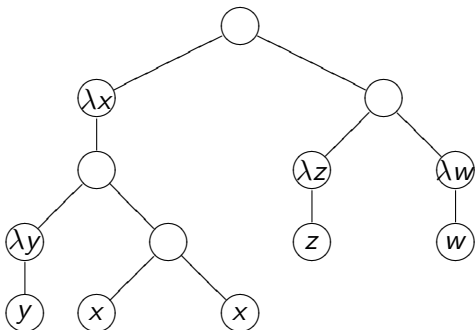








$\lambda$ -rules
$$(\lambda x. (\lambda y. y)(xx))((\lambda z. z)(\lambda w. w))$$

$$\rightarrow (\lambda y. y)((\lambda z. z)(\lambda w. w))$$
 $\beta$ -rule
$$(\lambda x. M)N \rightarrow M\{N/x\}$$


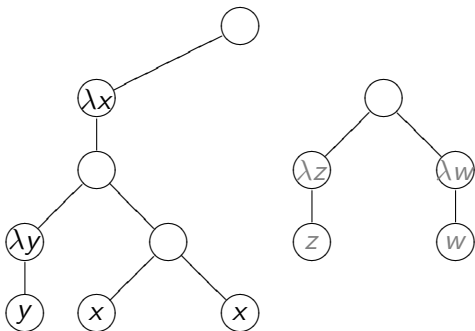
# $\lambda$ -rules

$$(\lambda x. (\lambda y. y)(xx))((\lambda z. z)(\lambda w. w))$$

$$\rightarrow (\lambda y. y)((\lambda z. z)(\lambda w. w))((\lambda z. z)(\lambda w. w))$$

 $\beta$ -rule

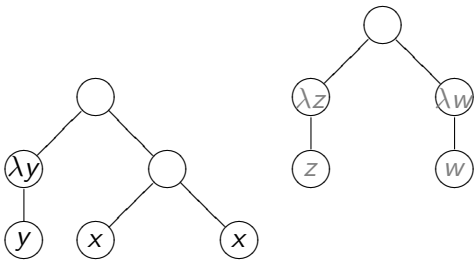
$$(\lambda x. M)N \rightarrow M\{N/x\}$$



# $\lambda$ -rules

$$\begin{aligned}
 & (\lambda x. (\lambda y. y)(xx))((\lambda z. z)(\lambda w. w)) \\
 \rightarrow & (\lambda y. y)((\lambda z. z)(\lambda w. w))((\lambda z. z)(\lambda w. w))
 \end{aligned}$$

$\beta$ -rule  
 $(\lambda x. M)N \rightarrow M\{N/x\}$



Interpretation via tree structure

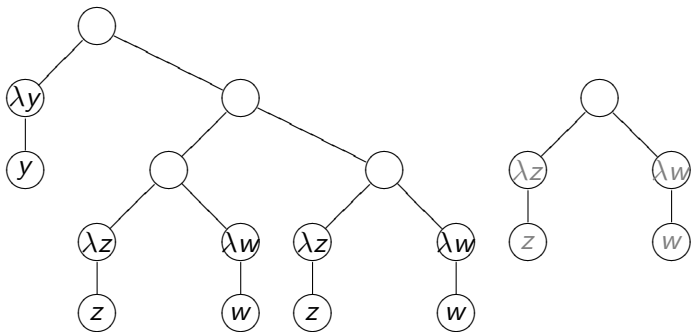
# $\lambda$ -rules

$$(\lambda x. (\lambda y. y)(xx))((\lambda z. z)(\lambda w. w))$$

$$\rightarrow (\lambda y. y)((\lambda z. z)(\lambda w. w))((\lambda z. z)(\lambda w. w))$$

$\beta$ -rule

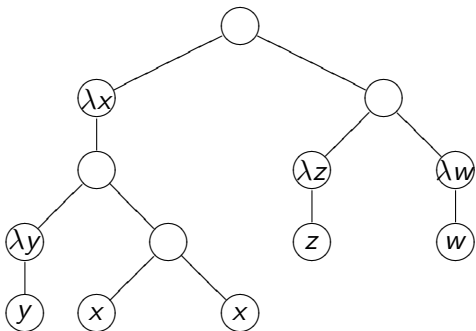
$$(\lambda x. M)N \rightarrow M\{N/x\}$$



# $\lambda$ -rules

$(\lambda x. (\lambda y. y)(xx))((\lambda z. z)(\lambda w. w))$

Eager evaluation
$\frac{N \rightarrow N'}{MN \rightarrow MN'}$

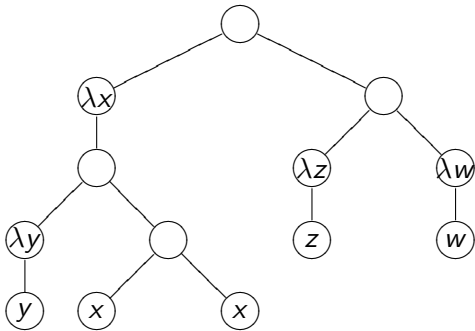


# $\lambda$ -rules

$$(\lambda x. (\lambda y. y)(xx))((\lambda z. z)(\lambda w. w))$$

$$\longrightarrow (\lambda x. (\lambda y. y)(xx))(\lambda w. w)$$

Eager evaluation
$\frac{N \longrightarrow N'}{MN \longrightarrow MN'}$

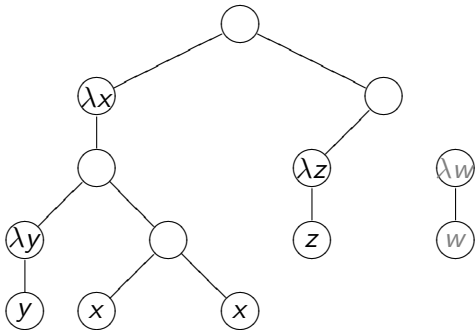


# $\lambda$ -rules

$$(\lambda x. (\lambda y. y)(xx))((\lambda z. z)(\lambda w. w))$$

$$\longrightarrow (\lambda x. (\lambda y. y)(xx))(\lambda w. w)$$

Eager evaluation
$\frac{N \longrightarrow N'}{MN \longrightarrow MN'}$



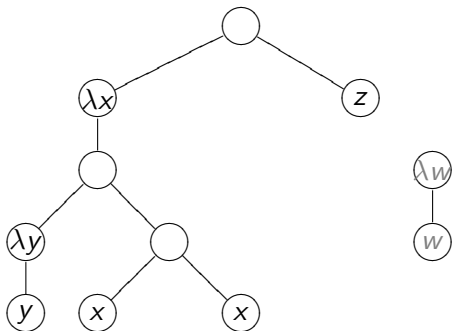
Interpretation via tree structure

# $\lambda$ -rules

$$(\lambda x. (\lambda y. y)(xx))((\lambda z. z)(\lambda w. w))$$

$$\longrightarrow (\lambda x. (\lambda y. y)(xx))(\lambda w. w)$$

Eager evaluation
$\frac{N \longrightarrow N'}{MN \longrightarrow MN'}$

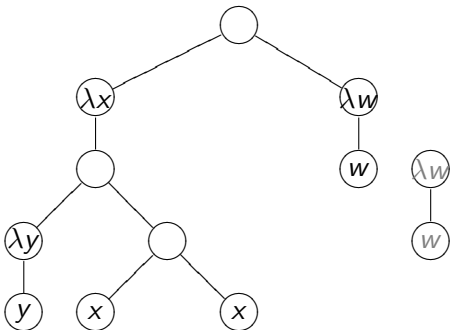


# $\lambda$ -rules

$$(\lambda x. (\lambda y. y)(xx))((\lambda z. z)(\lambda w. w))$$

$$\longrightarrow (\lambda x. (\lambda y. y)(xx))(\lambda w. w)$$

Eager evaluation
$\frac{N \longrightarrow N'}{MN \longrightarrow MN'}$

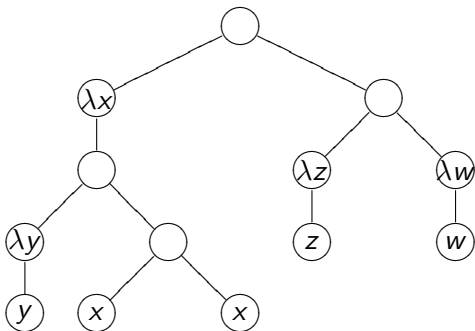


# $\lambda$ -rules

$(\lambda x. (\lambda y. y)(xx))((\lambda z. z)(\lambda w. w))$

Partial evaluation

$$\frac{M \rightarrow M'}{\lambda. M \rightarrow \lambda. M'}$$



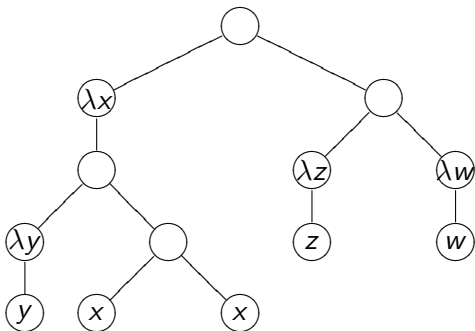
$\lambda$ -rules

$$(\lambda x. (\lambda y. y)(xx))((\lambda z. z)(\lambda w. w))$$

$$\longrightarrow (\lambda x. xx)((\lambda z. z)(\lambda w. w))$$

Partial evaluation

$$\frac{M \longrightarrow M'}{\lambda. M \longrightarrow \lambda. M'}$$

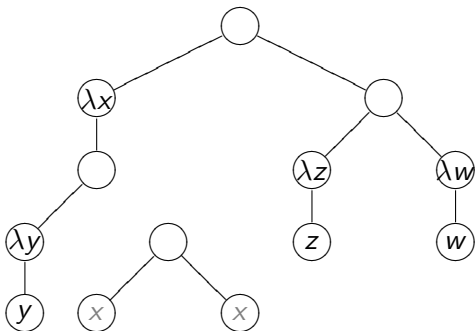


# $\lambda$ -rules

$$(\lambda x. (\lambda y. y)(xx))((\lambda z. z)(\lambda w. w))$$

$$\longrightarrow (\lambda x. xx)((\lambda z. z)(\lambda w. w))$$

Partial evaluation
$\frac{M \longrightarrow M'}{\lambda. M \longrightarrow \lambda. M'}$

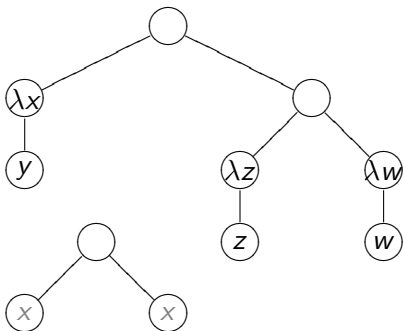


# $\lambda$ -rules

$$(\lambda x. (\lambda y. y)(xx))((\lambda z. z)(\lambda w. w))$$

$$\longrightarrow (\lambda x. xx)((\lambda z. z)(\lambda w. w))$$

Partial evaluation $\frac{M \longrightarrow M'}{\lambda. M \longrightarrow \lambda. M'}$
---

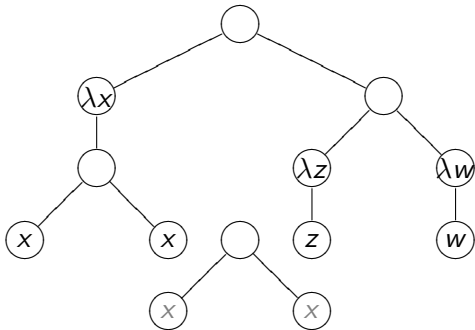


# $\lambda$ -rules

$$(\lambda x. (\lambda y. y)(xx))((\lambda z. z)(\lambda w. w))$$

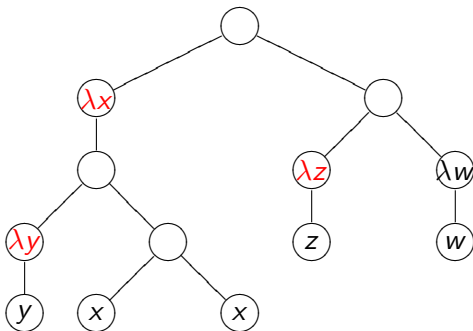
$$\longrightarrow (\lambda x. xx)((\lambda z. z)(\lambda w. w))$$

Partial evaluation $\frac{M \longrightarrow M'}{\lambda. M \longrightarrow \lambda. M'}$
---



# Redex node

$$(\lambda x. (\lambda y. y)(xx))((\lambda z. z)(\lambda w. w))$$



# Interpretation via tree structure

- $\lambda$ -term  $\implies$  tree
- $\beta$ -reduction  $\implies$  manipulation of trees

Let's program in  $\pi$ !

# Outline design

$Tree := Node_1 | Node_2 | \dots | Node_n$

$Node := \bar{n}\langle info \rangle + Operations$

$Operations := reduction$  if the node is a **redex node**  
 $substitution$  if the node is a **variable node**

$reduction :=$  (change the structure).

(freeze and backup the disconnected subtree)

$substitution := \bar{x}\langle info \rangle$

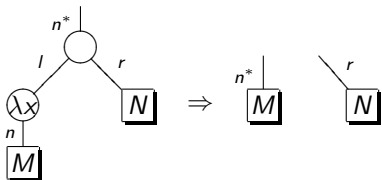
$info := name, parent, leftchild, rightchild, \dots$

# Node info

<i>name</i>	<i>parent</i>	<i>lchild</i>	<i>rchild</i>	<i>var</i>	<i>fun</i>
-------------	---------------	---------------	---------------	------------	------------

- $lchild = rchild = \perp$ . **variable node**
- $lchild = rchild \neq \perp$ . **abstraction node**
- $lchild \neq rchild$ . **application node**
- $lchild = rchild \neq \perp \wedge fun = \top$ . **redex node**

# Node operations - move



$$Move(n, n^*) \stackrel{\text{def}}{=} n(p, l, r, v, f).n^*(p^*, l^*, r^*, v^*, f^*).$$

$$(Node(n^*, p^*, l, r, v, f^*) |$$

**begin case**

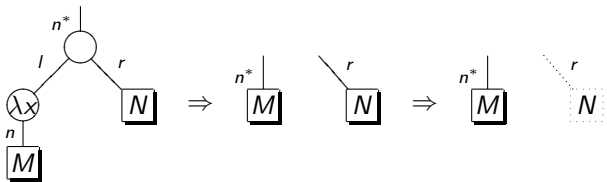
$$l = r \neq \perp \Rightarrow l(p', l', r', v', f').Node(l, n^*, l', r', v', f')$$

$$l \neq r \Rightarrow l(p', l', r', v', f').r(p'', l'', r'', v'', f'').$$

$$(Node(l, n^*, l', r', v', f') | Node(r, n^*, l'', r'', v'', f''))$$

**end case)**

# Node operations - freeze and backup



$$\text{Freeze}(r, x) \stackrel{\text{def}}{=} r(p, l, r, v, f).(x_0 x_1)$$

**begin case**

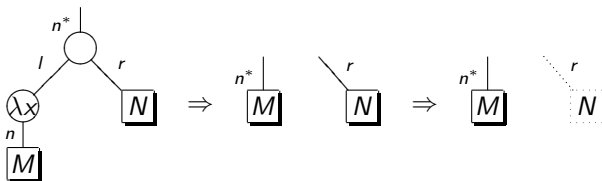
$$l \neq r \Rightarrow \text{Freeze}(l, x_0) \mid \text{Freeze}(r, x_1) \mid !x(p', n', f').(l' r')(Node(n', p', l', r', v, f')) \mid \dots$$

$$l = r \neq \perp \Rightarrow \text{Freeze}(l, x_0) \mid !x(p', n', f').\dots$$

$$l = r = \perp \Rightarrow !x(p', n', f').Node(n', p', \perp, \perp, v, f')$$

**end case**

# Node operations - freeze and backup



$$\text{Freeze}(r, x) \mid \text{Tree}(r, N) \Longrightarrow \text{Replica}(x, N)$$

$$\stackrel{\text{def}}{=} (x_0 x_1 \cdots x_n)$$

$$\mid !x(p', n', f').(l' r')(Node(n', p', l', r', v, f') \mid \bar{x}_0 \langle n', l', \top \rangle \mid \bar{x}_1 \langle n', r', \perp \rangle)$$

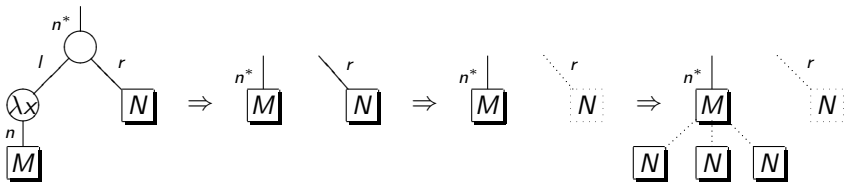
$$\mid !x_0(p'', n'', f'').(l'' r'') \dots$$

$$\mid !x_1(p''', n''', f''').(l''' r''') \dots$$

...

Tree structure in the  $\pi$ -calculus

# Node operations - substitute



$$\text{Substitution}(m, p, x, f) \stackrel{\text{def}}{=} \bar{x}\langle p, m, f \rangle$$

$$\text{Substitution}(m, p, x, f) | \text{Replica}(x, N) \implies \text{Tree}(m, N) | \text{Replica}(x, N)$$

# The encoding

$\llbracket M \rrbracket_\lambda$	$\stackrel{\text{def}}{=}$	$(n, s, \top)(\llbracket M \rrbracket_{n,\lambda}^\perp \mid \text{Sem}(s))$
$\llbracket x \rrbracket_{n,p}^f$	$\stackrel{\text{def}}{=}$	$L(n, p, \perp, \perp, x, f)$
$\llbracket \lambda x. M \rrbracket_{n,p}^f$	$\stackrel{\text{def}}{=}$	$(m, x)(L(n, p, m, m, x, f) \mid \llbracket M \rrbracket_{m,n}^\perp)$
$\llbracket MN \rrbracket_{n,p}^f$	$\stackrel{\text{def}}{=}$	$(l, r)(L(n, p, l, r, \perp, f) \mid \llbracket M \rrbracket_{l,n}^\top \mid \llbracket N \rrbracket_{r,n}^\perp)$

$$\text{Sem}(s) \stackrel{\text{def}}{=} \bar{s}.s.\text{Sem}(s)$$

Tree structure in the  $\pi$ -calculus

# The encoding — the definition of node process

$L(n, p, l, r, v, f)$	$= \bar{n}(p, l, r, v, f) + \text{begin case}$ $l = r = \perp \Rightarrow (a)\bar{v}(n, p, f, a).\bar{a}(\top, \top, \top)$ $l = r \neq \perp \wedge f = \top \Rightarrow \text{Abs}(n, p, l, v, \top)$ $l = r \neq \perp \wedge p = \lambda \Rightarrow \text{RAbs}(n, \lambda, l, v, f)$ $\text{end case}$
$\text{Abs}(n, p, m, x, f)$	$\stackrel{\text{def}}{=} s.p(p^1, l^1, r^1, v^1, f^1).m(p_1, l_1, r_1, v_1, f_1).(L(p, p^1, l_1, r_1, v_1, f^1)  $ $\text{begin case}$ $l_1 = r_1 \neq \perp \Rightarrow l_1(p_2, l_2, r_2, v_2, f_2).(L(l_1, p, l_2, r_2, v_2, f_2)$ $  (b)(\text{Backup}(r^1, x, b, \perp)   b.\bar{s}))$ $l_1 \neq r_1 \Rightarrow l_1(p_2, l_2, r_2, v_2, f_2).r_1(p'_2, l'_2, r'_2, v'_2, f'_2).(L(l_1, p, l_2, r_2, v_2, f_2)$ $  L(r_1, p, l'_2, r'_2, v'_2, f'_2)   (b)(\text{Backup}(r^1, x, b, \perp)   b.\bar{s}))$ $\text{end case})$
$\text{RAbs}(n, \lambda, m, x, f)$	$\stackrel{\text{def}}{=} \lambda(z).s.m(p_1, l_1, r_1, v_1, f_1).(b)(\text{Backup}(z, x, b, \top)   b.\bar{s}.L(m, \lambda, l_1, r_1, v_1, \perp))$
$\text{Backup}(n, x, b, e)$	$= n(p, l, r, v, f).$ $\text{begin case}$ $l = \perp \vee r = \perp \Rightarrow \bar{b}.!x(n', p', f', a).(o)(\text{Find}(v, a, o)   o(v')).$ $\text{begin case}$ $v' = \perp \wedge e = \top \Rightarrow \tau.[\Omega]_{n'p'}^{f'}$ $v' = \perp \wedge e = \perp \Rightarrow \tau.L(n', p', \perp, \perp, v, f')$ $v' \neq \perp \Rightarrow \tau.L(n', p', \perp, \perp, v', f')$ $\text{end case}$ $l = r \neq \perp \Rightarrow \tau.(x'b')( \text{Backup}(l, x', b', e)   b'.\bar{b}.!x(n', p', f', a).$ $(m'a'v')(L(n', p', m', m', v', f')   \bar{x}'(m', n', \perp, a')   \bar{a}'(v, v', a)))$ $\perp \neq l \neq r \neq \perp \Rightarrow \tau.(x_0x_1b_0b_1)( \text{Backup}(l, x_0, b_0, e)   \text{Backup}(r, x_1, b_1, e)$ $  b_0.b_1.\bar{b}.!x(n', p', f', a).(l'r')(L(n', p', l', r', \perp, f')$ $  \bar{x}_0(l', n', \top, a)   \bar{x}_1(r', n', \perp, a)))$ $\text{end case}$



# Correctness — full abstraction

**Theorem 1.** For all closed  $\lambda$ -terms  $M, N$ ,  $M =_a N$  if and only if  $\llbracket M \rrbracket_\lambda \approx \llbracket N \rrbracket_\lambda$ .

## Definition

A symmetric binary relation  $\mathcal{R}$  on  $\Lambda^0$  is an **applicative bisimulation** if the following properties hold whenever  $M\mathcal{R}N$ :

- If  $M \rightarrow^* \lambda x.M'$  then  $N \rightarrow^* \lambda x.N'$  for some  $N'$  such that  $M'\{L/x\}\mathcal{R}N'\{L/x\}$  for every  $L \in \Lambda^0$ .

The applicative bisimilarity  $=_a$  is the largest applicative bisimulation.

## Correctness — operational correspondence

**Theorem 2.** There is a **subbisimilarity**  $\mathcal{R}$  from the set of closed  $\lambda$ -terms to the set of  $\pi$ -processes.

**Preservation** If  $M \longrightarrow M'$ , then  $\llbracket M \rrbracket \xRightarrow{\tau} \equiv \llbracket M' \rrbracket$ ;

**Reflection** If  $\llbracket M \rrbracket \xrightarrow{\tau} P'$ , then there exists some  $M'$  such that  $M \longrightarrow M'$  and  $P' \Longrightarrow \equiv \llbracket M' \rrbracket$ ;

**Termination**  $M \uparrow$  iff  $\llbracket M \rrbracket \uparrow$ ;

- ① Technical background
  
- ② Encoding the full  $\lambda$ -calculus into the  $\pi$ -calculus
  - Interpretation via tree structure
  - Tree structure in the  $\pi$ -calculus
  
- ③ How does the encoding work?
  
- ④ Conclusion

# Conclusion

## What we have done

- An encoding from the full  $\lambda$ -calculus to the  $\pi$ -calculus with mismatch and parametric definition is proposed;
- It is proved to be *good*.

## Question

- Can the encoding be implemented in the  $\pi$ -calculus without **guarded choice**?

Thank you!

Questions?

Operational semantics of  $\pi$ -calculus I*Prefix*

$$\frac{}{a(x).P \xrightarrow{ay} P\{y/x\}} \quad \frac{}{\bar{a}x.P \xrightarrow{\bar{a}x} P}$$

*Composition*

$$\frac{P \xrightarrow{\mu} P'}{P | Q \xrightarrow{\mu} P' | Q} \quad \frac{P \xrightarrow{ax} P' \quad Q \xrightarrow{\bar{a}x} Q'}{P | Q \xrightarrow{\tau} P' | Q'}$$

$$\frac{P \xrightarrow{ax} P' \quad Q \xrightarrow{\bar{a}(x)} Q'}{P | Q \xrightarrow{\tau} (x)(P' | Q')}$$

*Restriction*

$$\frac{P \xrightarrow{\bar{a}x} P'}{(x)P \xrightarrow{\bar{a}(x)} P'} \quad \frac{P \xrightarrow{\mu} P' \quad x \text{ not in } \mu}{(x)P \xrightarrow{\mu} (x)P'}$$

Operational semantics of  $\pi$ -calculus II*Replication*

$$\frac{P \mid !P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'}$$

*Case*

$$\frac{\varphi_i \Leftrightarrow \top \quad \mu_i.P_i \xrightarrow{\mu'_i} P'_i}{\sum_{i \in I} \varphi_i \mu_i.P_i \xrightarrow{\mu'_i} P'_i}$$

*Definition*

$$\frac{P_i\{\tilde{y}_i/\tilde{x}_i\} \xrightarrow{\mu} P'}{D_i(\tilde{y}_i) \xrightarrow{\mu} P'}$$

# Definition: subbisimilarity

$\mathcal{R} \subseteq \Lambda^0 \times \mathcal{P}$  is a **subbisimilarity** if :

- ①  $\forall M \in \Lambda^0. \exists P. M \mathcal{R} P$ ;
- ② If  $M \mathcal{R} P$  and  $M \rightarrow M'$  then  $P \xrightarrow{\tau} P'$  and  $M' \mathcal{R} P'$  for some  $P'$ ;
- ③ If  $M \mathcal{R} P \xrightarrow{\tau} P'$  then  $M \mathcal{R} P'$  or  $M \rightarrow M' \mathcal{R} P'$  for some  $M'$ ;
- ④ If  $M \mathcal{R} P_0$  and  $P_0 \xrightarrow{\tau} P_1 \cdots \xrightarrow{\tau} P_i \xrightarrow{\tau} \cdots$  is an infinite sequence of  $\tau$ -actions, then there must be some  $k \geq 1$  and  $M'$ , such that  $M \rightarrow M' \mathcal{R} P_k$ ;
- ⑤ If  $(\lambda x. M) \mathcal{R} P$ , then  $P \Longrightarrow P' \xrightarrow{\lambda z} P''$  for some  $P', P''$  and some fresh  $z$  such that  $\lambda x. M \mathcal{R} P'$  and  $M\{N/x\} \mathcal{R}(z)(P'' \mid T(z, N))$  for every closed  $\lambda$ -term  $N$ .
- ⑥ If  $M \mathcal{R} P$  and  $P \xrightarrow{\lambda z} P'$  for some fresh  $z$ , then  $M \equiv \lambda x. M'$  for some  $x, M'$  such that  $M' \{N/x\} \mathcal{R}(z)(P' \mid T(z, N))$  for every closed  $\lambda$ -term  $N$ .

# References

- 1 R. Milner. Functions as Processes. *Mathematical Structures in Computer Science*, 2:119C146, 1992.
- 2 D. Sangiorgi. The Lazy  $\lambda$ -Calculus in a Concurrency Scenario. *Information and Computation*, 111:120C 153, 1994.